* **Azure Monitor**
  + Azure Metrics to create metrics for various resources
  + Azure Activity Logs to see events performed in Azure env
  + Azure Alert to monitor various type of actions or status of VM

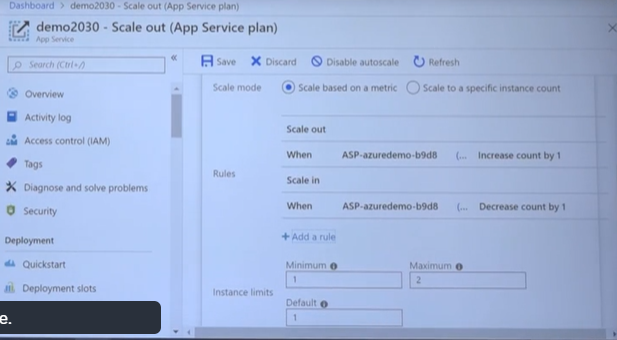
Create Rule:

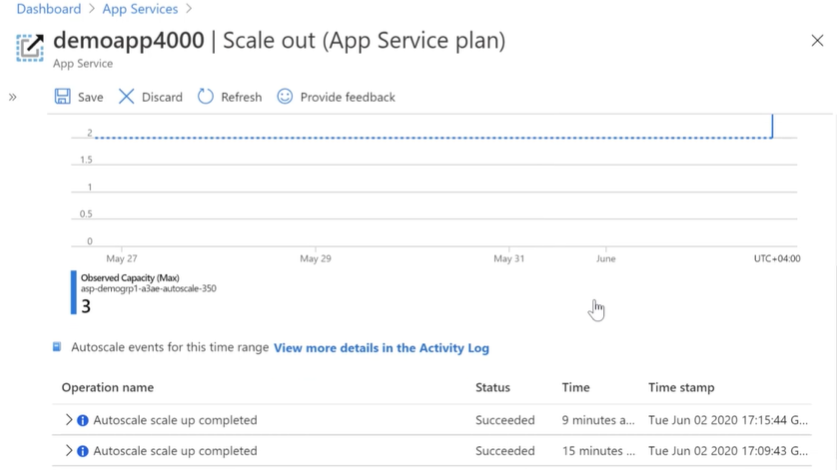
1. Create Resource
2. Create Rule based on various category(Metric,ActivityLog)
3. Create Action Group

* Azure CLI command

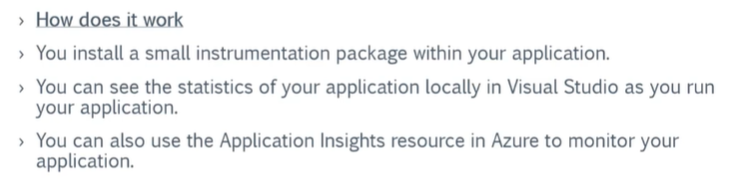
az monitor metrics alert create -n demoalert -g demogrp1 --scopes /subscriptions/20c6eec9-2d80-4700-b0f6-4fde579a8783/resourceGroups/demogrp1/providers/Microsoft.Compute/virtualMachines/linuxvm --condition "avg Percentage CPU > 80" --window-size 5m --evaluation-frequency 1m --description "High CPU"

* **AutoScaling WebAPP**
  + Scale out option is used to enable autoscaling based on rules
  + A rule can be added using metrics source of current resource(i.e memory of VM) but it can also be done using other resources(i.e storage queue exceeds 3 messages at a time)
  + This feature is only possible with standard app service plan + level
  + Scale Up and Scale down rules can be added for better cost and efficiency





* **Application Insight**



**Overview**

* This is an application Performance Management service for web developers.
* You can use this tool to monitor your applications.
* It can help developers detect anomalies in the application.
* It can help diagnose issues.
* It can also help understand how users use your application.
* It also helps you improve performance and usability of your application.

**Different aspects that get monitored with Application Insights**

* Request rates, the response times and failure rates – This is done at the page level.
* Exception recorded by your application.
* Page views and their load performance as reported from the user’s browser.
* User and session counts.
* Performance counters of the underlying Windows or Linux Machines.
* Diagnostic trace logs from your application.
* Any custom events or metrics that the developer writes themselves in the code

**Different ways to see how your site is behaving**

**Funnels**

* You can create a funnel from one stage to another stage of your application.
* You can then see how users are progressing through the stages of the funnel.

**User Flows**

* This helps visualize how users navigate between pages in your site. This can help answer question such as
* Does the user navigate away from a page on your site
* What do users click on a page on your site
* Where are the places where users churn most on your site
* Are there places where users repeat the same action over and over

**Impact**

* This helps decide if a page is having an impact on your application.
* It can help answer the question as to whether the page load time is impacting how many people convert on a page in the application.

