**FINAL PROJECT**

Create a **Web API Project** to store Product Information. Use Entity Framework to store the product information in the database. The user should be able to perform all the CRUD Operations. Configure **GET, POST, PUT** and **DELETE**.

The Product Entity should have the following properties:

* ProductID
* ProductName
* Price
* Brand
* ManufactureDate
* ExpirationDate

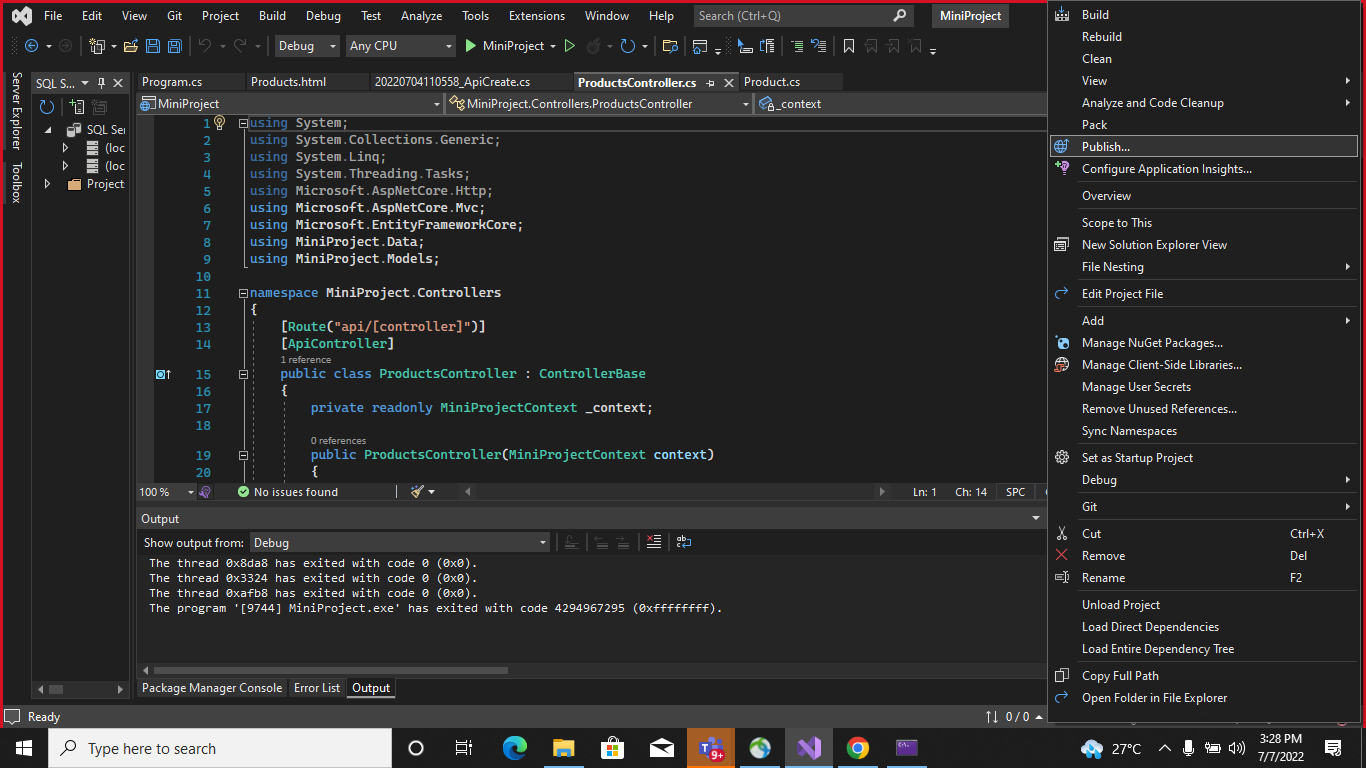
Use Data Annotations to

* Mark the Primary Key
* Make ProductName Mandatory
* Make Price a Number

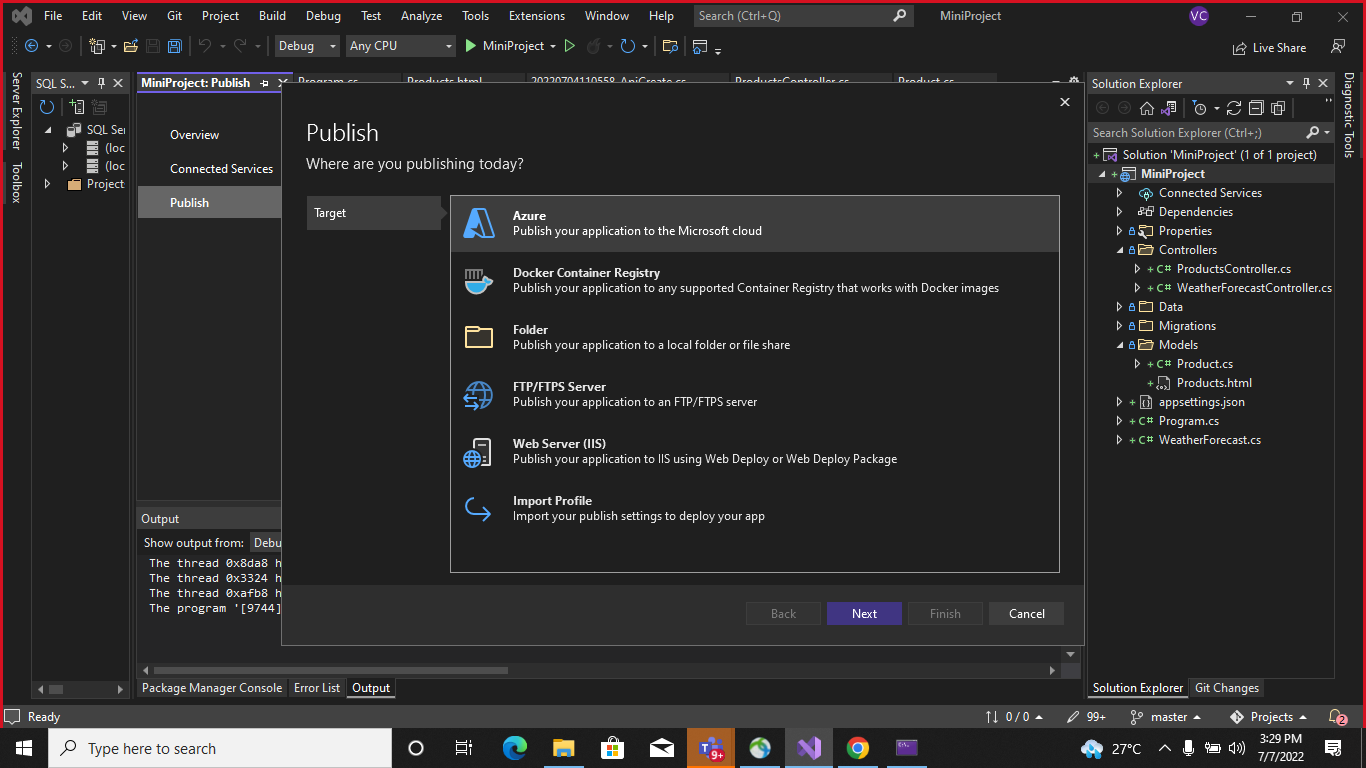
Create a JQuery and AJAX Client to consume the Web API and show the result.

Azure Hosting:

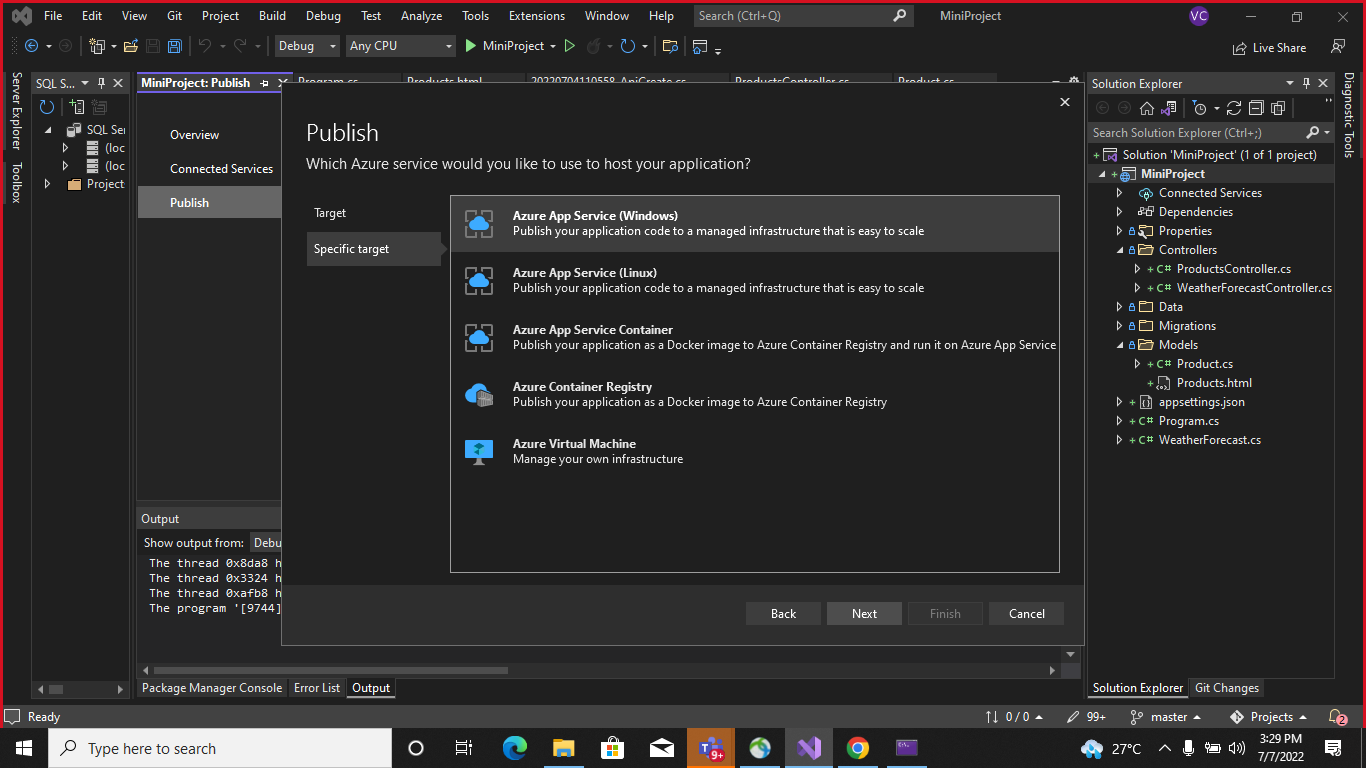
* Host the web api in azure and consume the same using JQuery Client.
* Configure Scale out by adding rules for custom scaling
* Configure Deployment slots for staging and production
* Configure Application Insights for the project
* Configure Swagger for the api
* Work with Log Analytics with the sample logs available
* Publishing API on Azure
* In Solution explorer, right click on project and select publish.



