

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, DELETE**.

The Product Entity should have the following properties:

- Product ID
- Product Name
- Price
- Brand
- Manufacture Date
- Expiration Date

Use Data Annotations to

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

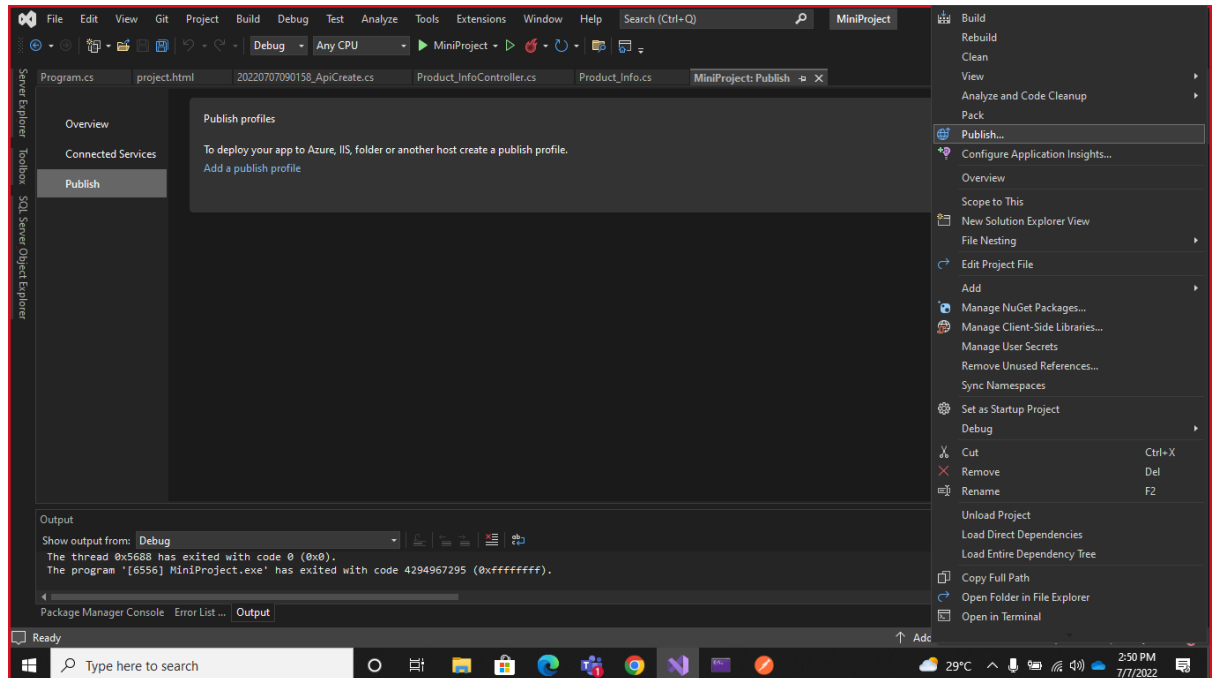
Create a JQuery and AJAX Client to consume the Web API and show the result.

Azure Hosting:

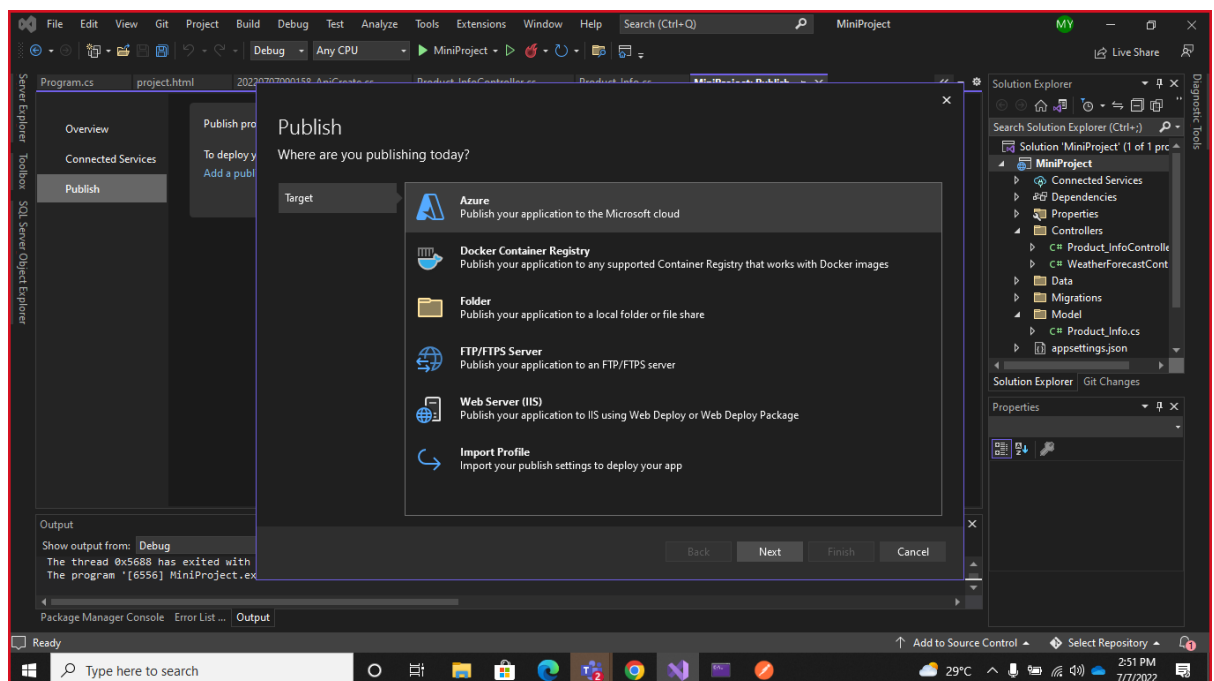
- Host the web API in azure and consume the same using JQuery Client.
- Configure Scale out by adding rules for custom scaling.
- Configure Deployment slots for staging and production.
- Configure Application Insights for the project.
- Configure Swagger for the API.
- Work with Log Analytics with the sample logs available.

1. Host the web API in azure and consume the same using JQuery Client.

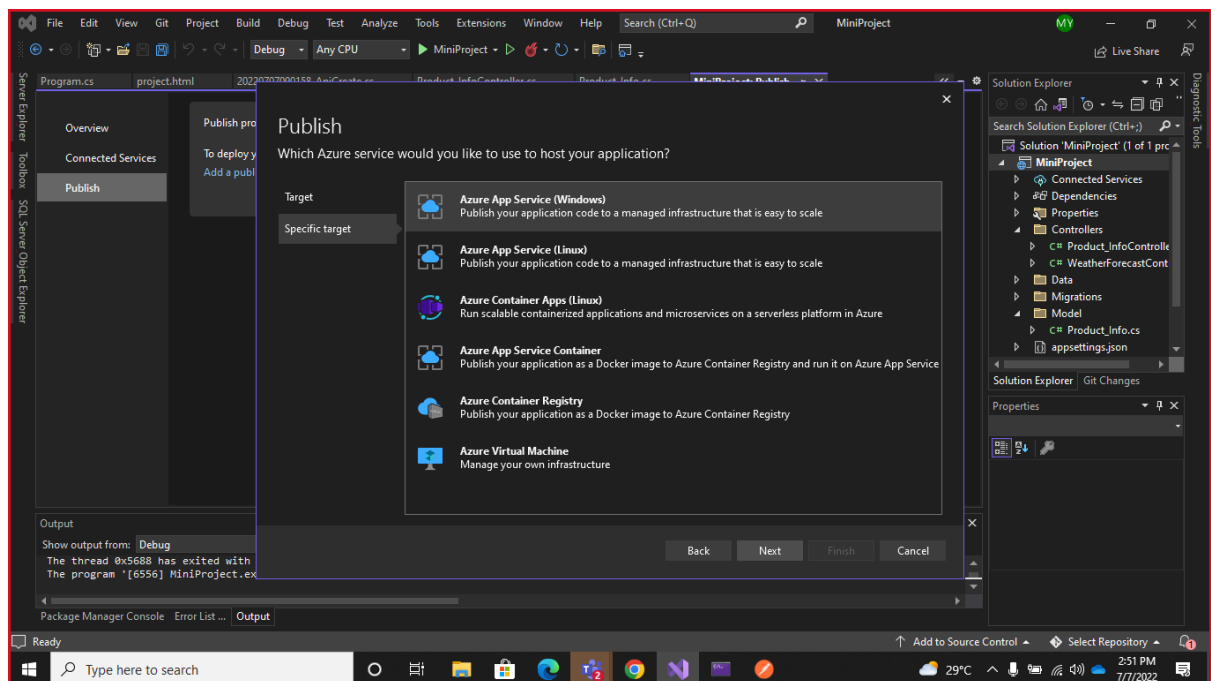
Publishing the Web API from Visual Studio to Azure Portal.