**Retention**

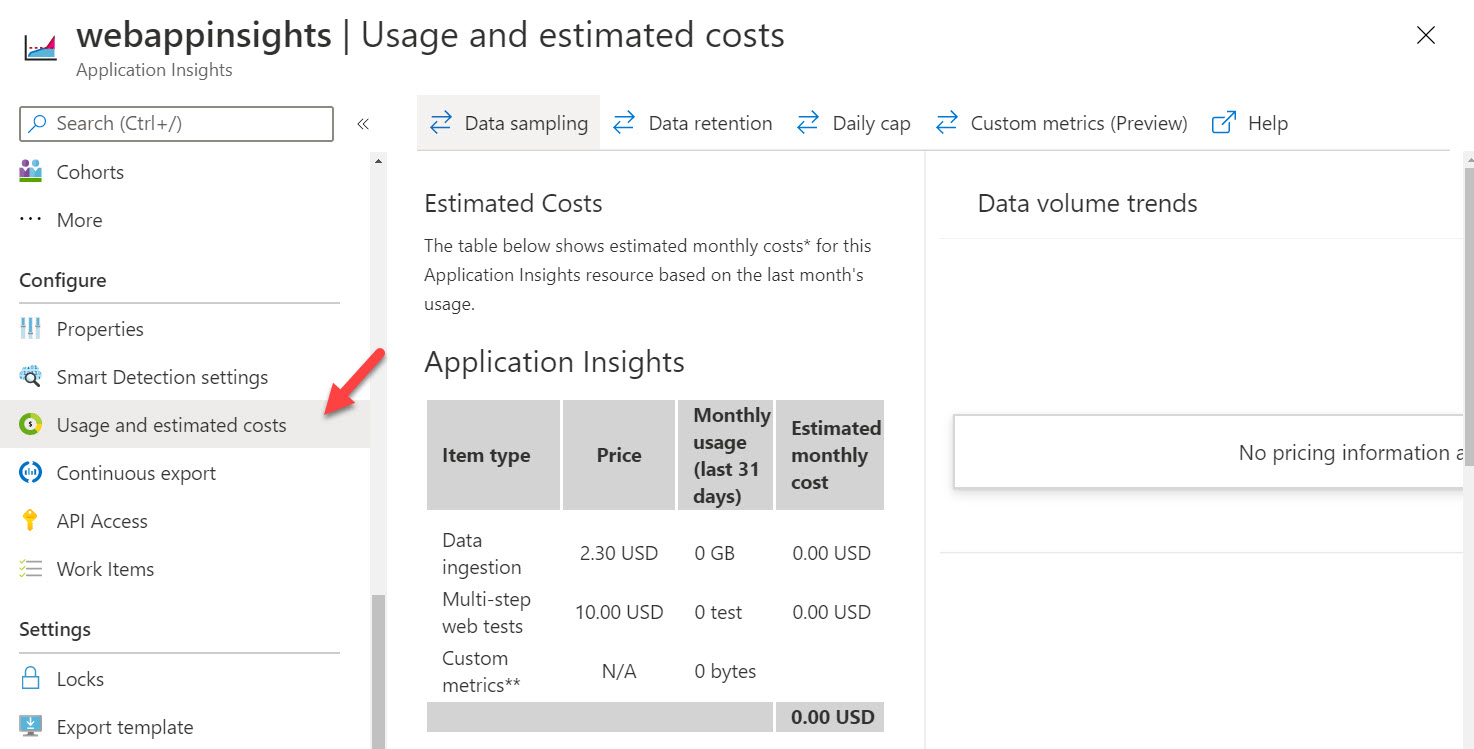
* This helps you understand how many users return to your application.
* It can also help understand if users are able to perform certain tasks in your application.