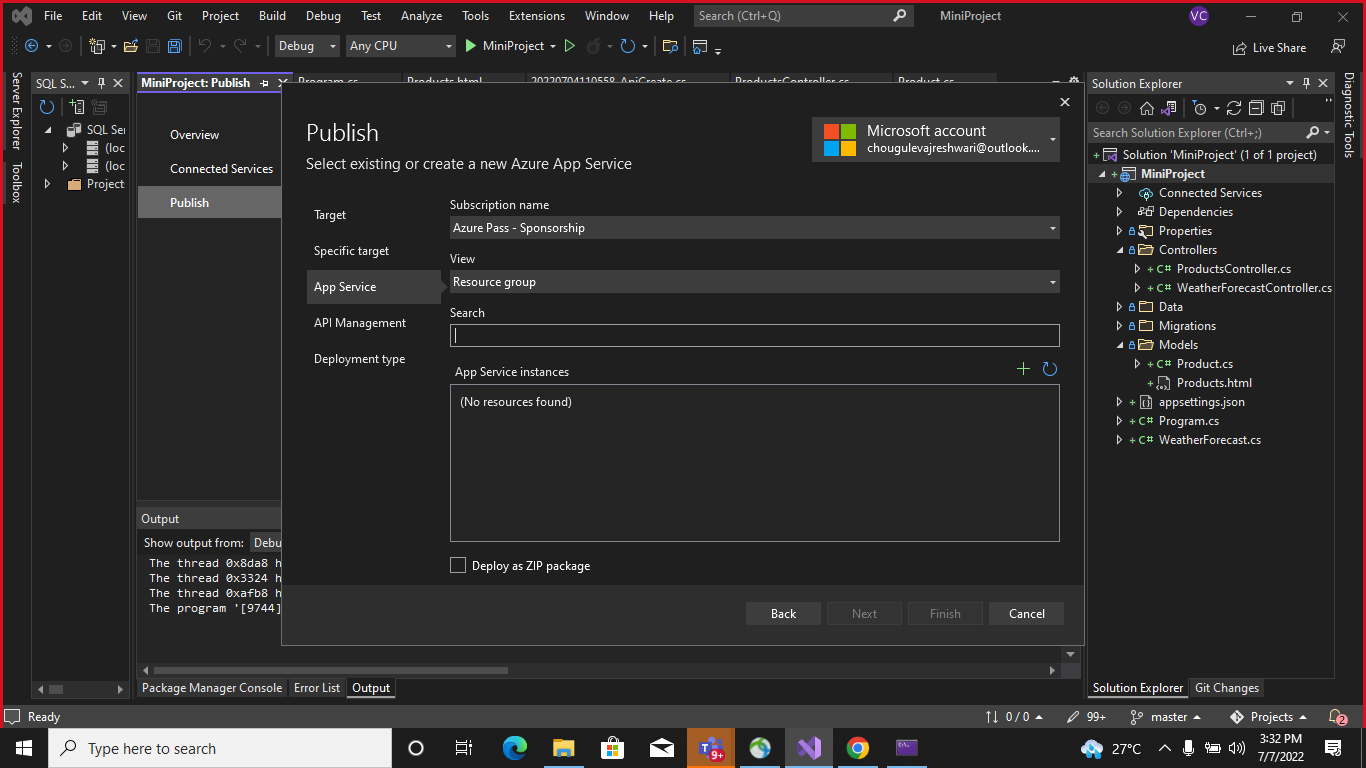
* In the publish dialog, select Azure and select the next button.



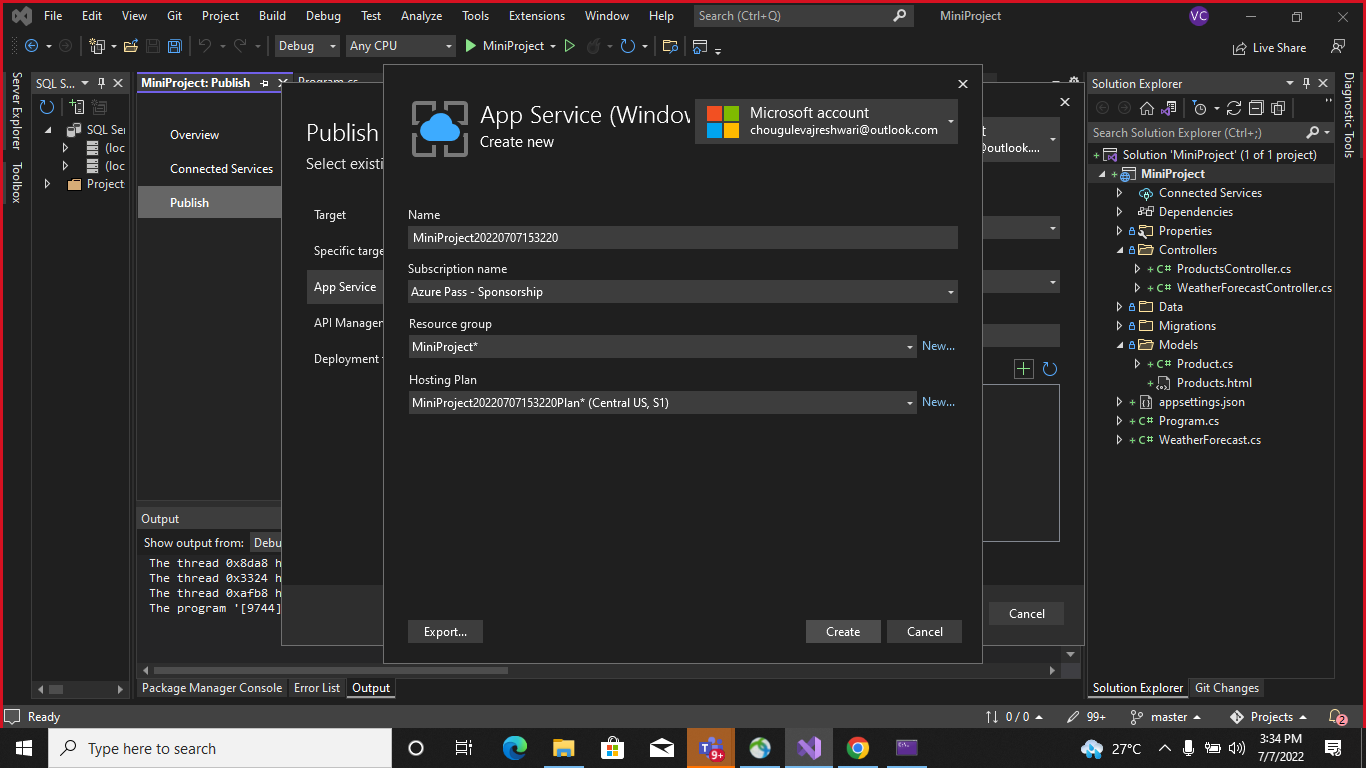
* Select Azure App Service (Windows) and select the next button.