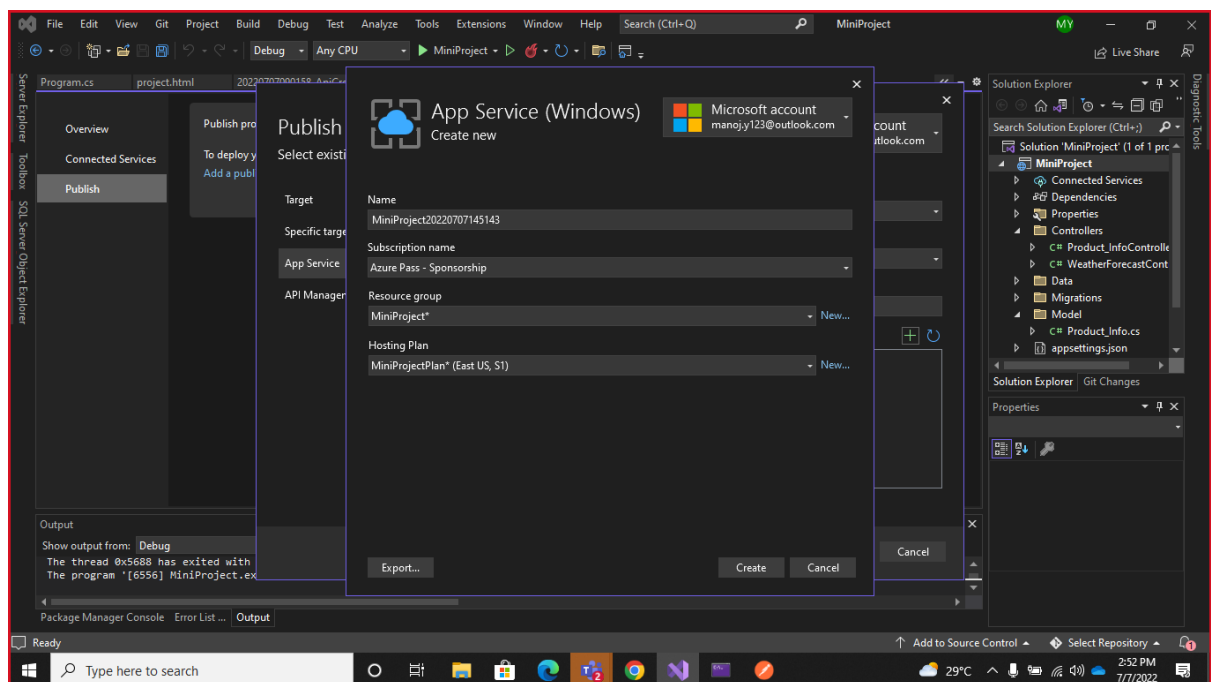
Configuring the Azure service



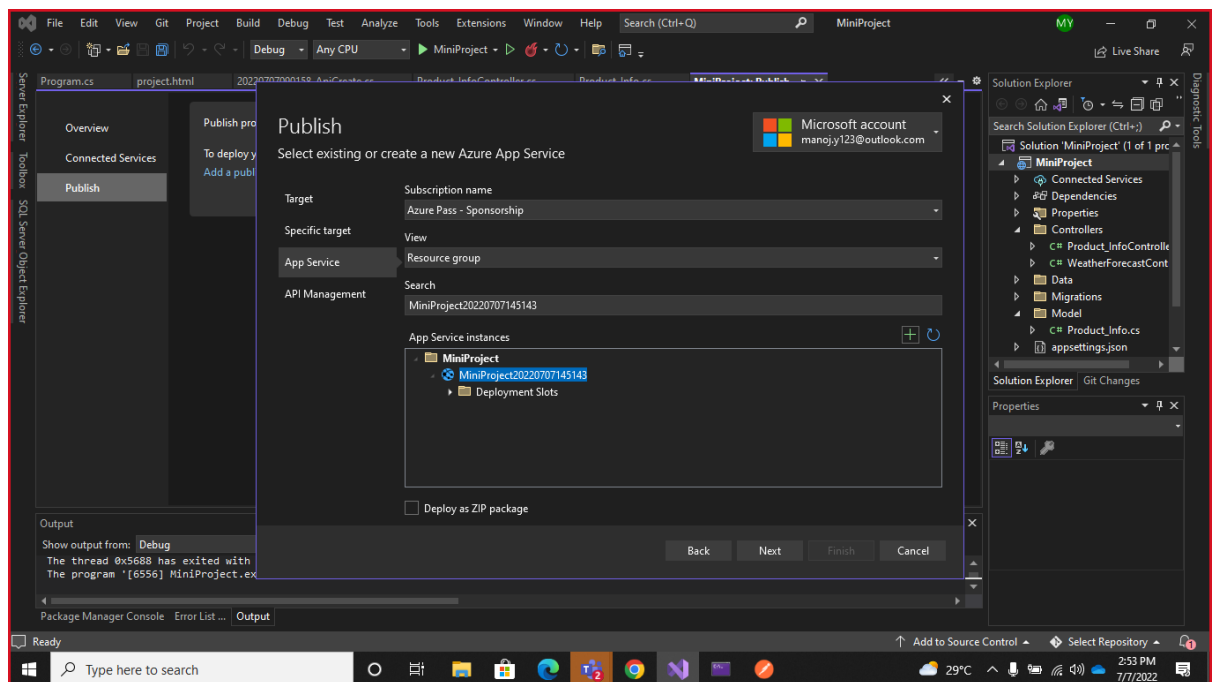
Publishing the Web API to Azure App Service (Windows).



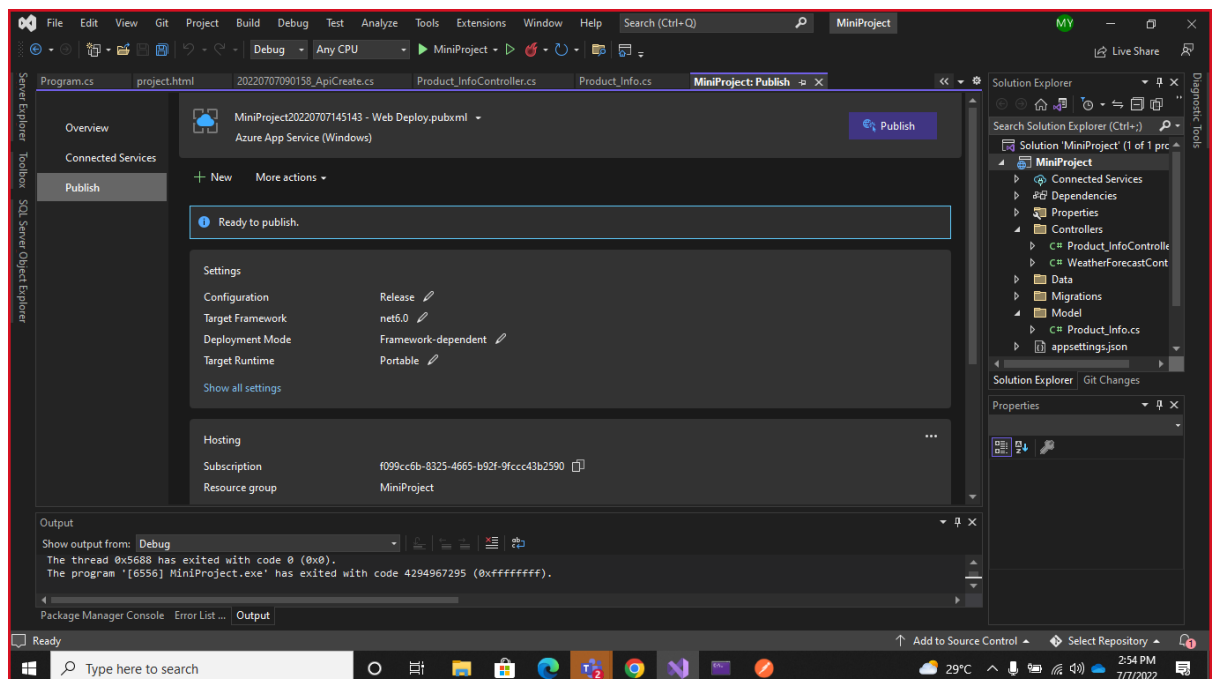
Creating the App Service for Windows.



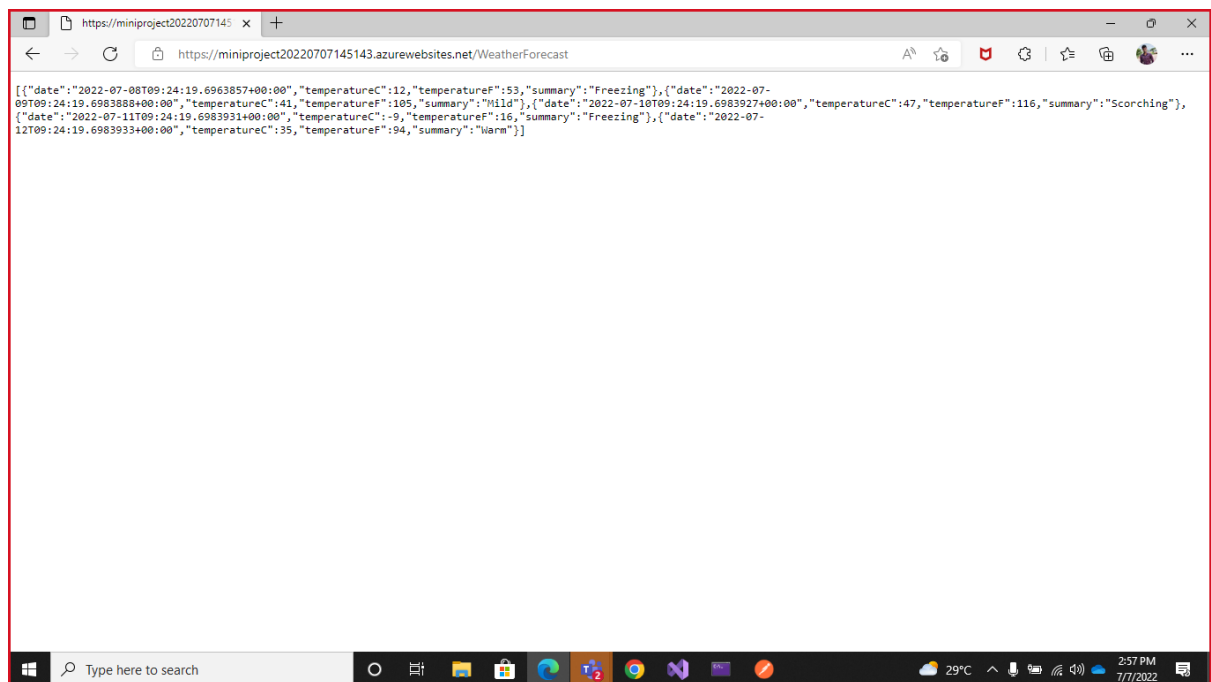
Configuring the Resource Group to Publish.