**Availability**

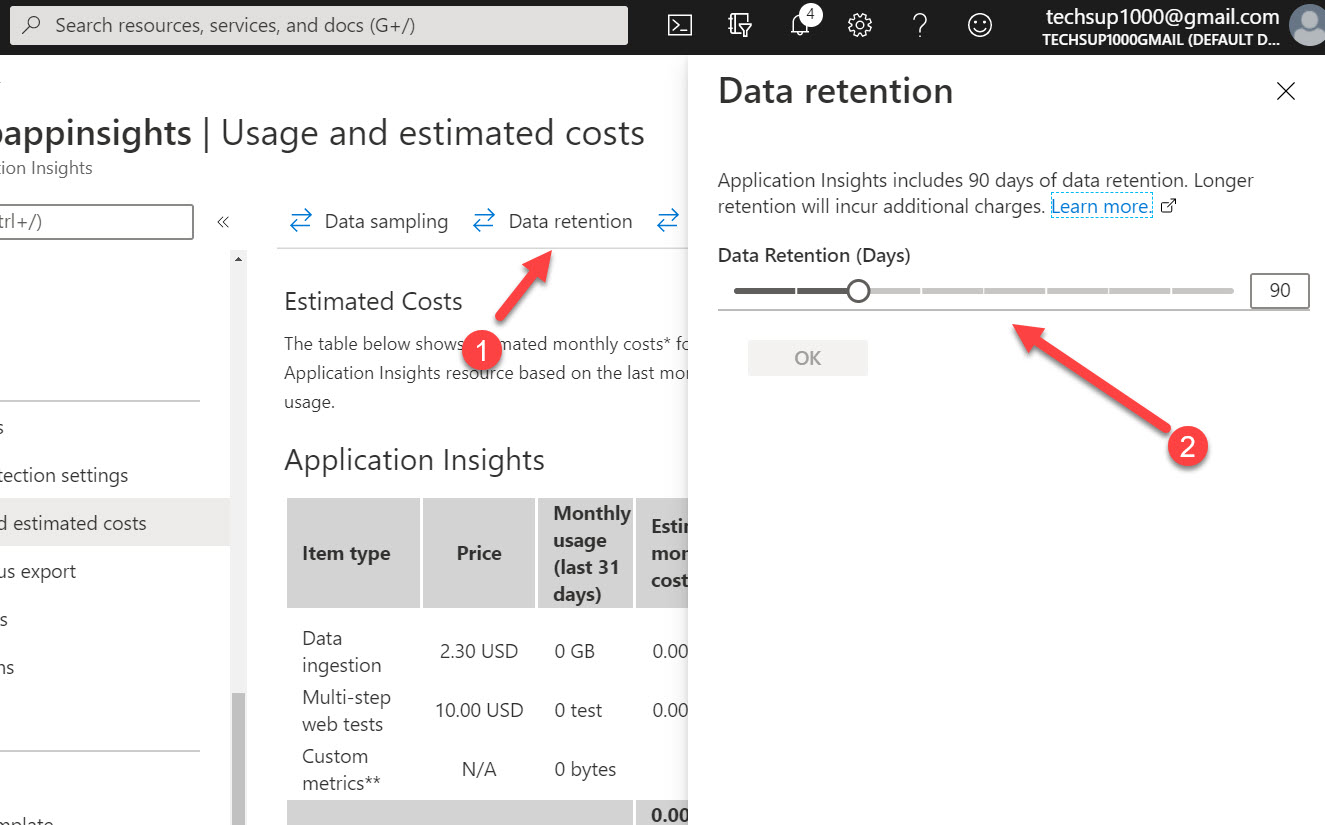
* A test can be created to ping the url from various locations and report can be seen

**Usage and estimation costs**

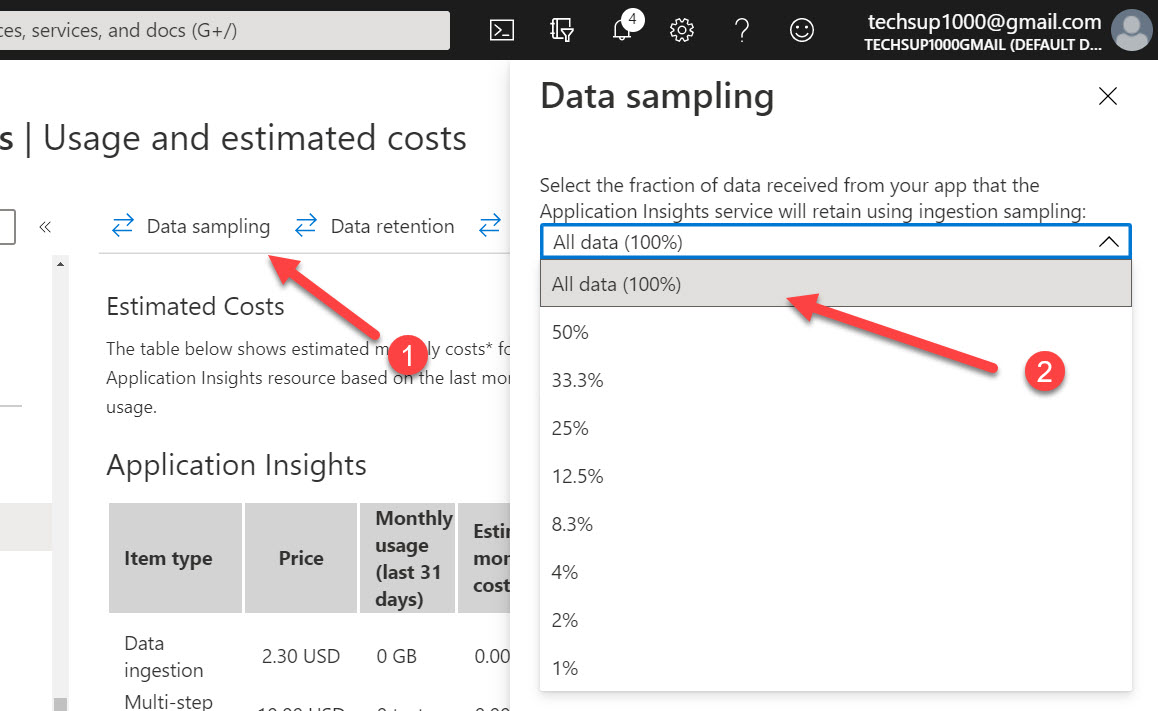
If you go to your application insights resource and go to Usage and estimated costs, you will see your usage and estimated costs for using application insights