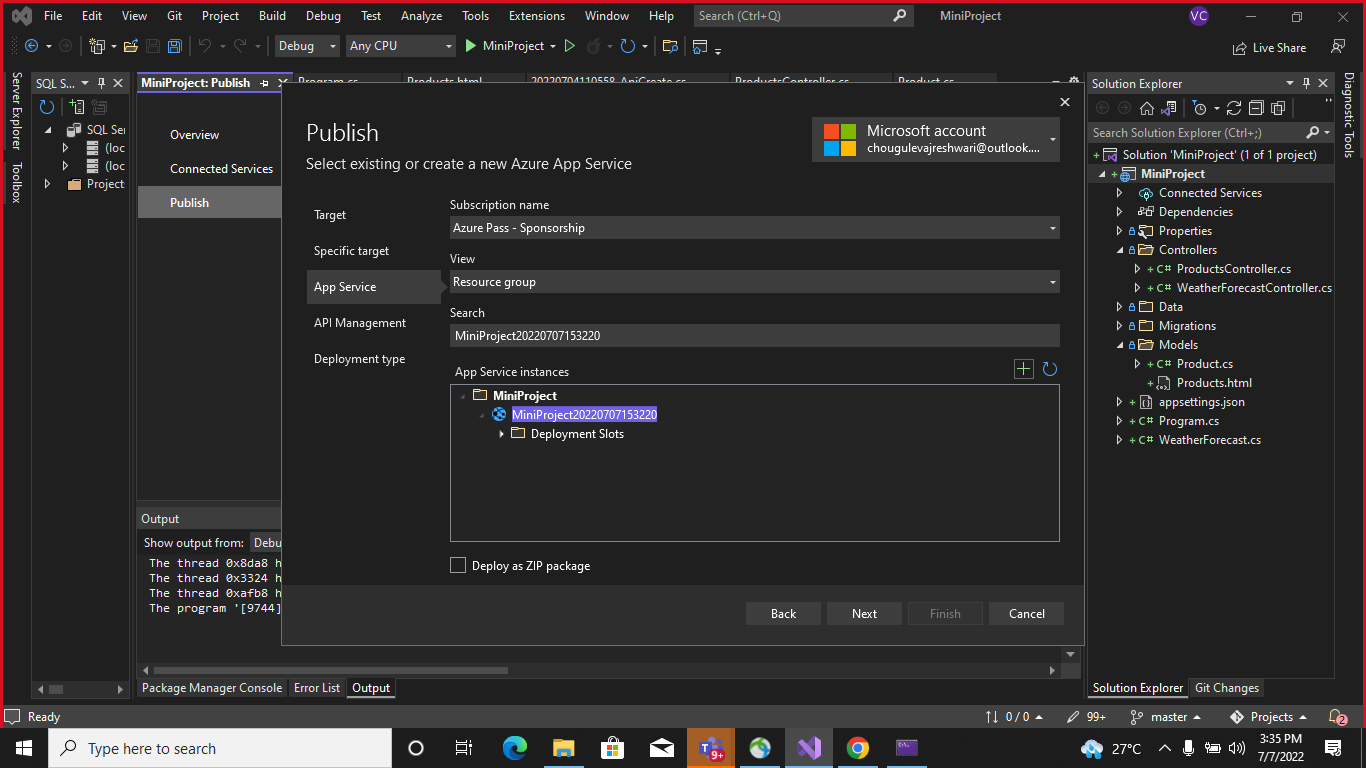
* Select Create a new Azure App Service



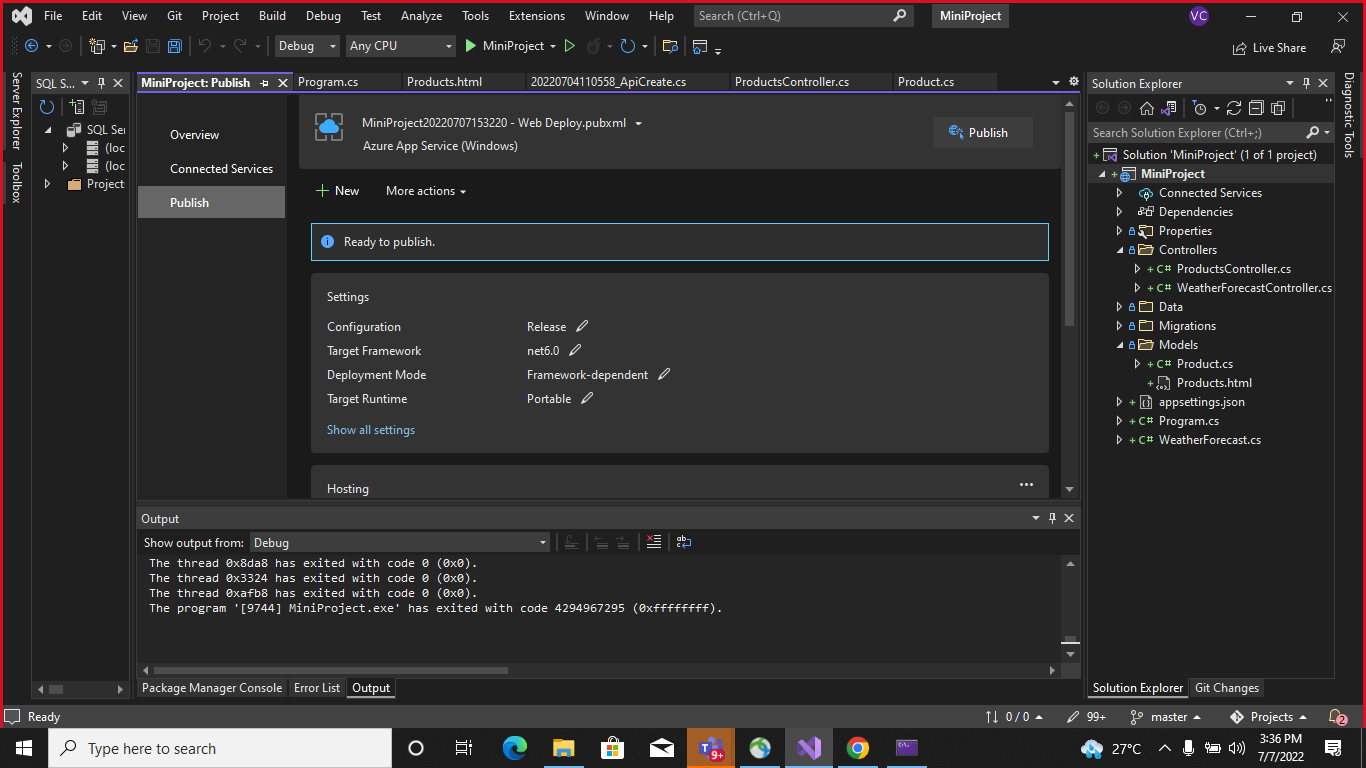
* Click on Create (You can keep as it is or change fields)



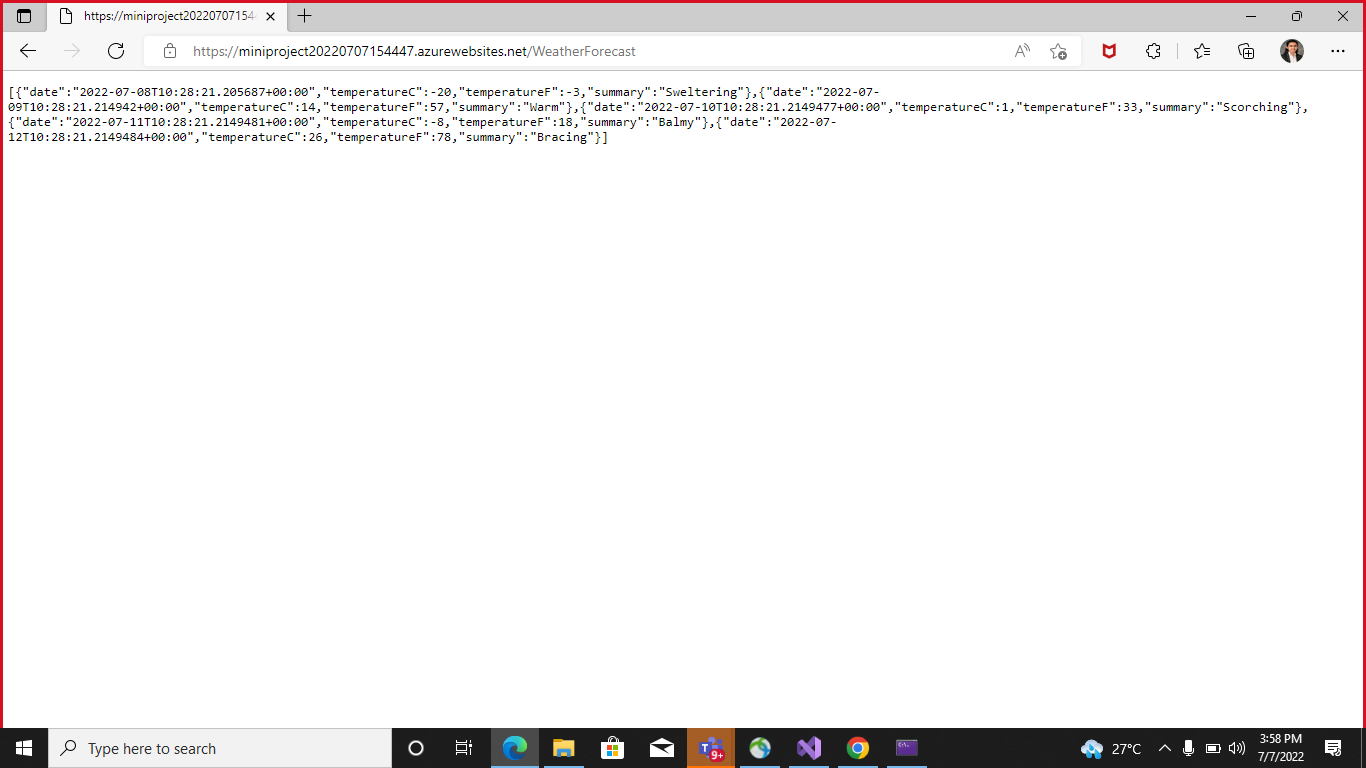
* Click on Finish



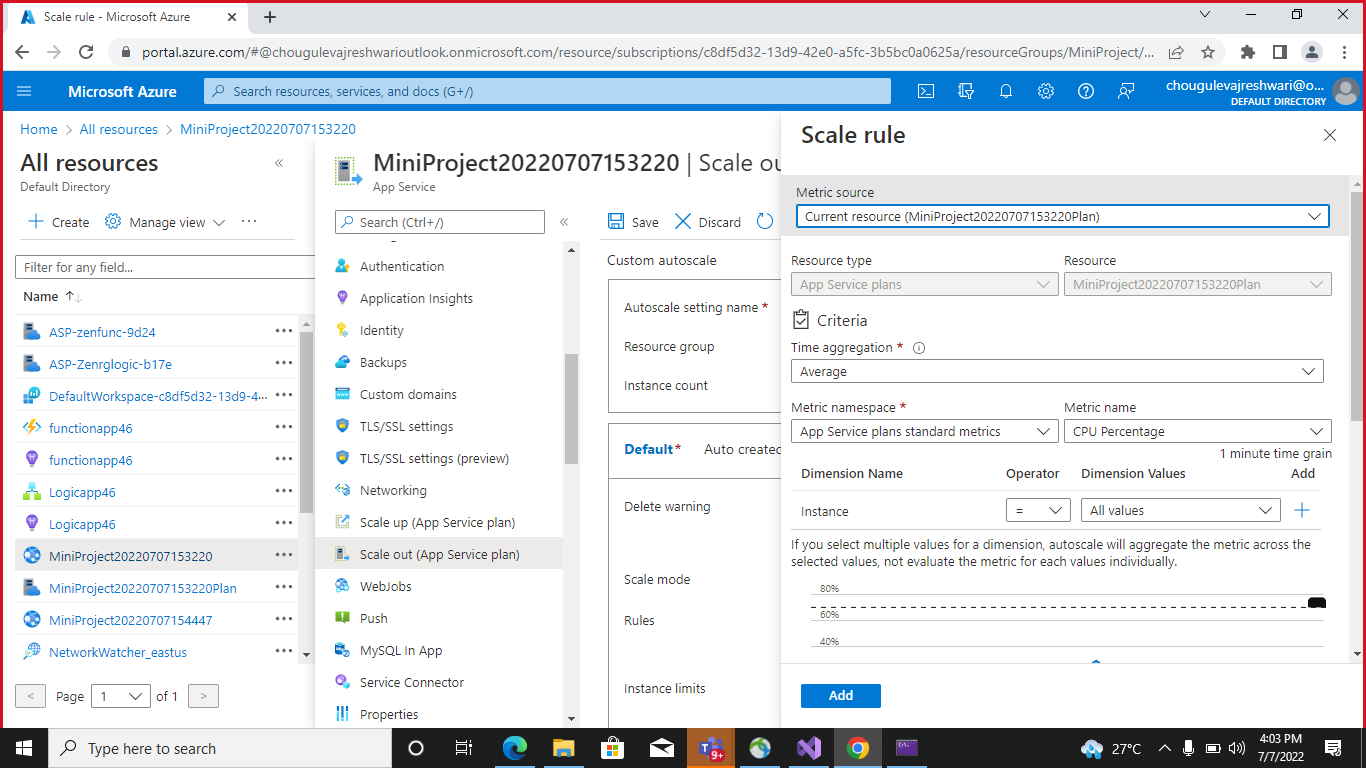
* After creating, the dialog automatically closed and the publish dialog gets focus again. The instance that was created is automatically selected.