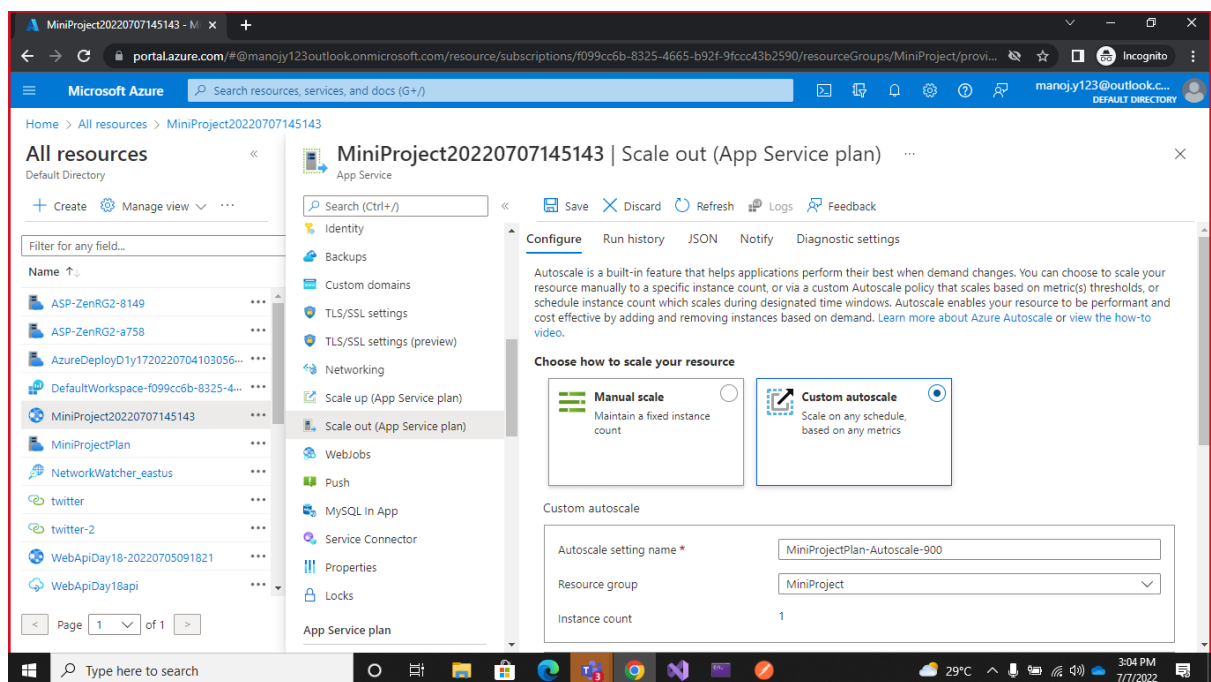
Successful configuration of API to Azure and ready to Publish.



Hosting the Web API.

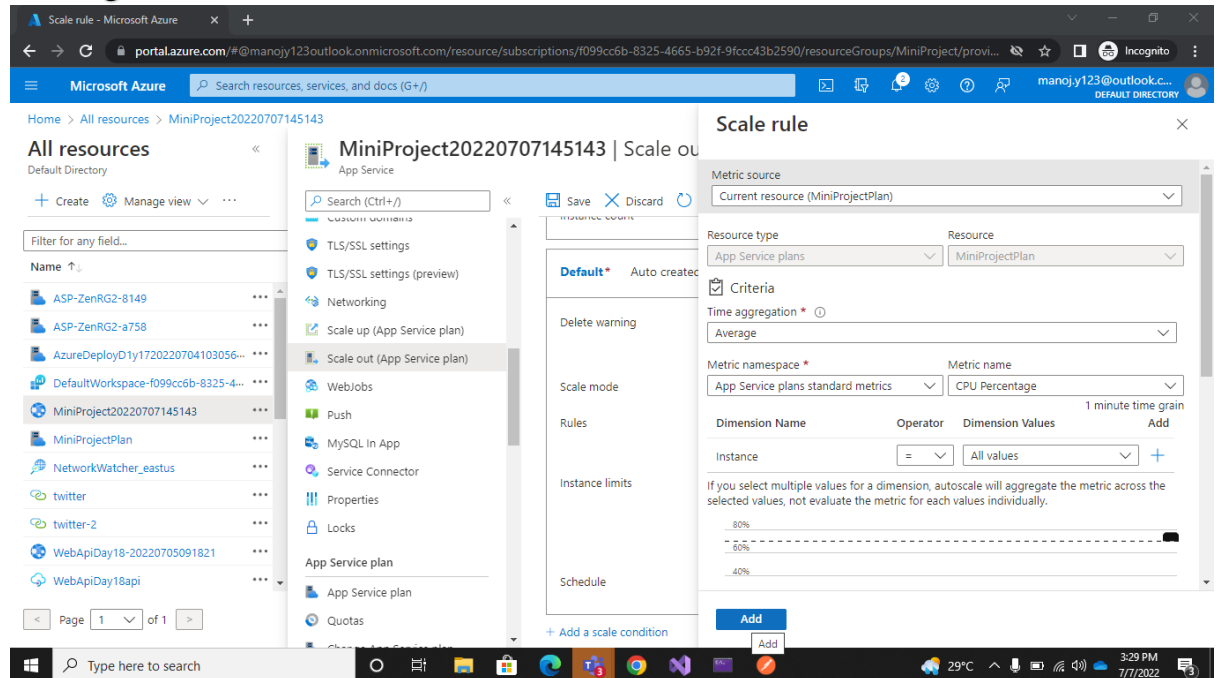


2. Configure Scale out by adding rules for custom scaling.

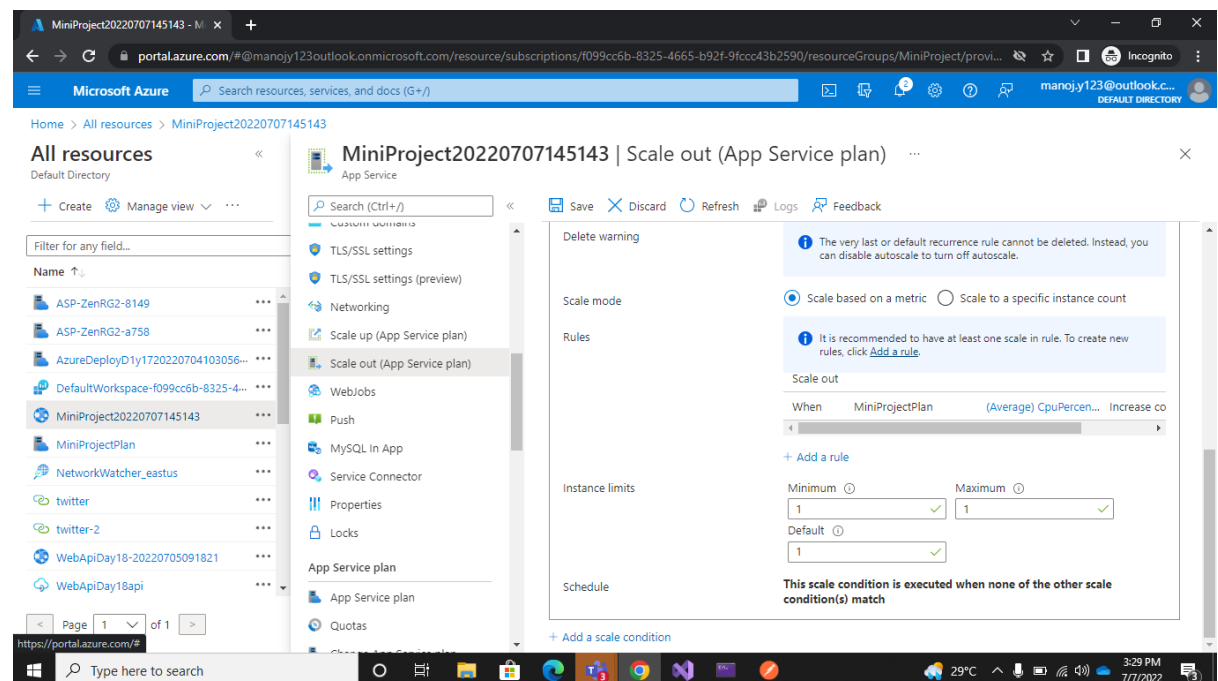


Scale Out: A scale out operation is the equivalent of creating multiple copies of your web site and adding a load balancer to distribute the demand between them. When you scale out a web site in Windows Azure Web Sites there is no need to configure load balancing separately since this is already provided by the platform.

Adding a Scale Rule.