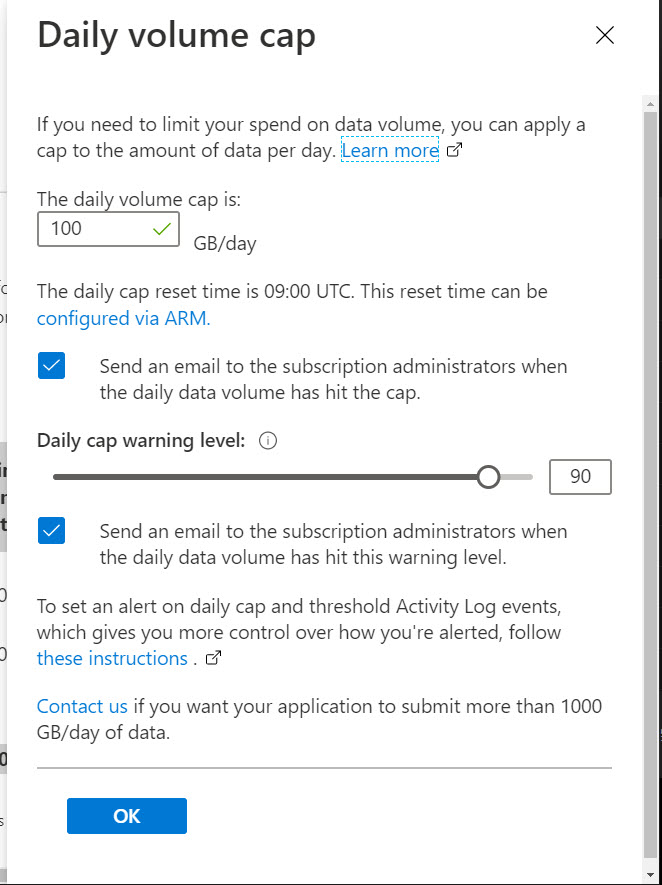
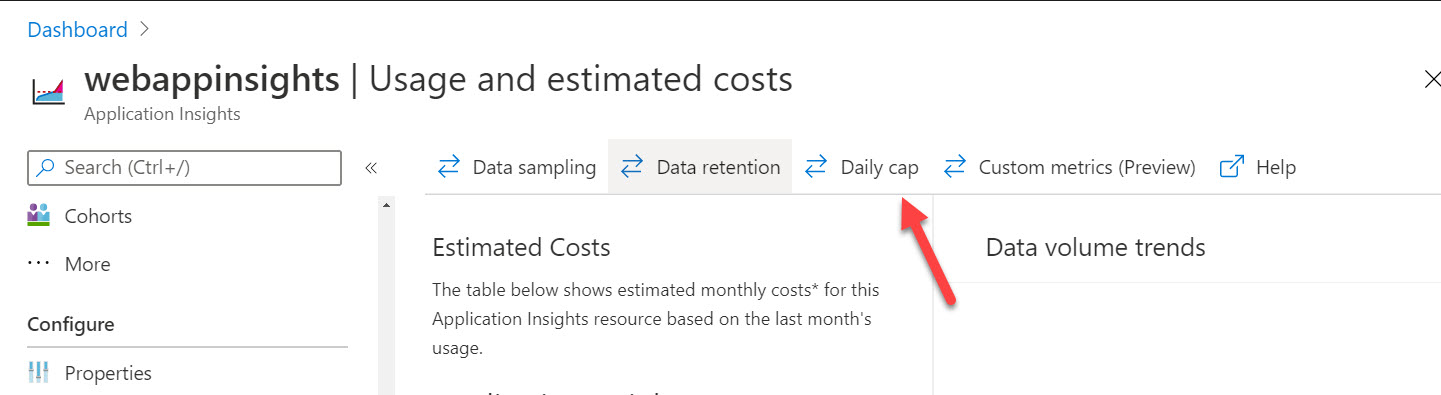
If you go onto Data retention, you can decide for how long you want to retain the data collected by Application Insights. The maximum retention period is 730 days.