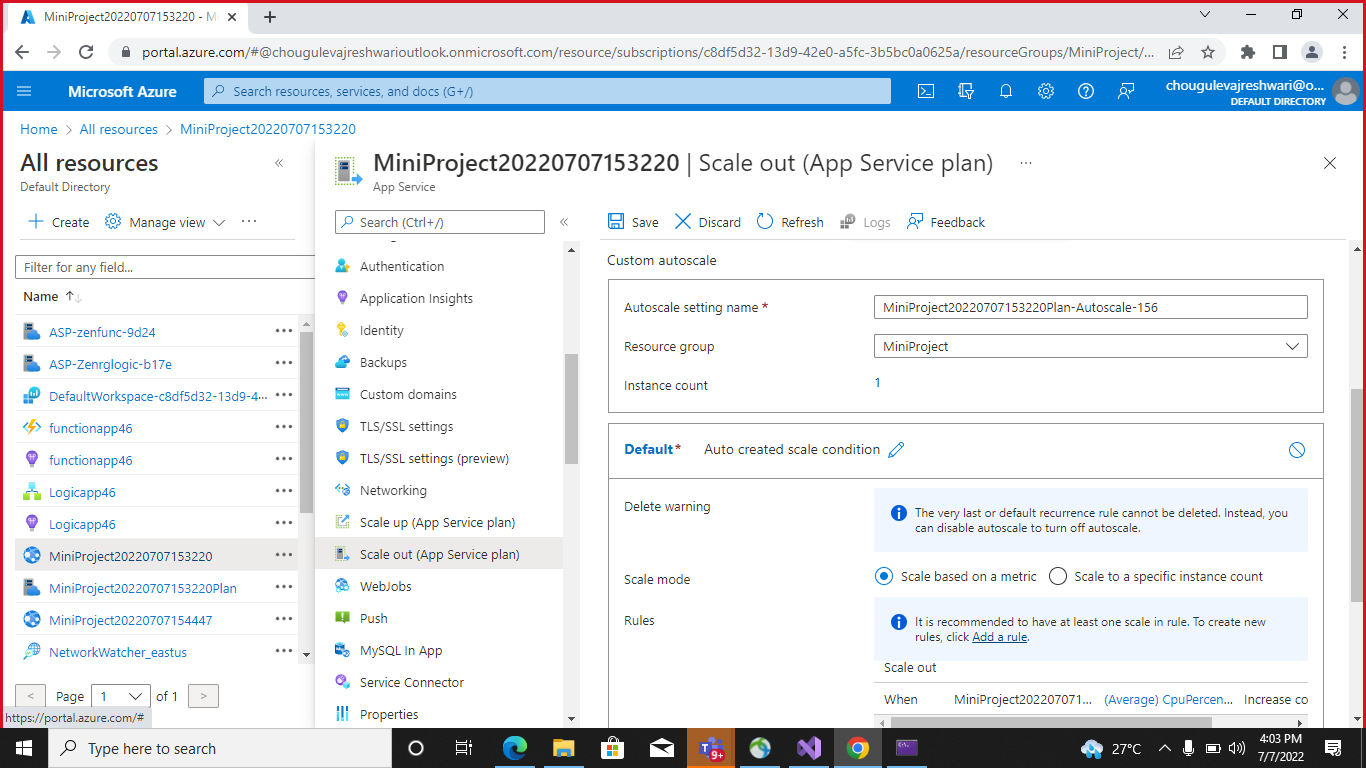


* Next if you click on Publish button you’ll get JSON result

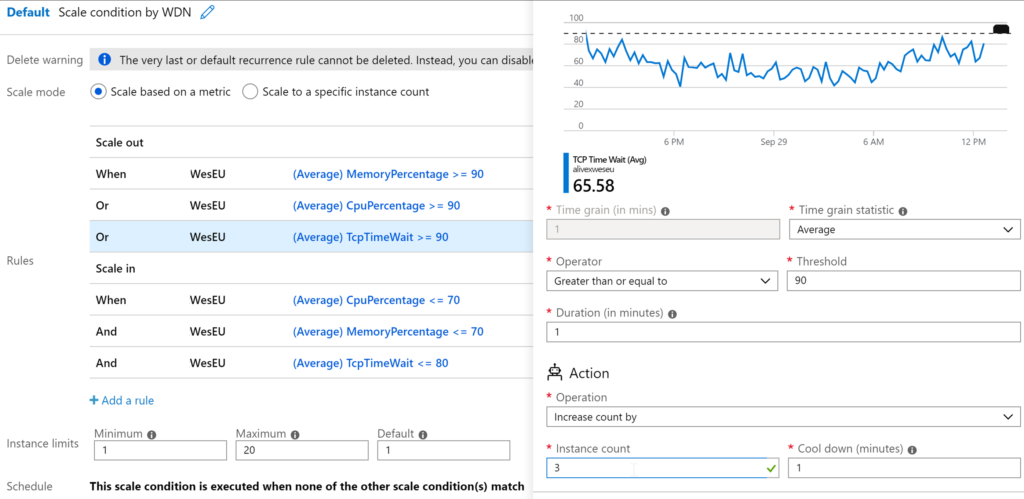


* Configure Scale out by adding rules for custom scaling
* Open Azure portal and in that open App Service, go to settings and select scale out (App Service Plan), Select Custom autoscale.
* Then click add a rule. This open as a context pane on the right side.





By default, this sets the option to scale your instance count by 1 if the CPU percentage of the resource exceeds 70 percent. Leave it at its default value and click add.