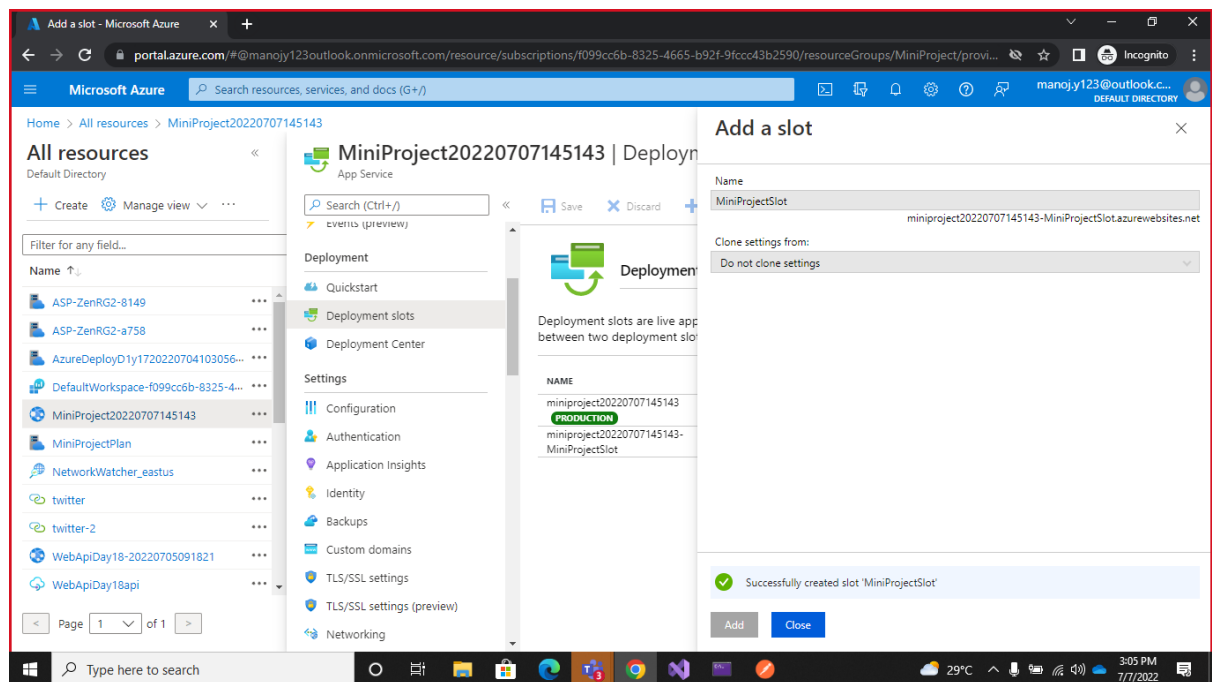
Configuring Scale Out by adding rules for custom scaling.



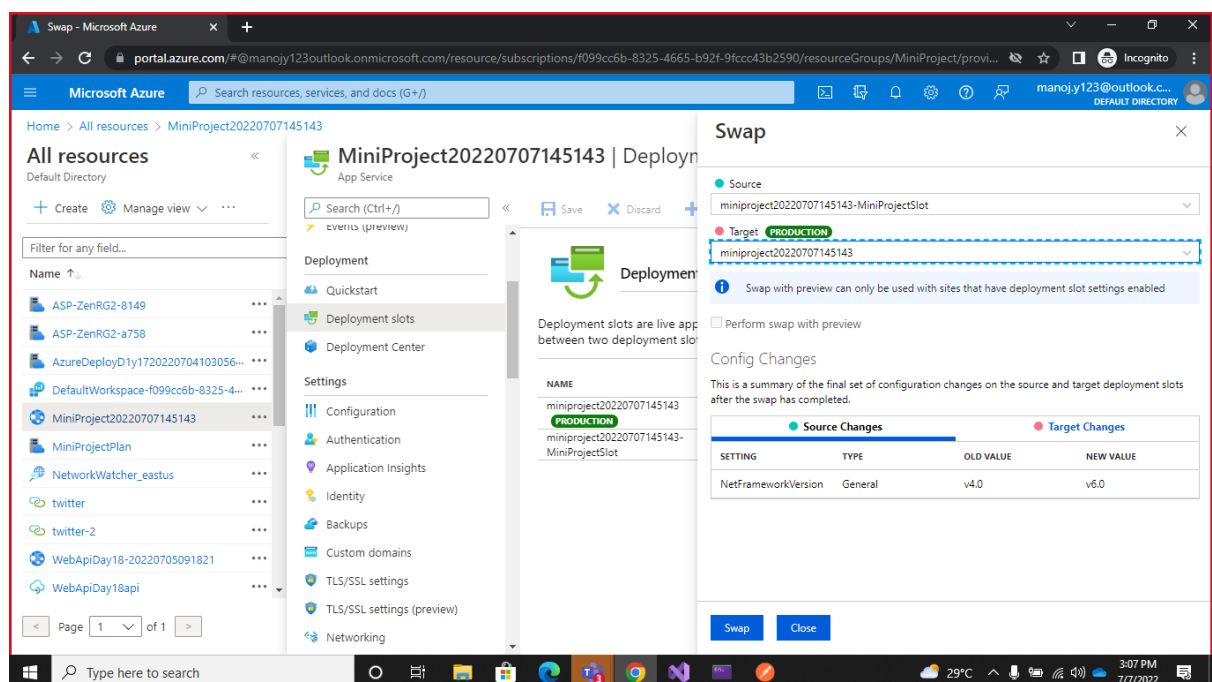
3. Configure Deployment slots for staging and production.

Deployment Slots: In Azure App Services, you can very easily add an additional deployment slot. This is a full-fledged App Service – in this case, another Web App – that sits next to your original Web App. The deployment slot has a different URL, maybe something like staging.website.com.

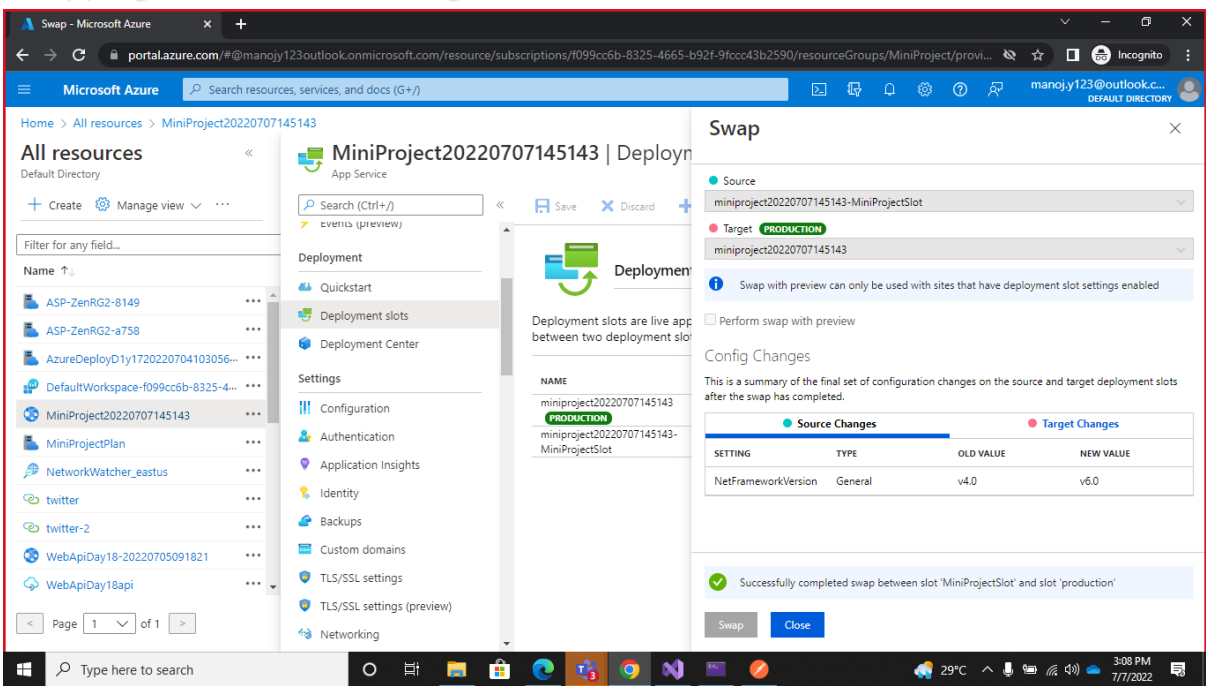
Adding a Deployment Slot.