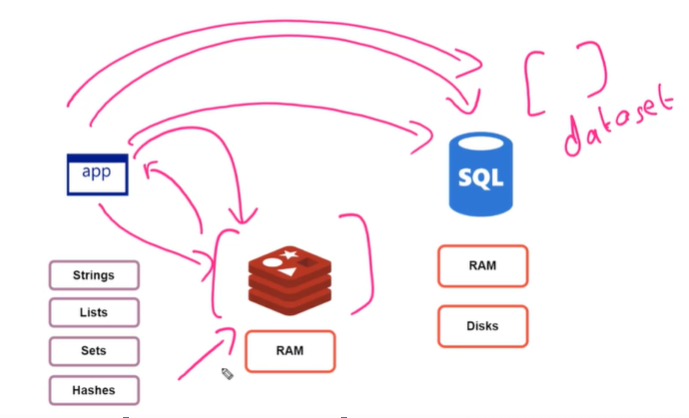
If you go to Data sampling, you can decide how much data you want to be sent onto Application Insights. Sometimes if you have massive amounts of data being generated , it might slow down your application. In such cases , you can send data samples instead onto Application Insights



You can also set a Daily cap on what is the maximum amount of data that can be collected by Application Insights on a daily basis



* **Azure Cache for redis**



// GET: Products

public async Task<IActionResult> Index()

{

List<Product> lst = new List<Product>();

var \_cache\_obj = \_cache.GetString("Products");

if (string.IsNullOrEmpty(\_cache\_obj))

{

lst = \_context.Products.ToList();

/\*

\* Use the following statements if you want to set an expiration time for the redis keys

DistributedCacheEntryOptions options = new DistributedCacheEntryOptions();

options.SetAbsoluteExpiration(TimeSpan.FromSeconds(30));

\_cache.SetString("Products", JsonConvert.SerializeObject(lst),options);

\*/

\_cache.SetString("Products", JsonConvert.SerializeObject(lst));

}

else

{

lst = JsonConvert.DeserializeObject<List<Product>>(\_cache\_obj);

}

return View(lst);

}

**Working with \_cache :**

static void Main(string[] args)

{

IDatabase l\_cache = redisconn.Value.GetDatabase();

//Set a value in the redis database

//l\_cache.StringSet("AvgOrders", "120");

// If you want to set an expiration for the keys

//l\_cache.KeyExpire("AvgOrders", new TimeSpan(0, 0, 30));

// To delete a key

//l\_cache.KeyDelete("AvgOrders");

//Console.WriteLine("The Average Daily Orders are " + l\_cache.StringGet("AvgOrders"));

// You can also work with class objects

// Add the object to the cache

Customer obj = new Customer(10, "userA");

l\_cache.StringSet("obj1", JsonConvert.SerializeObject(obj));

// Get the object from the cache

Customer newobj = JsonConvert.DeserializeObject<Customer>(l\_cache.StringGet("obj1"));

Console.WriteLine("The Customer ID is " + newobj.customerID);

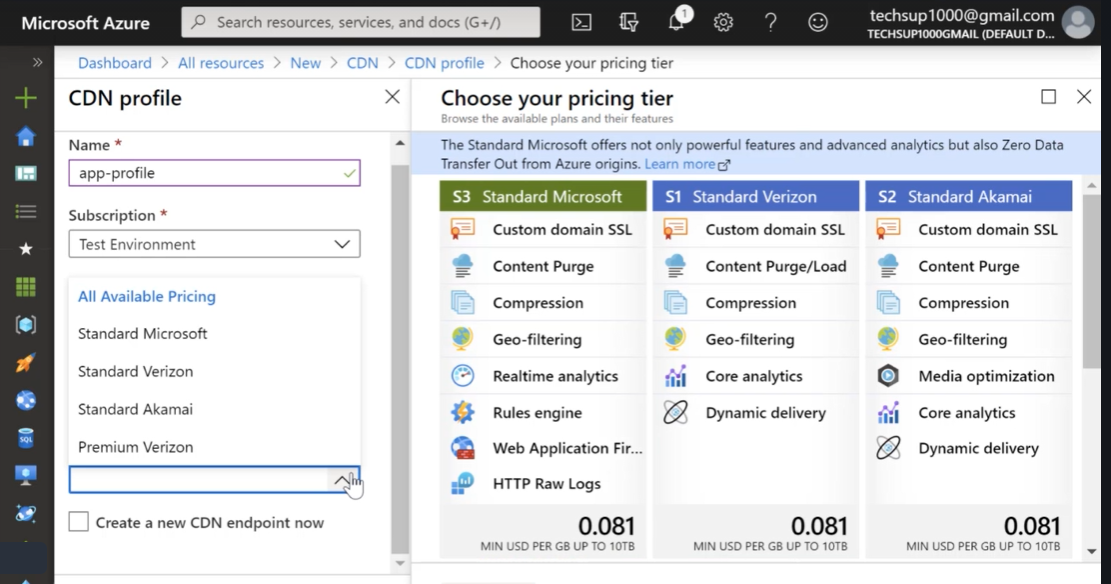
Console.WriteLine("The Customer Name is " + newobj.customerName);