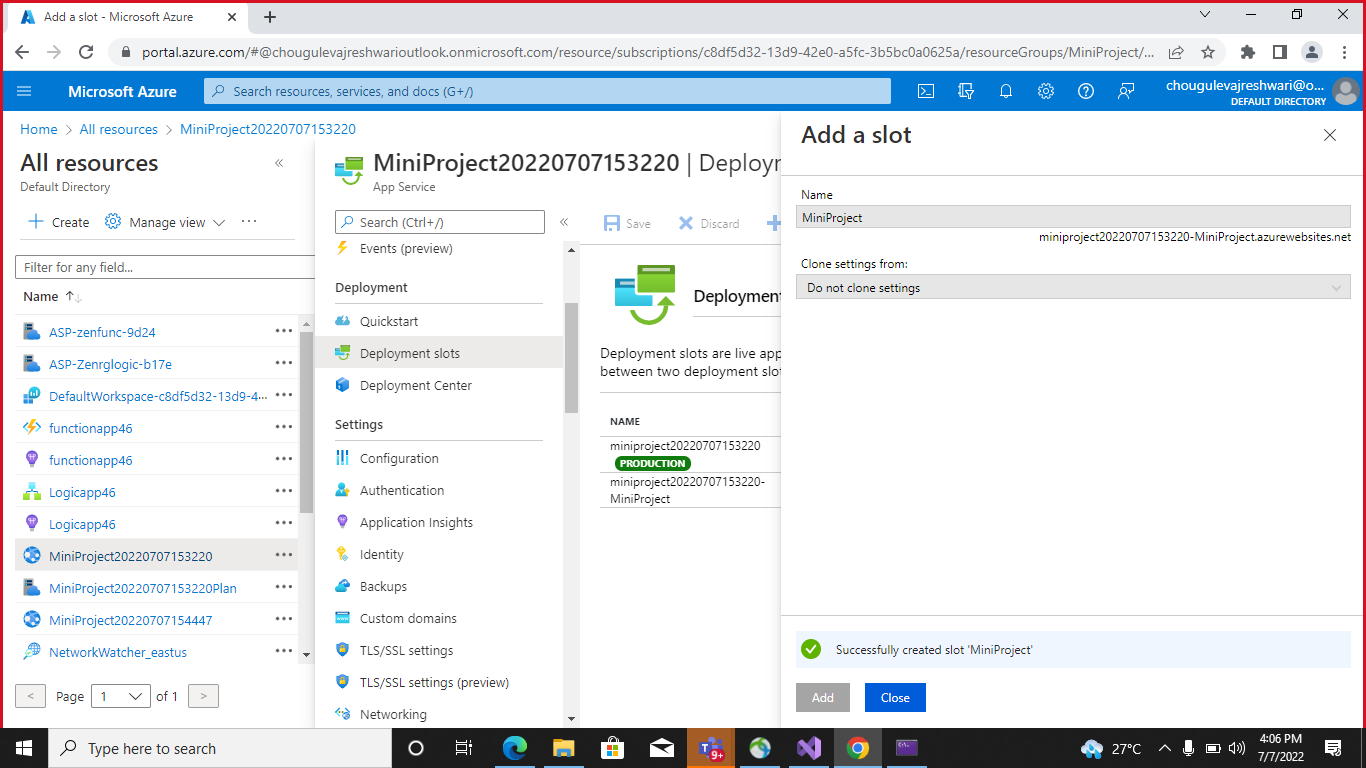


* Scale Out-

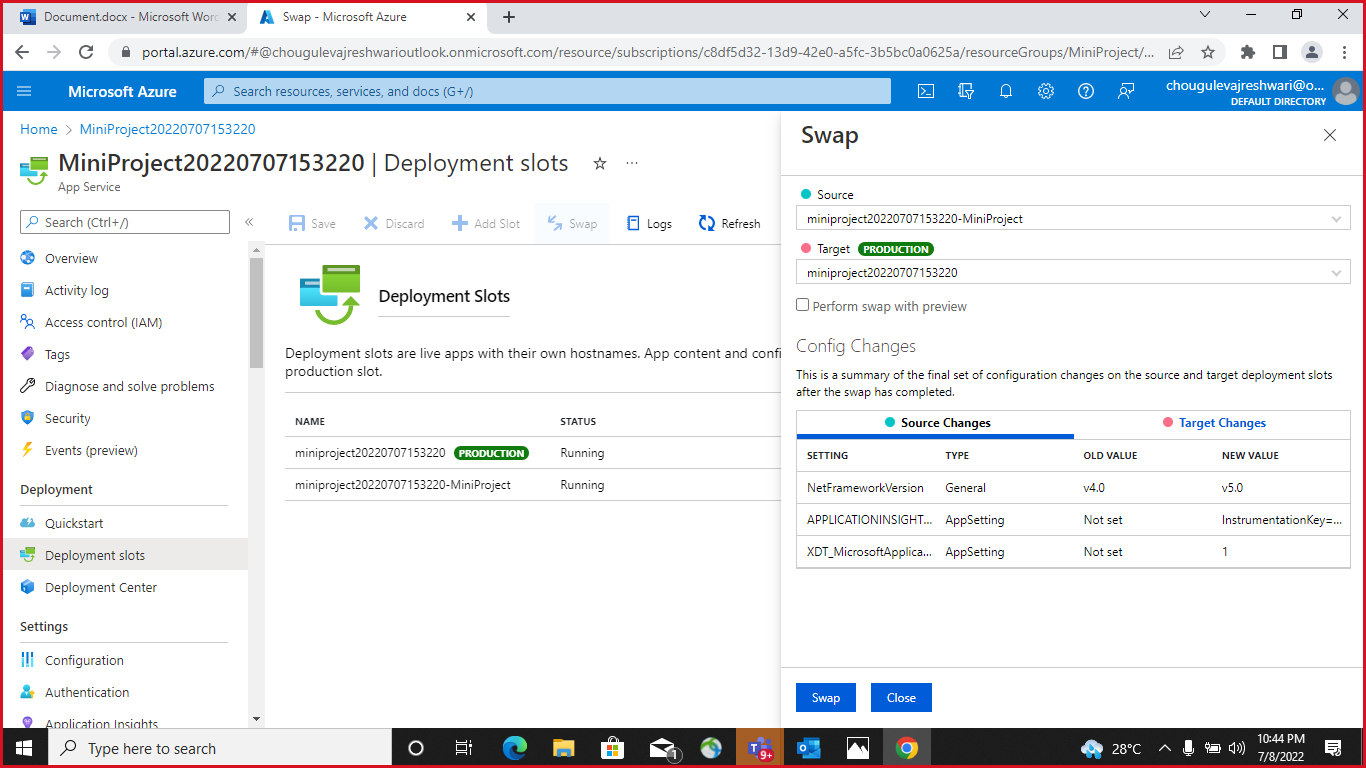
It is basically adding multiple instances of the application that runs in your app. In other words, it increases the number of VM instances up to 30 depending upon your pricing tier. However, in an Isolated tier, we can further scale up to 100 instances based on our requirements. Additionally, we can do a scale-out count manually or set it to auto-scaling based on some rules.

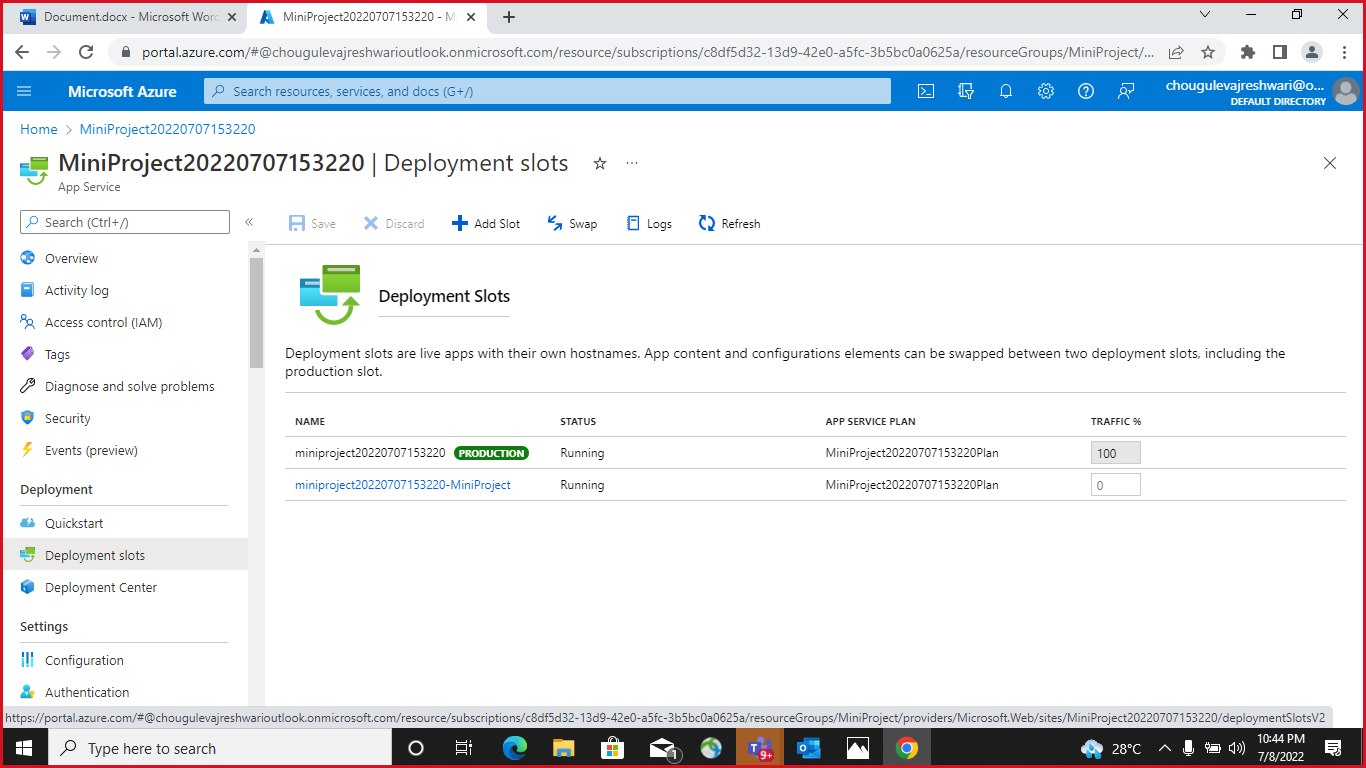
Scaling out is a special option available to Azure App Service. This feature allows you to create one or multiple clones of your existing application to better balance traffic or resource load. Essentially, it’s like creating multiple App Services that all live behind a Load Balancer. The Scale Out feature is available in two methods, Manual Scale and Custom Autoscale. Both can help you accomplish your goal of allowing your application to be more performant for your users.

* Configure Deployment slots for staging and production
* Select Deployment slots, and the select (+) Add slot



* Verify the configuration settings for swap and select swap.