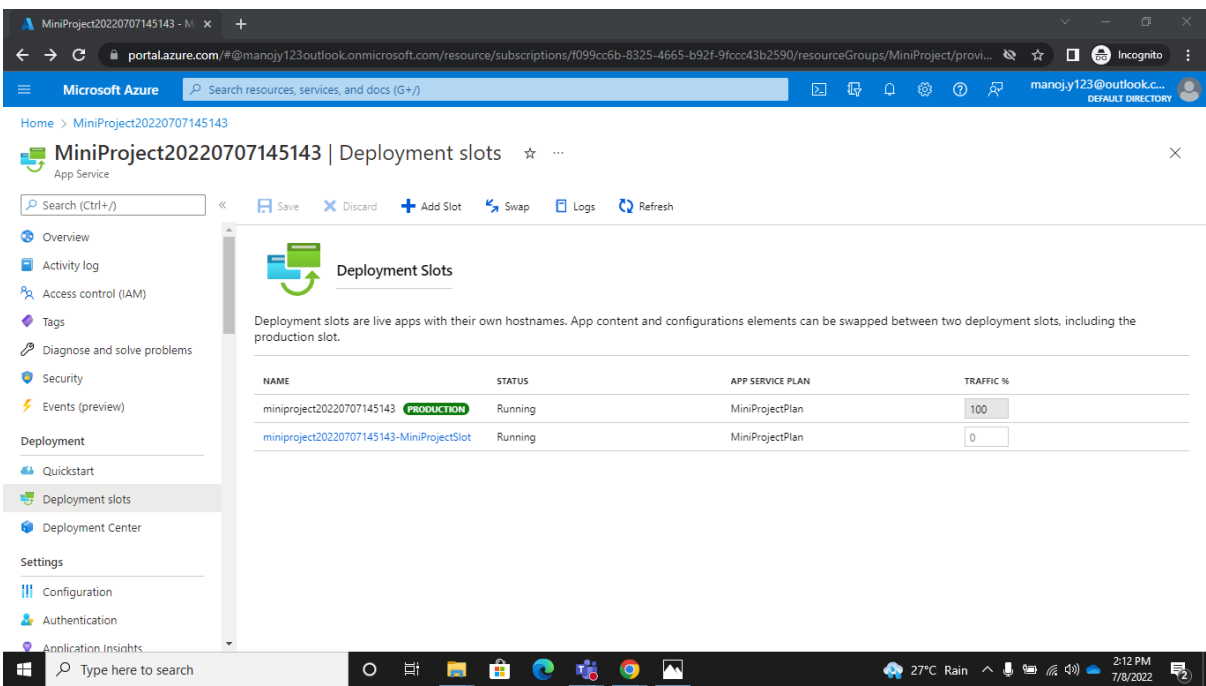
Configuring Deployment slots for Staging and production.



Swapping the Source and Target.



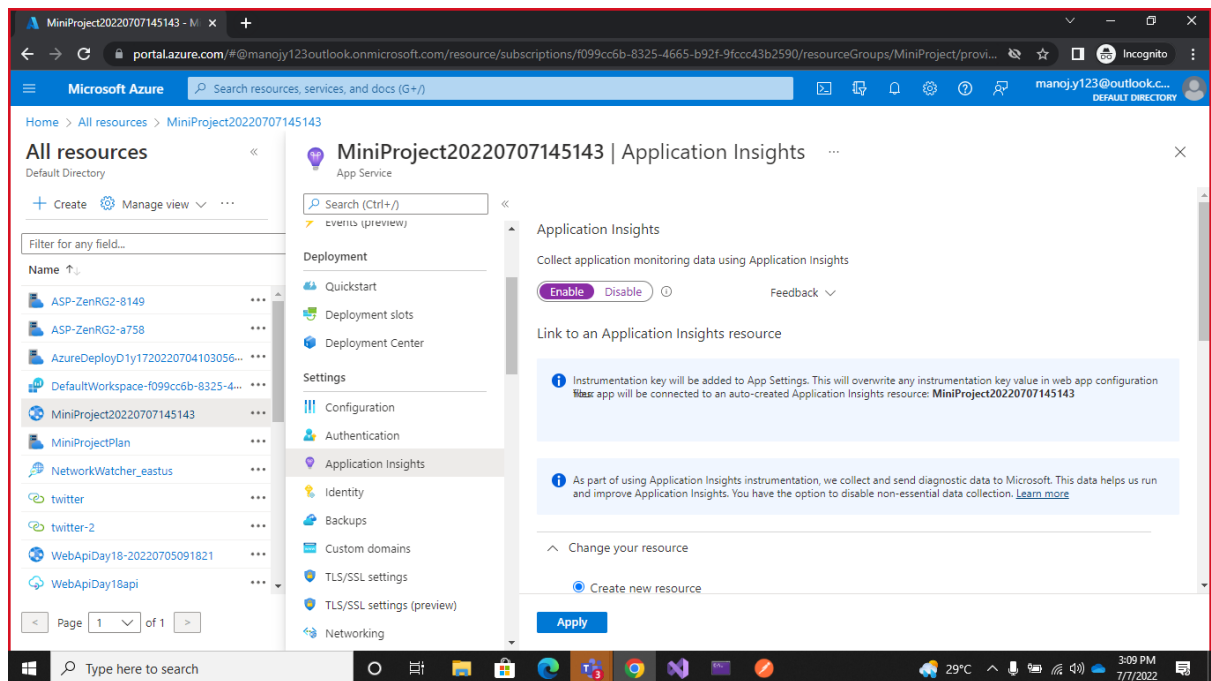
Successful Creation of Deployment Slots.



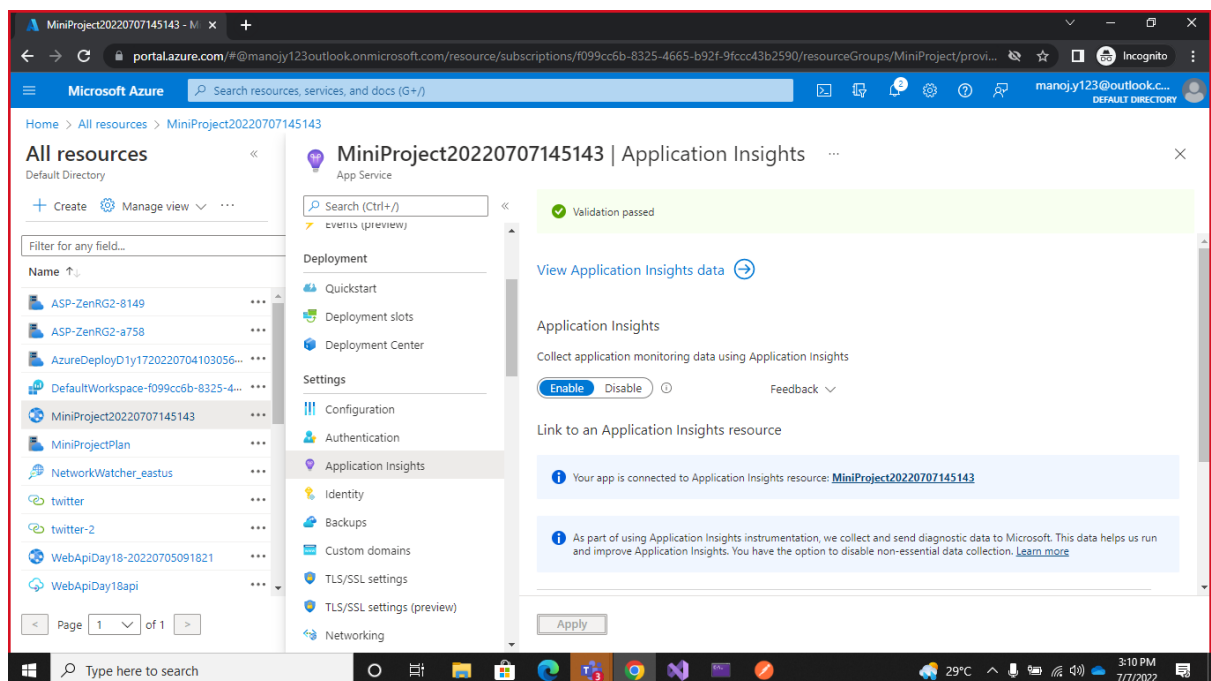
4. Configure Application Insights for the project.

Application Insights:

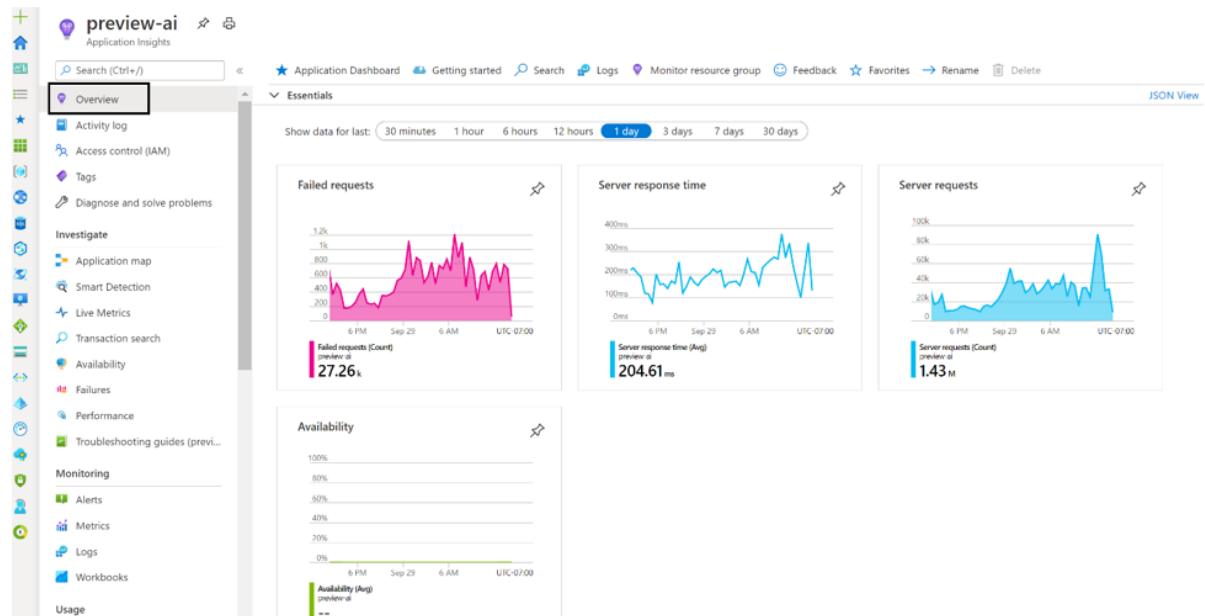
Application Insights, a feature of Azure Monitor, is widely used within the enterprise landscape for monitoring and diagnostics. Data that has already been collected from a specific tenant or environment is pushed to your own Application Insights environment.