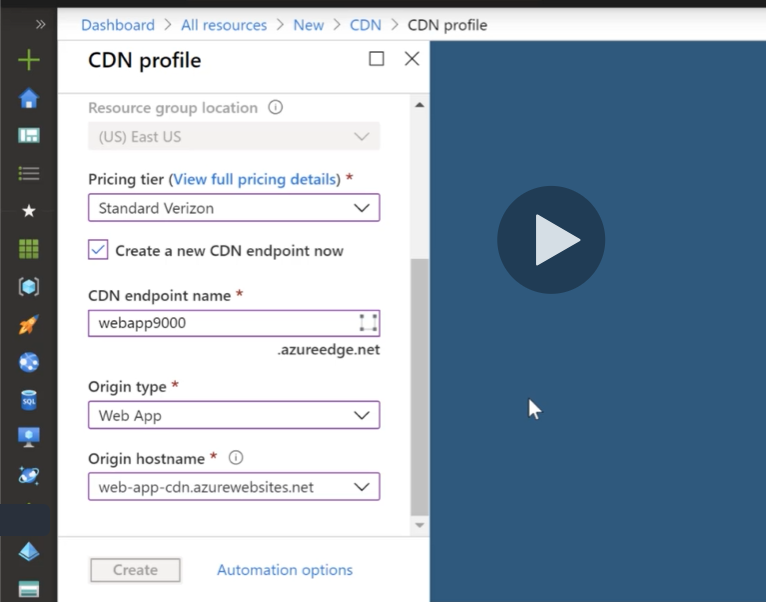
redisconn.Value.Dispose();

Console.ReadKey();

}

* **Azure Content delivery network**

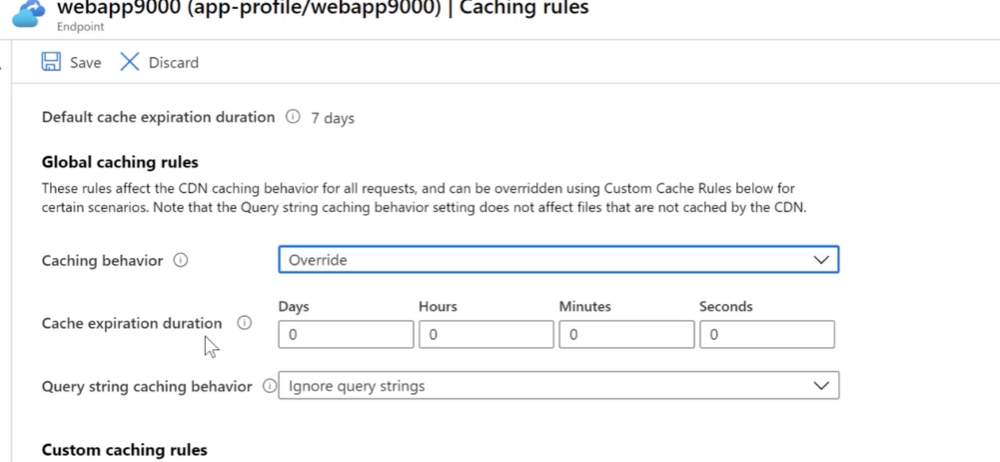


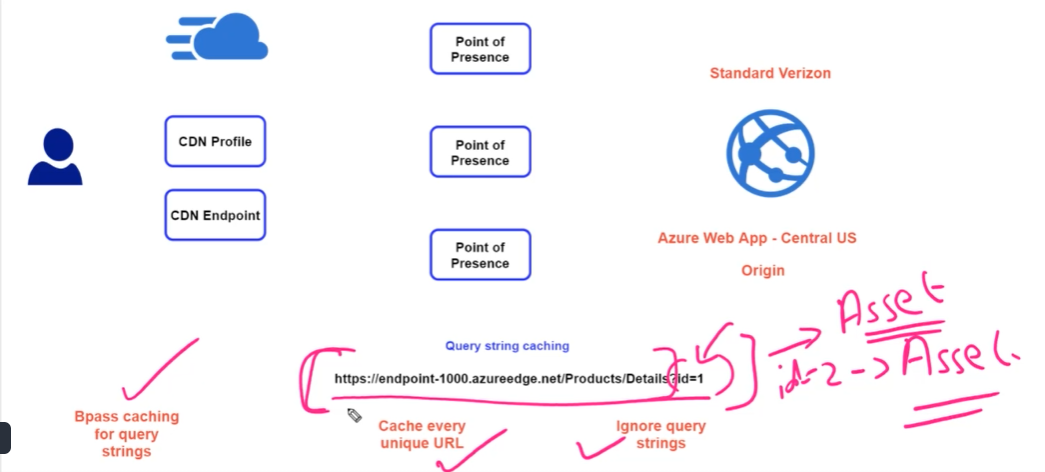


**Create Availability Tests to execute the web app page to see performance difference.**

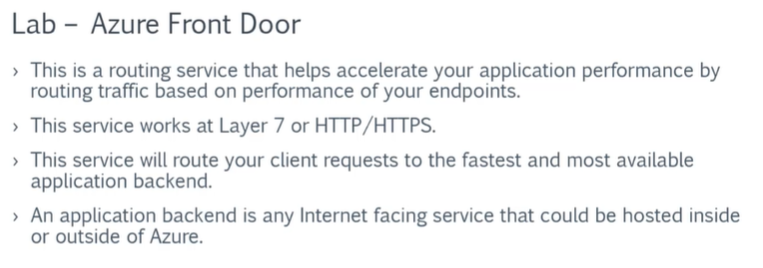
**Point of presence device available in each region helps improve performance**