r

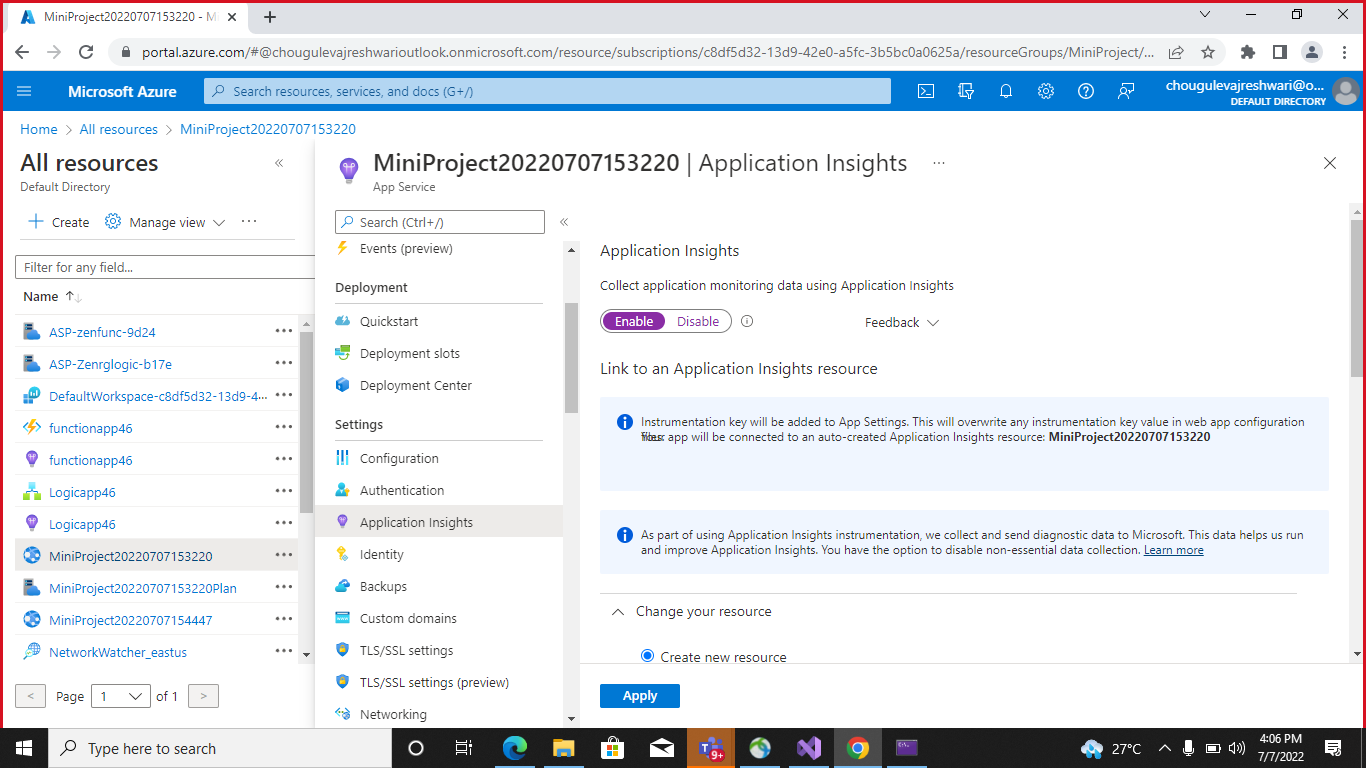


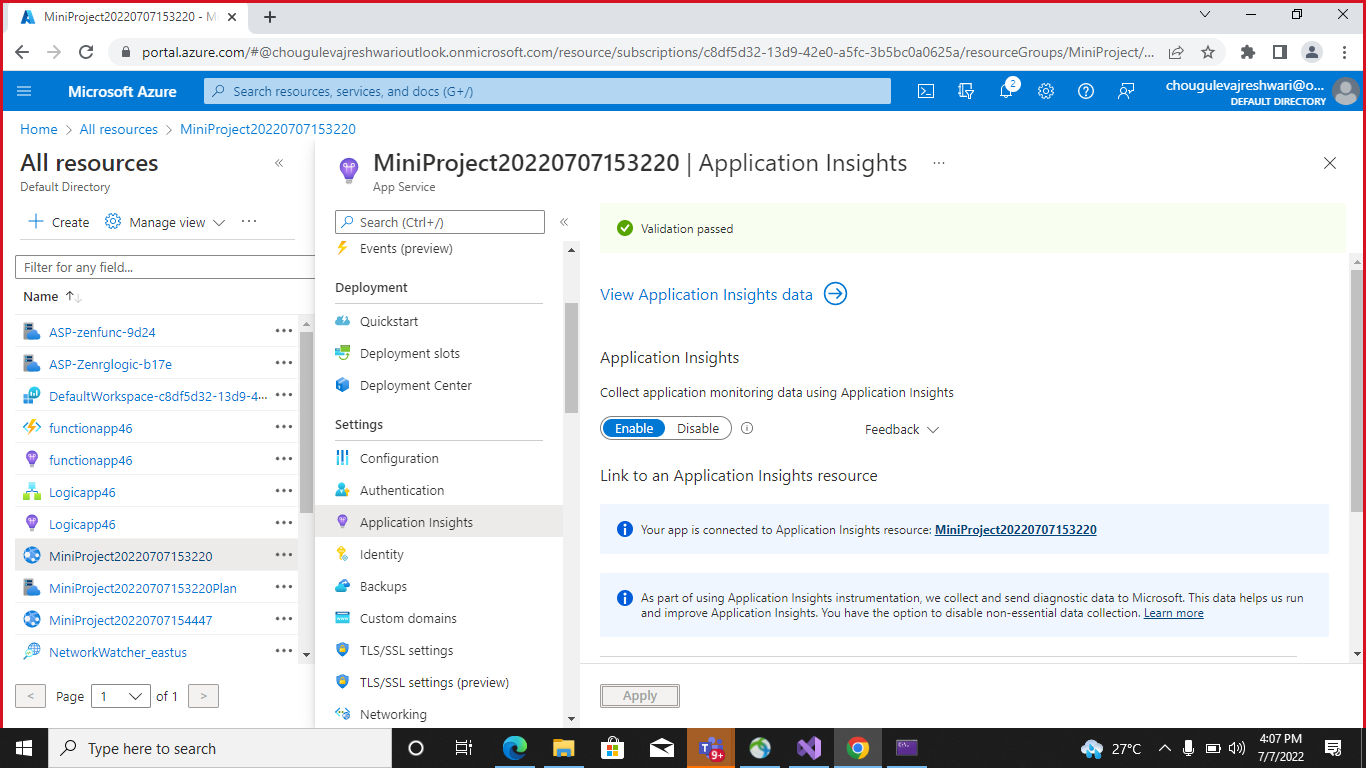
* Deployment Slots -

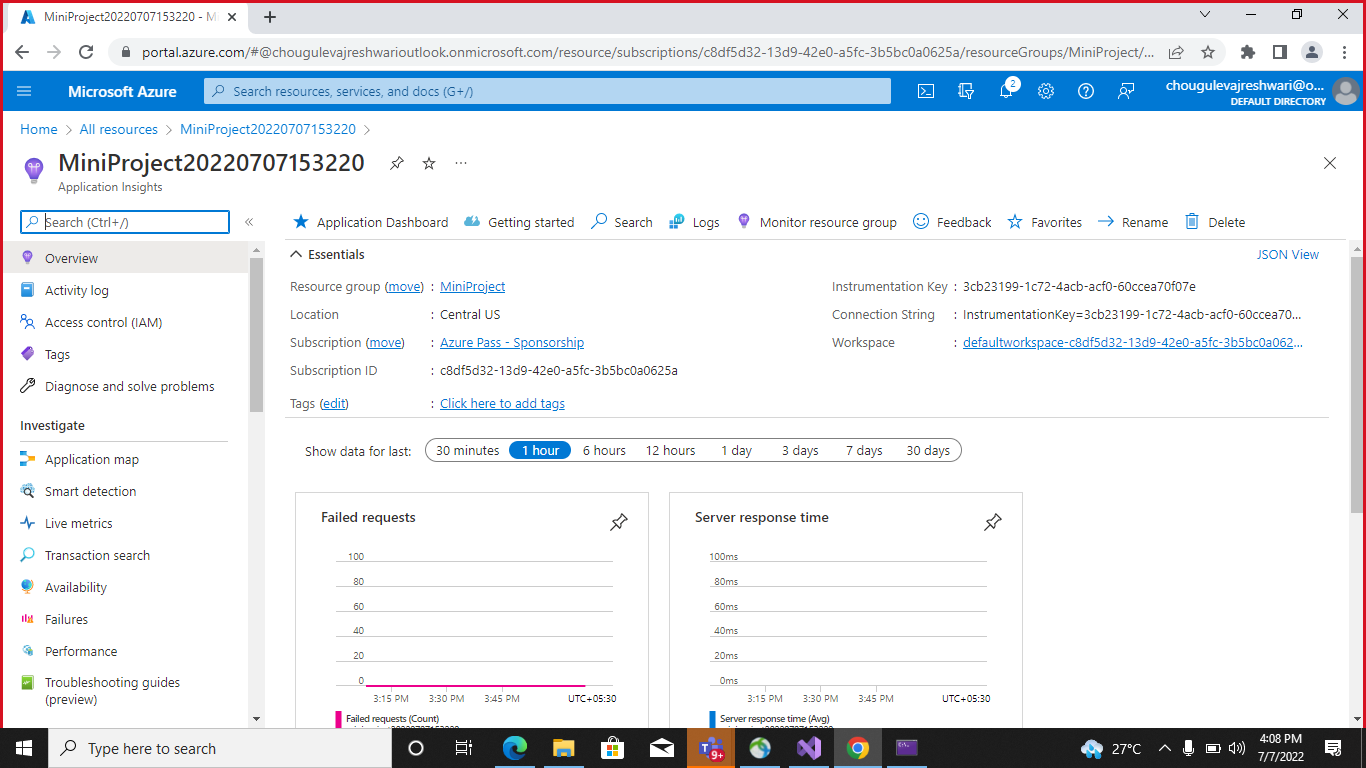
Azure Functions deployment slots allow your function app to run different instances called "slots". Slots are different environments exposed via a publicly available endpoint. One app instance is always mapped to the production slot, and you can swap instances assigned to a slot on demand. Function apps running under the Apps Service plan may have multiple slots, while under the Consumption plan only one slot is allowed.

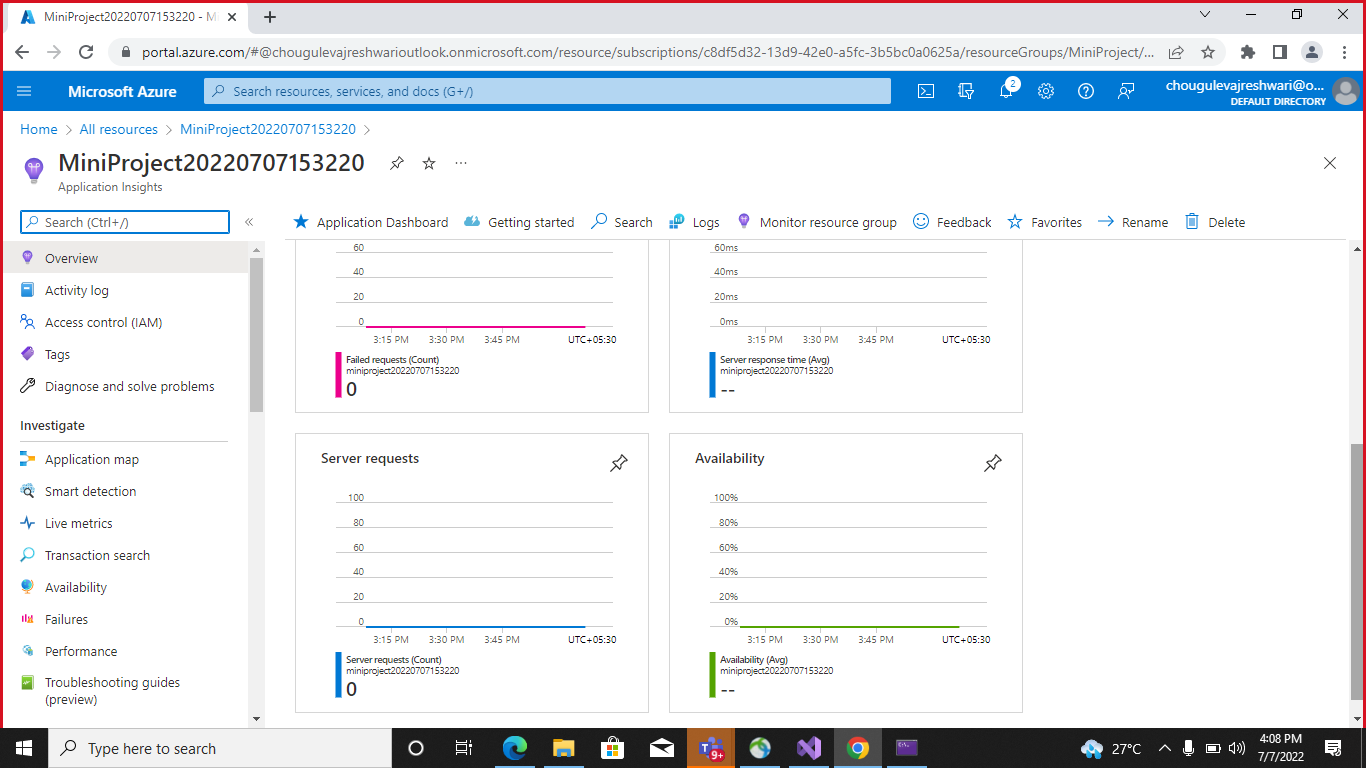
Deployment slots are live apps with their own host names. App content and configurations elements can be swapped between two deployment slots, including the production slot.