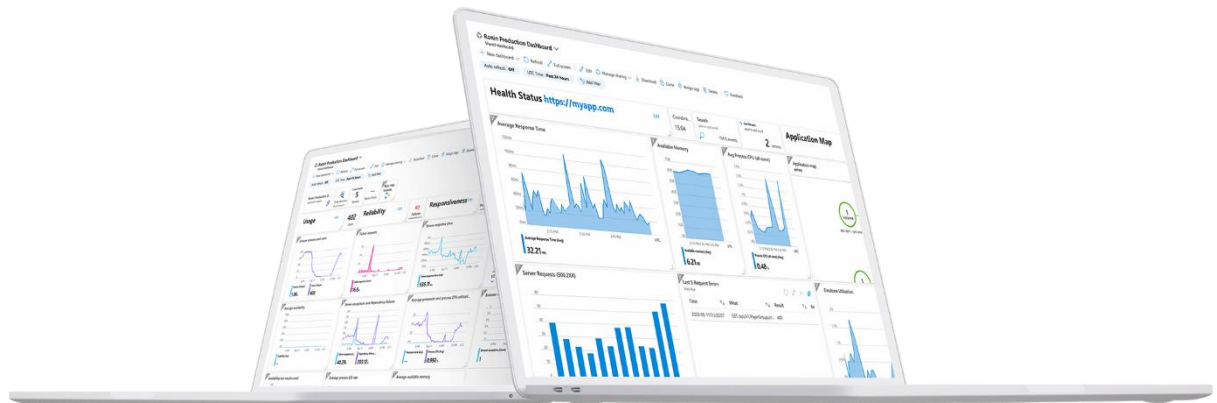
Validation of Application Insights.



Application Insights Overview.



Application insights provides different views. The **Overview** panel shows a summary of the key diagnostic metrics of your app and is a gateway to the other features of the portal



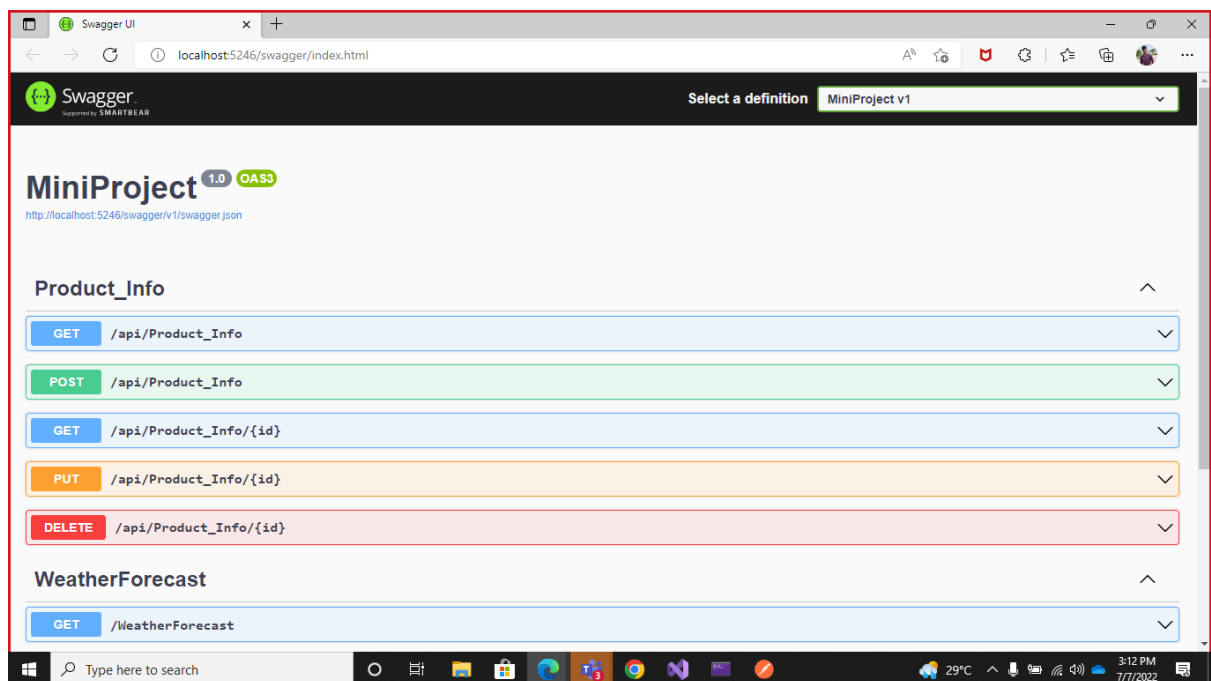
Application Insights has a wide range of features to help you use this data:

- Create a dashboard for an overview of the health of your org.
- Perform proactive monitoring by using Smart Detection.
- Set up alerts for important scenarios based on your org.
- Visualize and track common navigation patterns from a usage perspective. This will help you understand, for example, whether a user always selects a specific tab first before navigating back to the main tab and closing the form. If so, this might indicate that a field should be positioned on the first tab, instead of another tab, to save the user time every time they open this record.
- Create custom queries to troubleshoot performance and errors by using the **Logs** panel under **Monitoring** on the left pane.

5. Configure Swagger for the API.

Swagger: Swagger allows you to describe the structure of your APIs so that machines can read them. The ability of APIs to describe their own structure is the root of all awesomeness in Swagger. By reading your API's structure, we can automatically build beautiful and interactive API documentation. We can also automatically generate client libraries for your API in many languages and explore other possibilities like automated testing. Swagger does this by asking your API to return a YAML or JSON that contains a detailed description of your entire API. This file is essentially a resource listing of your API which adheres to [OpenAPI Specification](#).

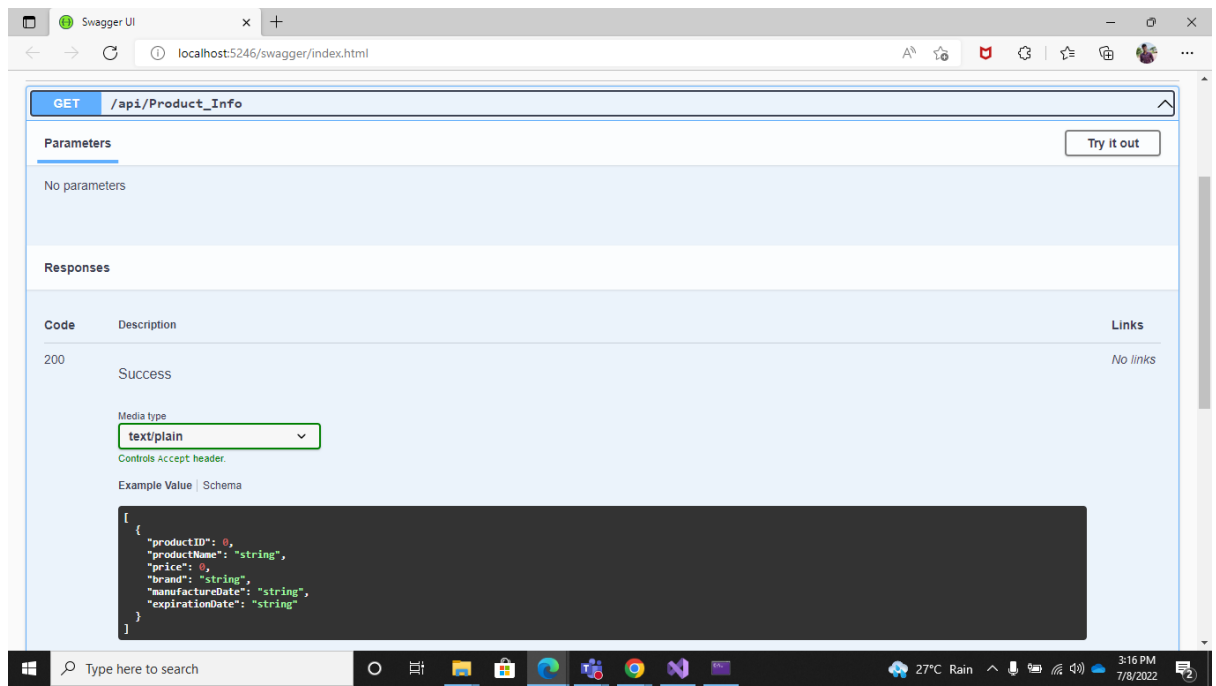
Swagger for Product Information.



Advantages of Swagger:

- Synchronizes the API documentation with the server and client at the same pace.
- Allows us to generate REST API documentation and interact with the REST API. The interaction with the REST API using the Swagger UI Framework gives clear insight into how the API responds to parameters.
- Provides responses in the format of JSON and XML.
- Implementations are available for various technologies, such as Scala, Java, and HTML5.