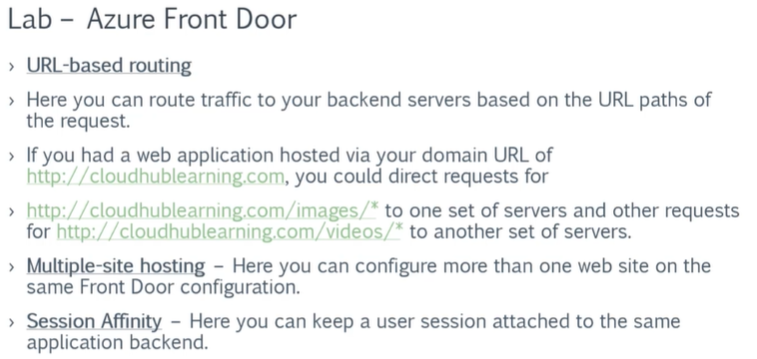


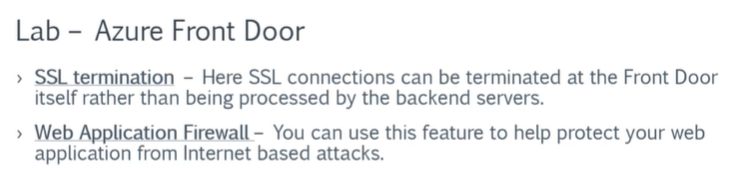


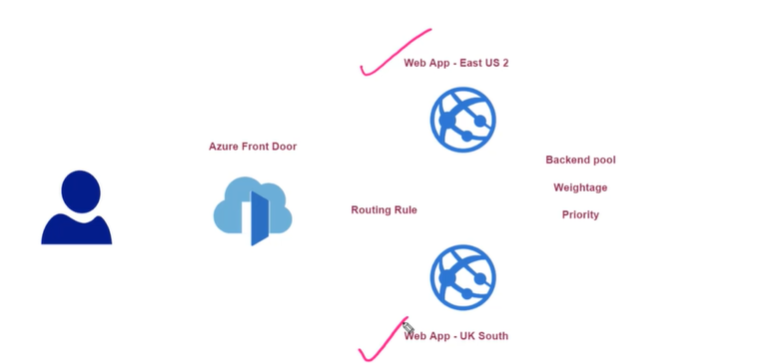


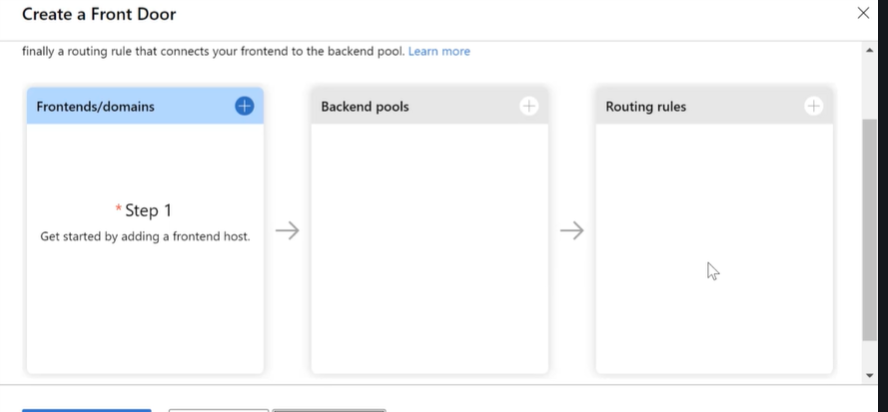
* **Azure Front Door Service**





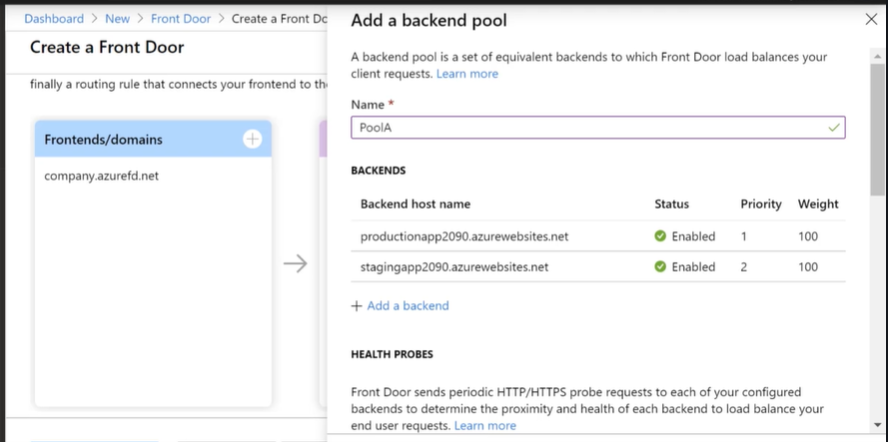






**Fault tolerant Mode:**

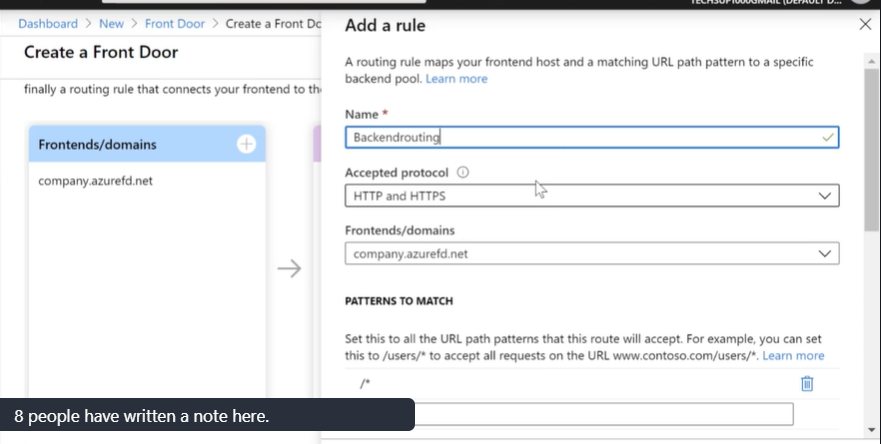
**Priority: 1 and 2, Weight: 100**

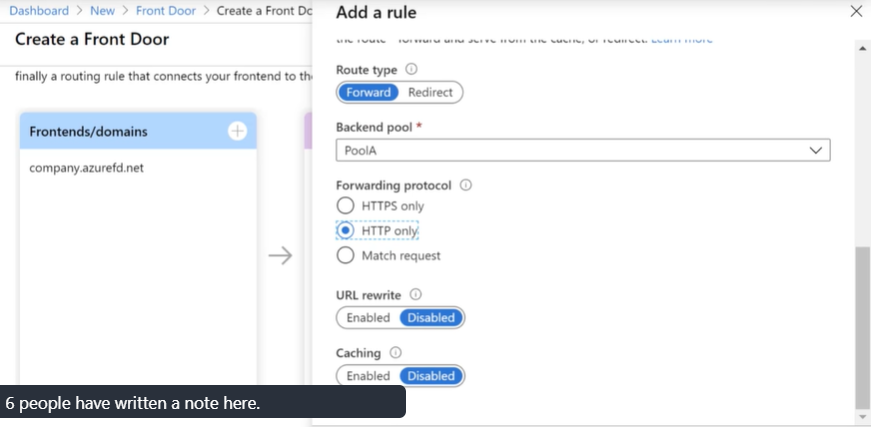


**To determine if backend pool is healthy , health probes can be used.**

**Load Balanced Mode:**

**Priority : 1 , Weight :50**





**Azure Front Door Caching and Compression**

By default, Azure Front Door has the capability to deliver large files. If a file is large, it splits the file into chunks of 8 MB. If the file is not in the cache, the large file is retrieved from the origin. Once the first 8 MB chunk is received , it is sent to the user and then in parallel , the next 8 MB chunk is retrieved from the origin.

Azure Front Door also supports compression. It supports all of the following MIME types. But the file needs to be in size between 1 KB and 8 MB. It supports both gzip and Brotli compression.

* "application/eot"
* "application/font"
* "application/font-sfnt"
* "application/javascript"
* "application/json"
* "application/opentype"
* "application/otf"
* "application/pkcs7-mime"
* "application/truetype"
* "application/ttf",
* "application/vnd.ms-fontobject"
* "application/xhtml+xml"
* "application/xml"
* "application/xml+rss"
* "application/x-font-opentype"
* "application/x-font-truetype"
* "application/x-font-ttf"
* "application/x-httpd-cgi"
* "application/x-mpegurl"
* "application/x-opentype"
* "application/x-otf"
* "application/x-perl"
* "application/x-ttf"
* "application/x-javascript"
* "font/eot"
* "font/ttf"
* "font/otf"
* "font/opentype"
* "image/svg+xml"
* "text/css"
* "text/csv"
* "text/html"
* "text/javascript"
* "text/js", "text/plain"
* "text/richtext"
* "text/tab-separated-values"
* "text/xml"
* "text/x-script"
* "text/x-component"
* "text/x-java-source"
* **Transient Faults**