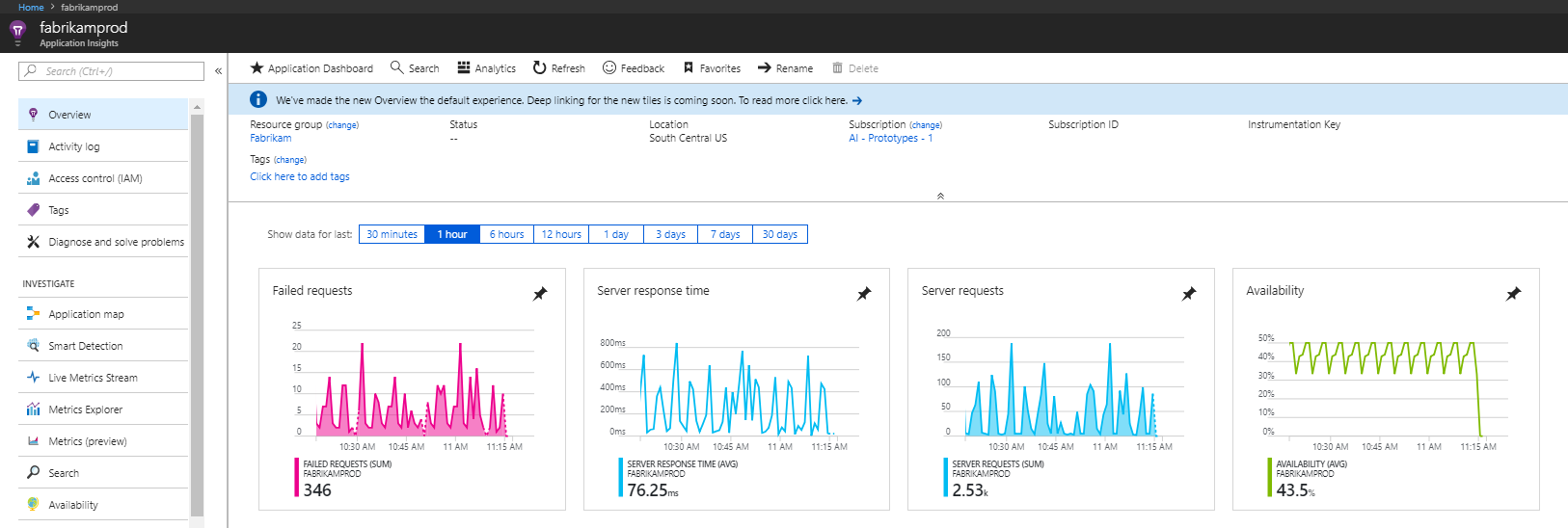
* Configure Application Insights for the project











* Application Insights -

Application Insights is a feature of [Azure Monitor](https://docs.microsoft.com/en-us/azure/azure-monitor/overview) that provides extensible application performance management (APM) and monitoring for live web apps. Developers and DevOps professionals can use Application Insights to:

* Automatically detect performance anomalies.
* Help diagnose issues by using powerful analytics tools.
* See what users actually do with apps.
* Help continuously improve app performance and usability.

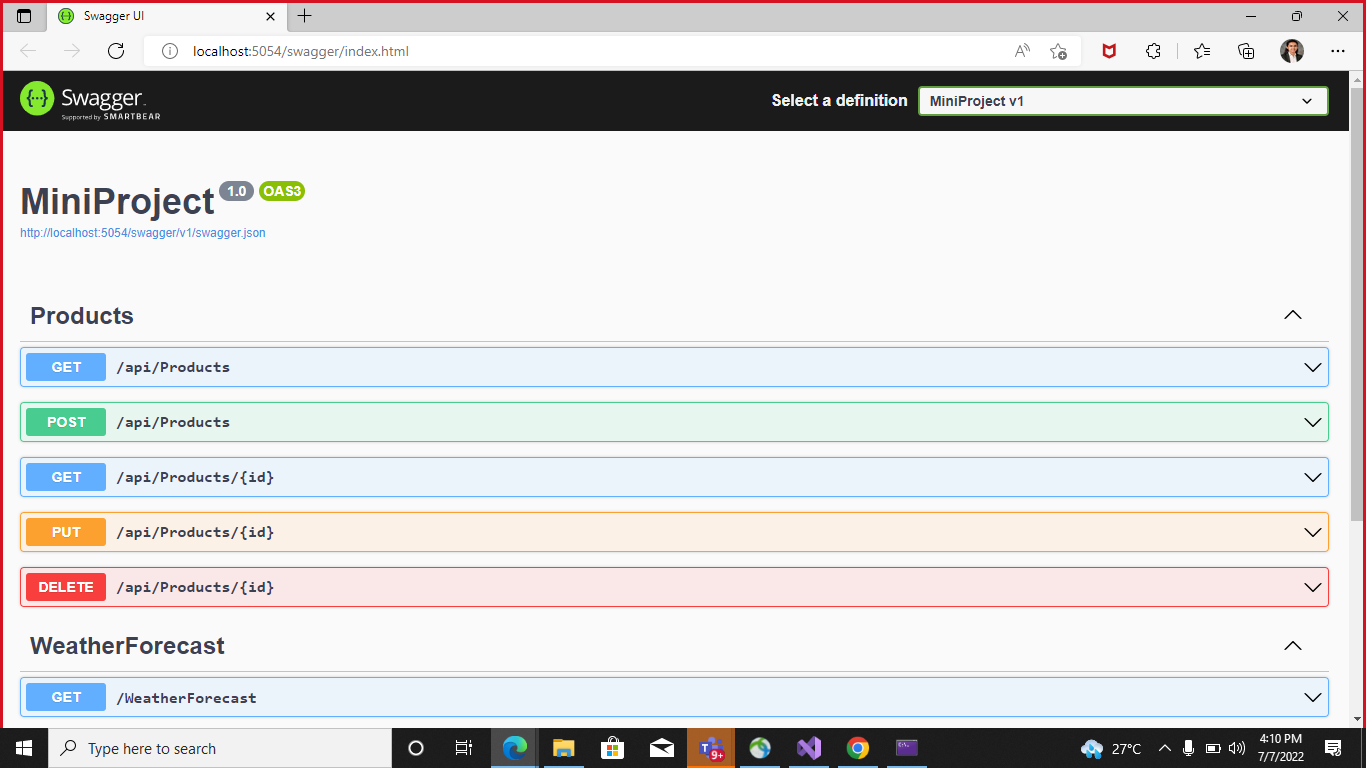
Application Insights:

* Supports a wide variety of platforms, including .NET, Node.js, Java, and Python.
* Works for apps hosted on-premises, hybrid, or on any public cloud.
* Integrates with DevOps processes.
* Has connection points to many development tools.
* Can monitor and analyze telemetry from mobile apps by integrating with Visual Studio [App Center](https://appcenter.ms/)
* Configure Swagger for the Api
* Swagger -

Swagger is an open source set of rules, specifications and tools for developing and describing RESTful APIs. The Swagger framework allows developers to create interactive, machine and human-readable API documentation.

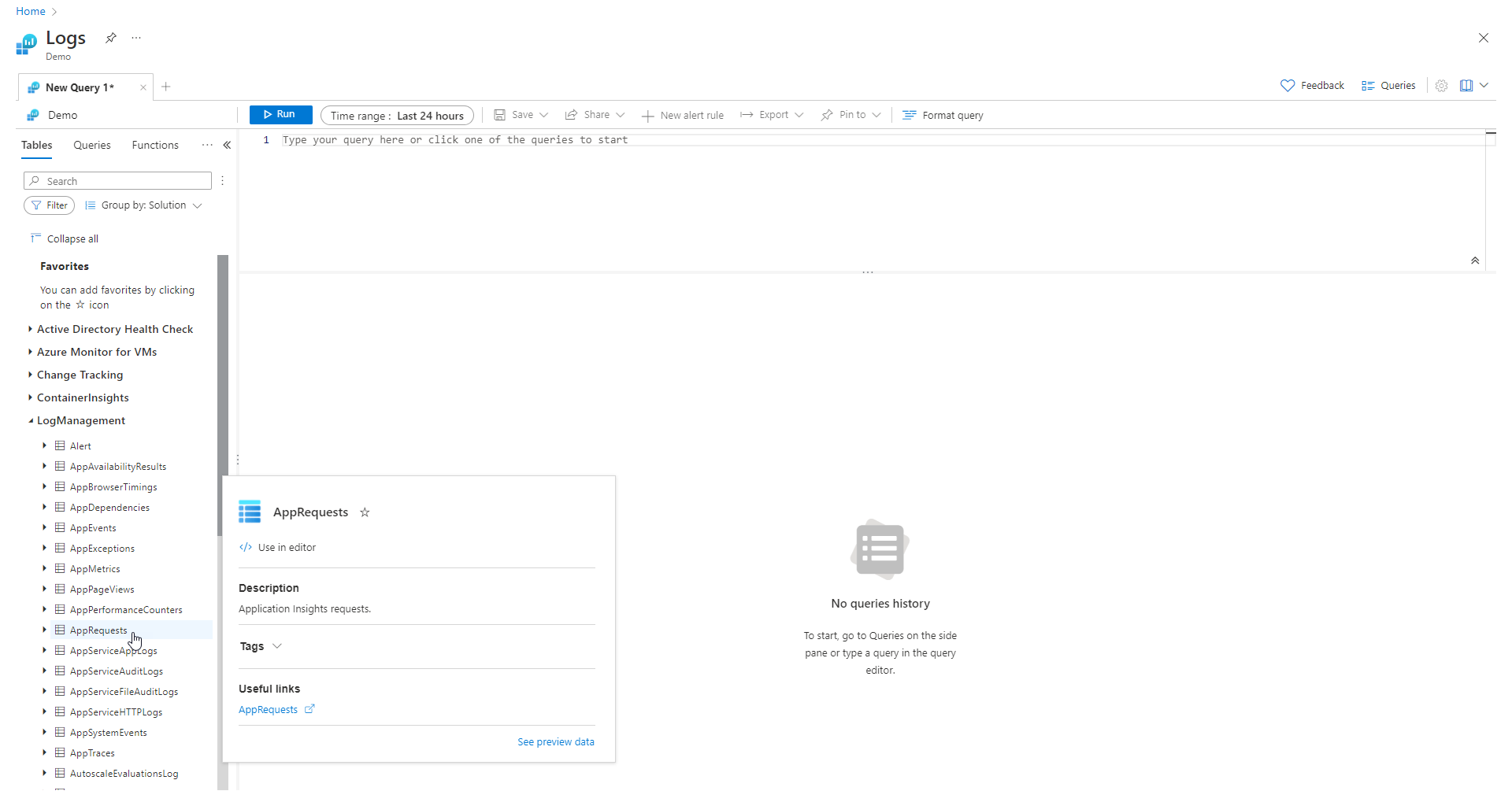
Swagger is the standard way of documenting the Standard APIs. Swagger is helpful when deploying APIs in azure. Swagger is primarily used for documenting API