GET:

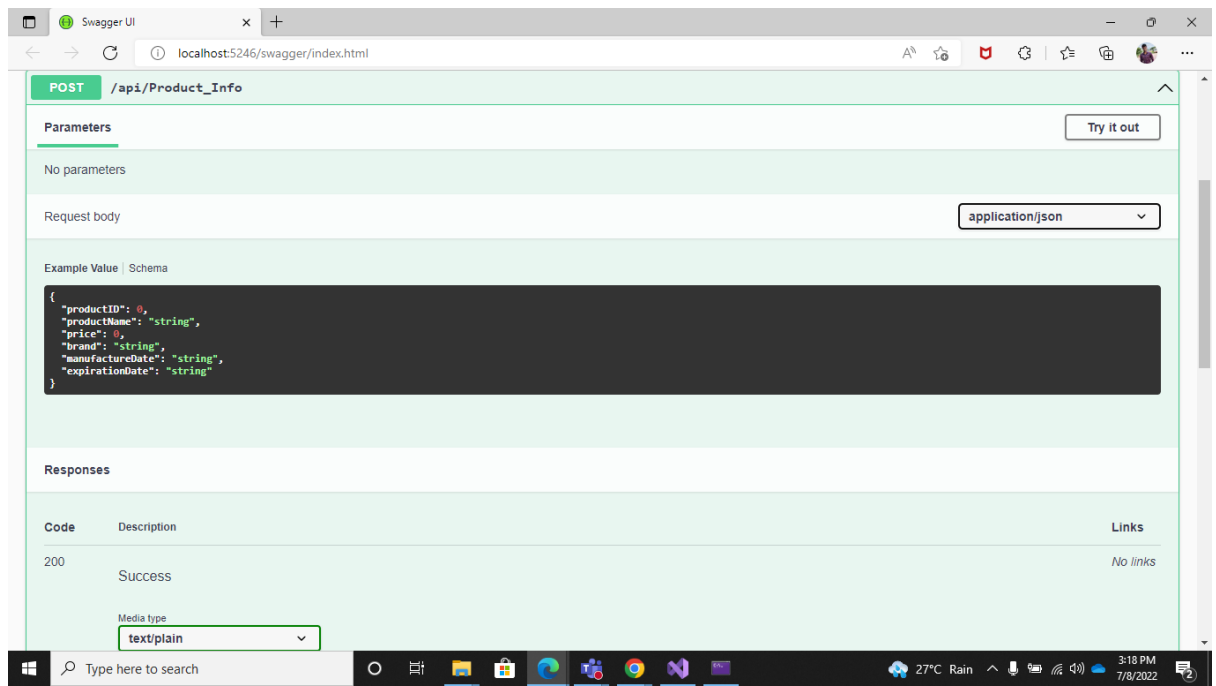


A screenshot of the Swagger UI interface in a web browser. The browser tab is titled "Swagger UI" and the address bar shows "localhost:5246/swagger/index.html". The main header displays the HTTP method "GET" and the endpoint "/api/Product_Info". Below the header, the "Parameters" section indicates "No parameters". The "Responses" section shows a table with one entry: a 200 status code with the description "Success". Under this response, there is a "Media type" dropdown menu set to "text/plain" and a "Controls Accept header" link. An "Example Value" section displays a JSON object:

```
{  "productID": 0,  "productName": "string",  "price": 0,  "brand": "string",  "manufactureDate": "string",  "expirationDate": "string"}
```

. A "Try it out" button is located in the top right corner of the response section. The Windows taskbar at the bottom shows the search bar, several application icons, and system information including "27°C Rain" and the time "3:16 PM 7/8/2022".

POST:

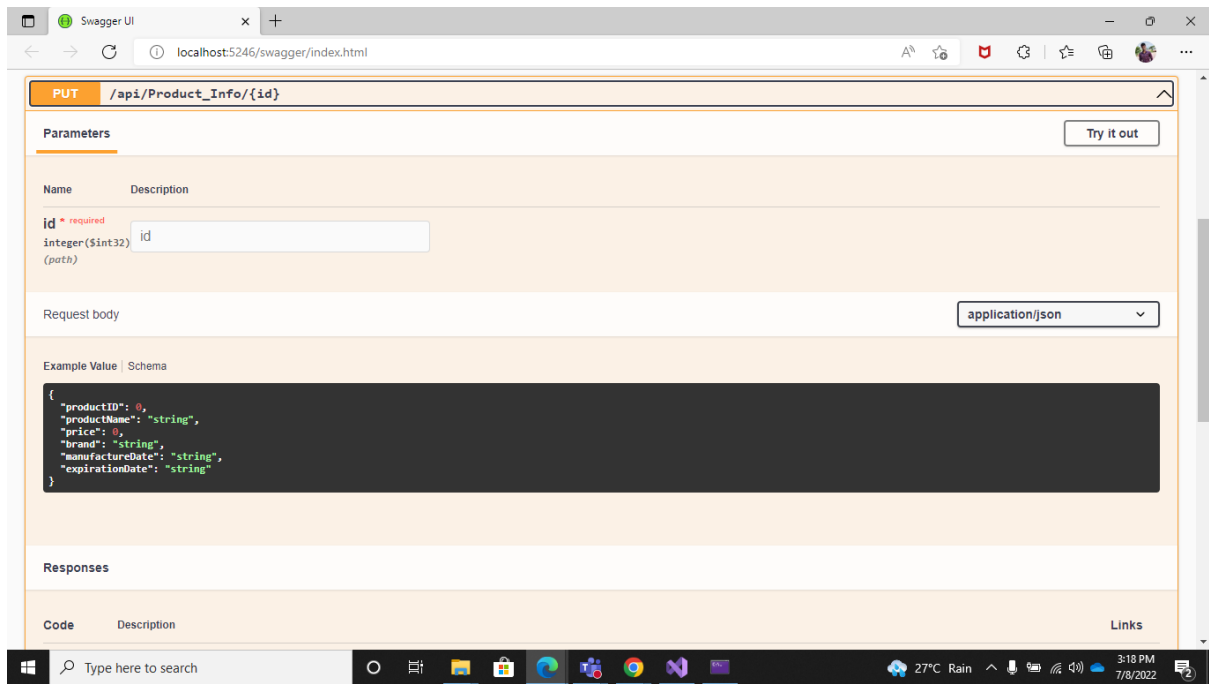


A screenshot of the Swagger UI interface in a web browser, showing the "POST" method for the endpoint "/api/Product_Info". The browser tab is titled "Swagger UI" and the address bar shows "localhost:5246/swagger/index.html". The main header displays the HTTP method "POST" and the endpoint "/api/Product_Info". Below the header, the "Parameters" section indicates "No parameters". The "Request body" section features a dropdown menu set to "application/json". An "Example Value" section displays a JSON object:

```
{  "productID": 0,  "productName": "string",  "price": 0,  "brand": "string",  "manufactureDate": "string",  "expirationDate": "string"}
```

. The "Responses" section shows a table with one entry: a 200 status code with the description "Success". Under this response, there is a "Media type" dropdown menu set to "text/plain". A "Try it out" button is located in the top right corner of the response section. The Windows taskbar at the bottom shows the search bar, several application icons, and system information including "27°C Rain" and the time "3:18 PM 7/8/2022".

PUT:



The image shows the Swagger UI for a PUT endpoint. The browser address bar displays 'localhost:5246/swagger/index.html'. The endpoint is '/api/Product_Info/{id}' with a PUT method. A 'Try it out' button is present. The parameters section shows a required 'id' of type 'integer(\$int32)' as a path parameter. The request body is set to 'application/json'. An example value is shown in a dark box:

```
{  "productID": 0,  "productName": "string",  "price": 0,  "brand": "string",  "manufactureDate": "string",  "expirationDate": "string"}
```

. The responses section is currently empty.

Name	Description
id * required	id
integer(\$int32)	(path)

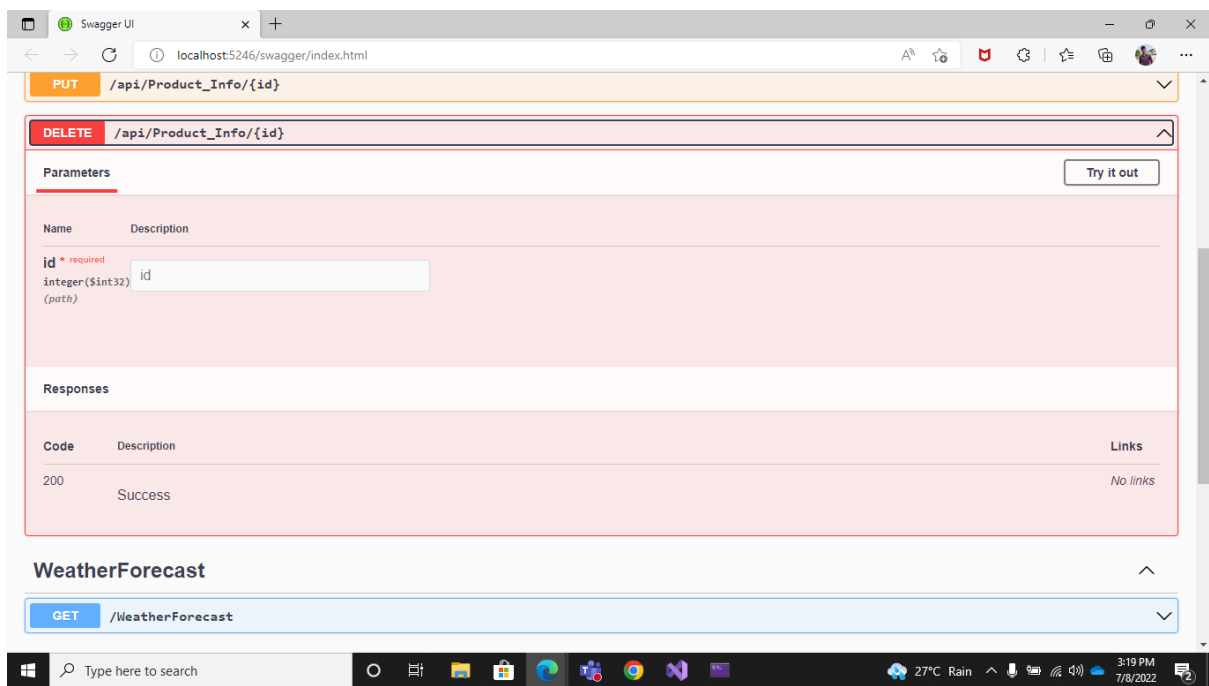
Request body: application/json

Example Value | Schema

```
{  "productID": 0,  "productName": "string",  "price": 0,  "brand": "string",  "manufactureDate": "string",  "expirationDate": "string"}
```

Code	Description	Links
------	-------------	-------

DELETE:



The image shows the Swagger UI for a DELETE endpoint. The browser address bar displays 'localhost:5246/swagger/index.html'. The endpoint is '/api/Product_Info/{id}' with a DELETE method. A 'Try it out' button is present. The parameters section shows a required 'id' of type 'integer(\$int32)' as a path parameter. The responses section shows a 200 status code with a 'Success' description and 'No links'. Below this, a 'WeatherForecast' endpoint is visible with a GET method.