The building APIs that are internal in the enterprise or for the public consumption, the theme is the same that the developers usually use in the apps that they are building. For the other developers to be able to use our API, the API must be properly documented



* Log Analytics

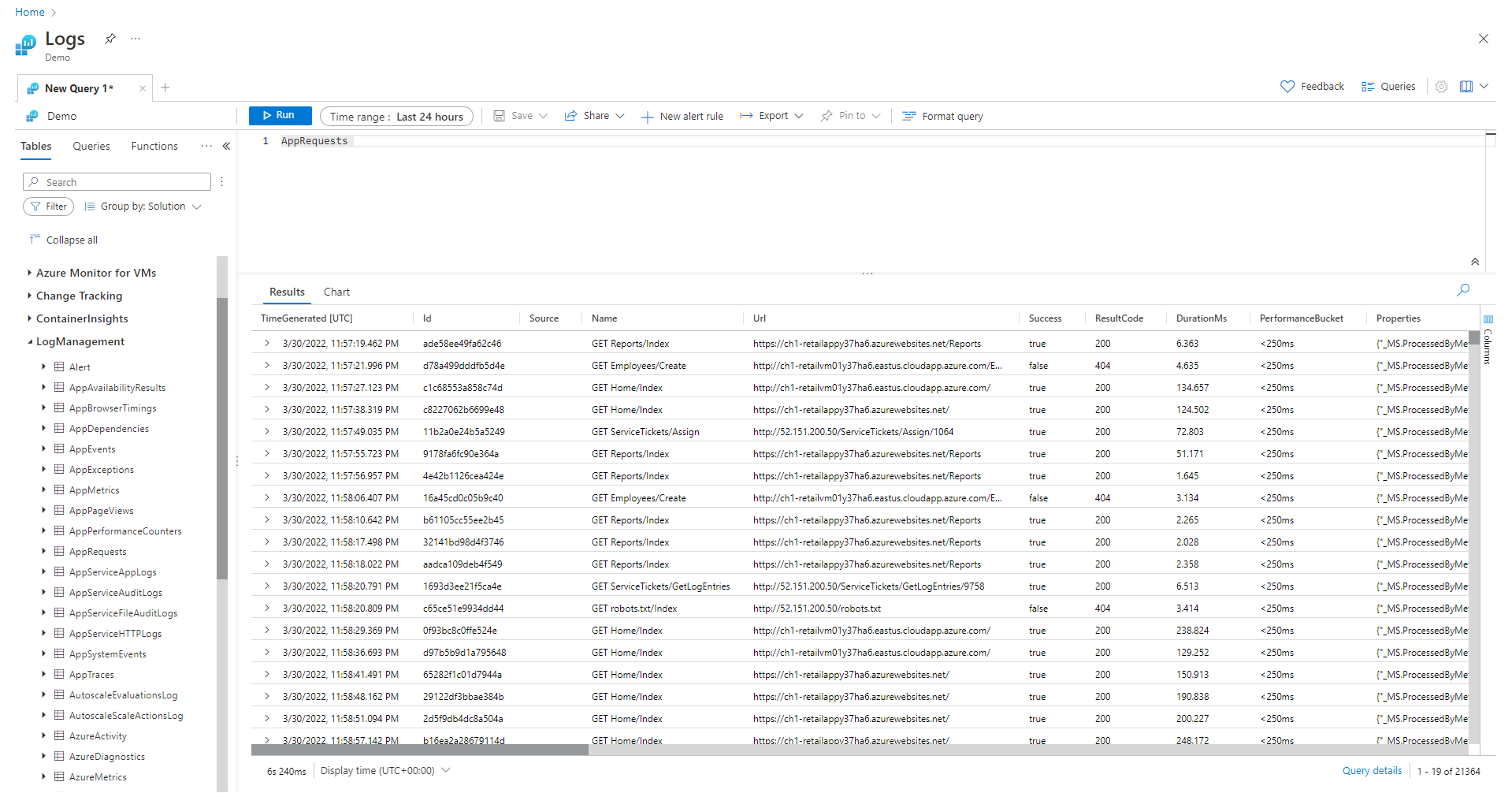
Log Analytics is a tool in the Azure portal to edit and run log queries from data collected by Azure Monitor logs and interactively analyze their results. You can use Log Analytics queries to retrieve records that match particular criteria, identify trends, analyze patterns, and provide various insights into your data.



## Write a Query

Let's write a query by using the AppRequests table. Double-click its name to add it to the query window. You can also type directly in the window. You can even get IntelliSense that will help complete the names of tables in the current scope and Kusto Query Language (KQL) commands.

This is the simplest query that we can write. It just returns all the records in a table. Run it by selecting the Run button or by selecting Shift+Enter with the cursor positioned anywhere in the query text.

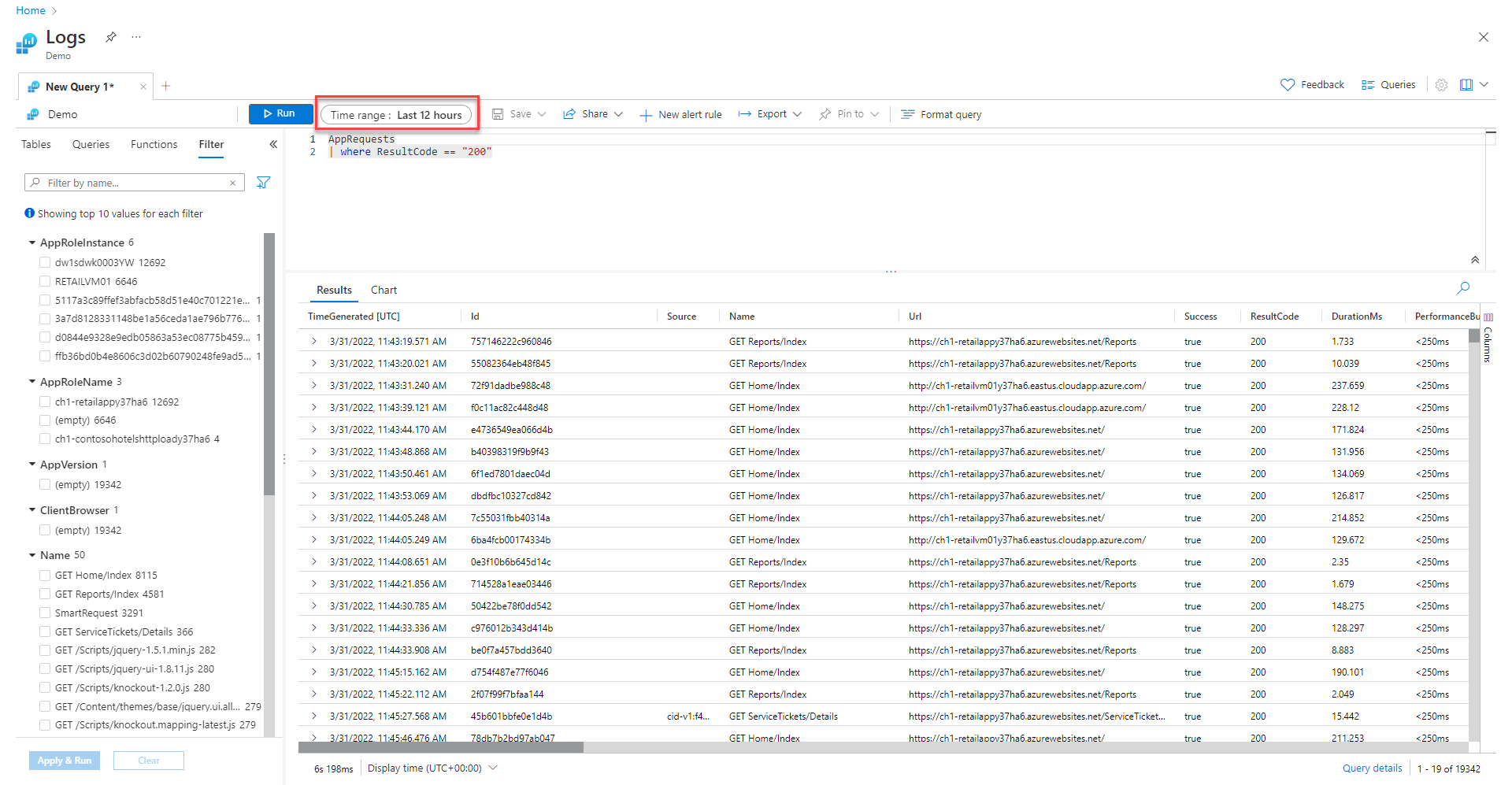


### Time range

All queries return records generated within a set time range. By default, the query returns records generated in the last 24 hours.

You can set a different time range by using the [where operator](https://docs.microsoft.com/en-us/azure/data-explorer/kusto/query/tutorial?pivots=azuremonitor#filter-by-boolean-expression-where-1) in the query. You can also use the Time range dropdown list at the top of the screen.

Let's change the time range of the query by selecting Last 12 hours from the Time range dropdown. Select Run to return the results.



## Analyze results

In addition to helping you write and run queries, Log Analytics provides features for working with the results. Start by expanding a record to view the values for all of its columns.