Name	Description
id * required	id
integer(\$int32)	(path)

Responses

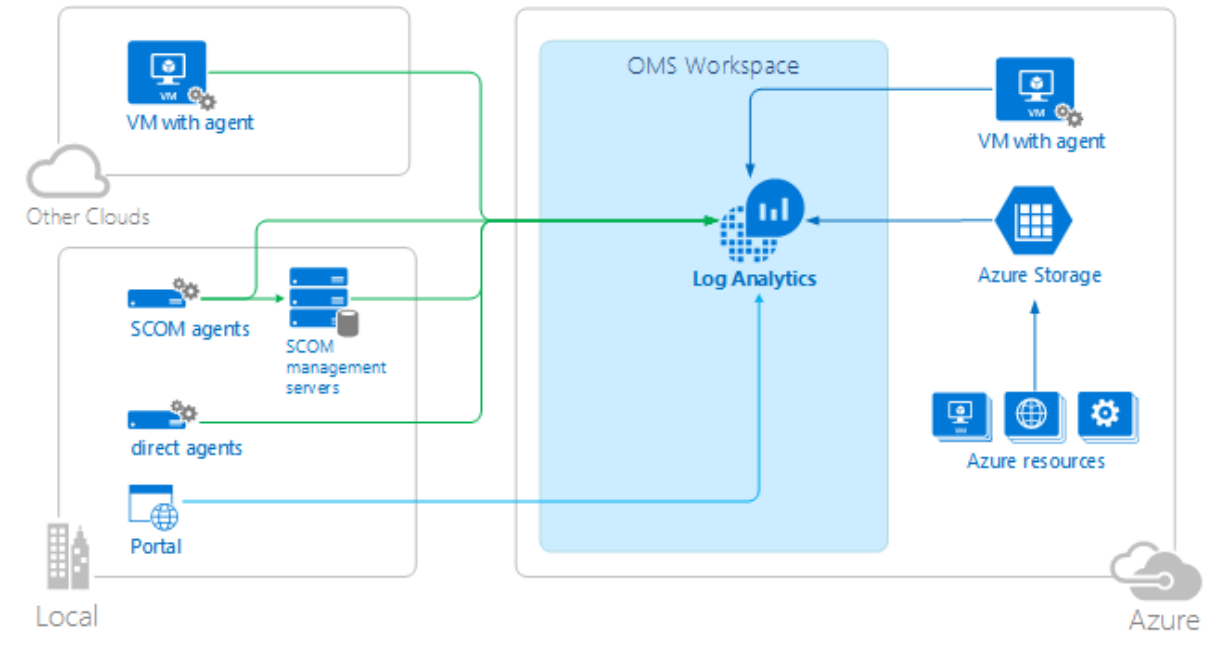
Code	Description	Links
200	Success	No links

WeatherForecast

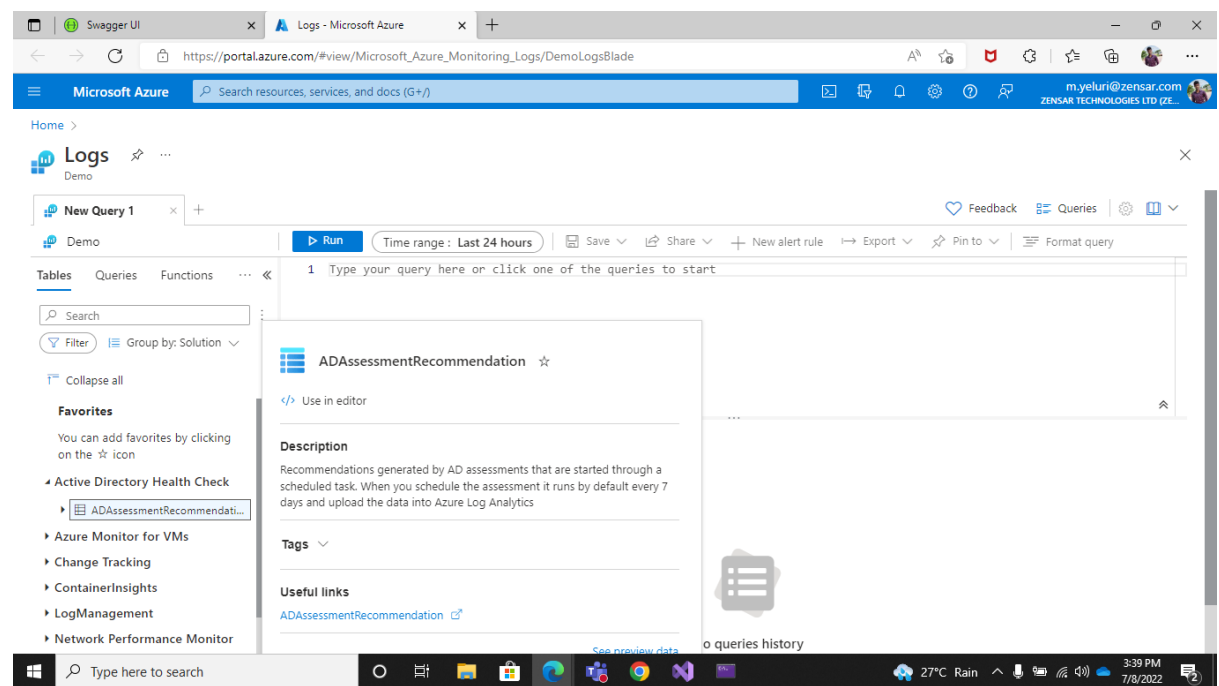
GET /WeatherForecast

6. Work with Log Analytics with the sample logs available.

Log Analytics: Log Analytics is a tool in the Azure portal that's used to edit and run log queries with data in Azure Monitor Logs.



Working with Log Analytics.



Setting the Time Range for Logs.

The screenshot shows the Microsoft Azure portal interface for the 'Logs' section. The 'Time range' dropdown menu is open, displaying options: Last 30 minutes, Last hour, Last 4 hours, Last 12 hours, Last 24 hours, Last 48 hours, Last 3 days, Last 7 days (selected), Set in query, and Custom. A link for 'Time range syntax' is also visible. The interface includes a search bar, a 'Run' button, and a 'No queries history' message.

History of Logs.

The screenshot shows the Microsoft Azure portal interface for the 'Logs' section. A query is entered: `ADAssessmentRecommendation | where _ResourceId contains "ab"`. The 'Results' tab is selected, displaying a table with the following data:

TimeGenerated [UTC]	AssessmentId	AssessmentName	RecommendationId	Recommendation
7/5/2022, 9:52:30.696 AM	22141292-68bc-4904-b030-5e894a7b1cb3	AD	e1fc9908-1810-455a-97de-5f53738141eb	Resolve Directory System
7/5/2022, 9:52:30.726 AM	22141292-68bc-4904-b030-5e894a7b1cb3	AD	c6eb7e0c-b86a-438f-9dce-9btf50293dc9	Unless specifically requi
7/5/2022, 9:52:30.726 AM	22141292-68bc-4904-b030-5e894a7b1cb3	AD	4eabc96c-682a-4d81-9919-0c32af52aa3f	Amend dynamic port cc
7/5/2022, 9:52:30.726 AM	22141292-68bc-4904-b030-5e894a7b1cb3	AD	f676b73a-7a9b-4358-962f-60b4c3569536	Dynamic Port Ranges C
7/5/2022, 9:52:30.726 AM	22141292-68bc-4904-b030-5e894a7b1cb3	AD	11d49a22-7cad-43b7-81cf-f466cf77189	Amend dynamic port cc
7/5/2022, 9:52:30.726 AM	22141292-68bc-4904-b030-5e894a7b1cb3	AD	d8640839-78cd-45a1-a942-10b536923f52	Domain Controllers witl
7/5/2022, 9:52:30.726 AM	22141292-68bc-4904-b030-5e894a7b1cb3	AD	4bcc1c2a-4168-49b8-b5bb-1d1c10ec7796	Disable the Allow Repli

Log Analytics: Log Analytics adds features specific to Azure Monitor, such as filtering by time range and the ability to create an alert rule from a query. Both tools include an explorer that lets you scan through the structure of available tables. The Azure Data Explorer web UI primarily works with tables in Azure Data Explorer databases. Log Analytics works with tables in a Log Analytics workspace.