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

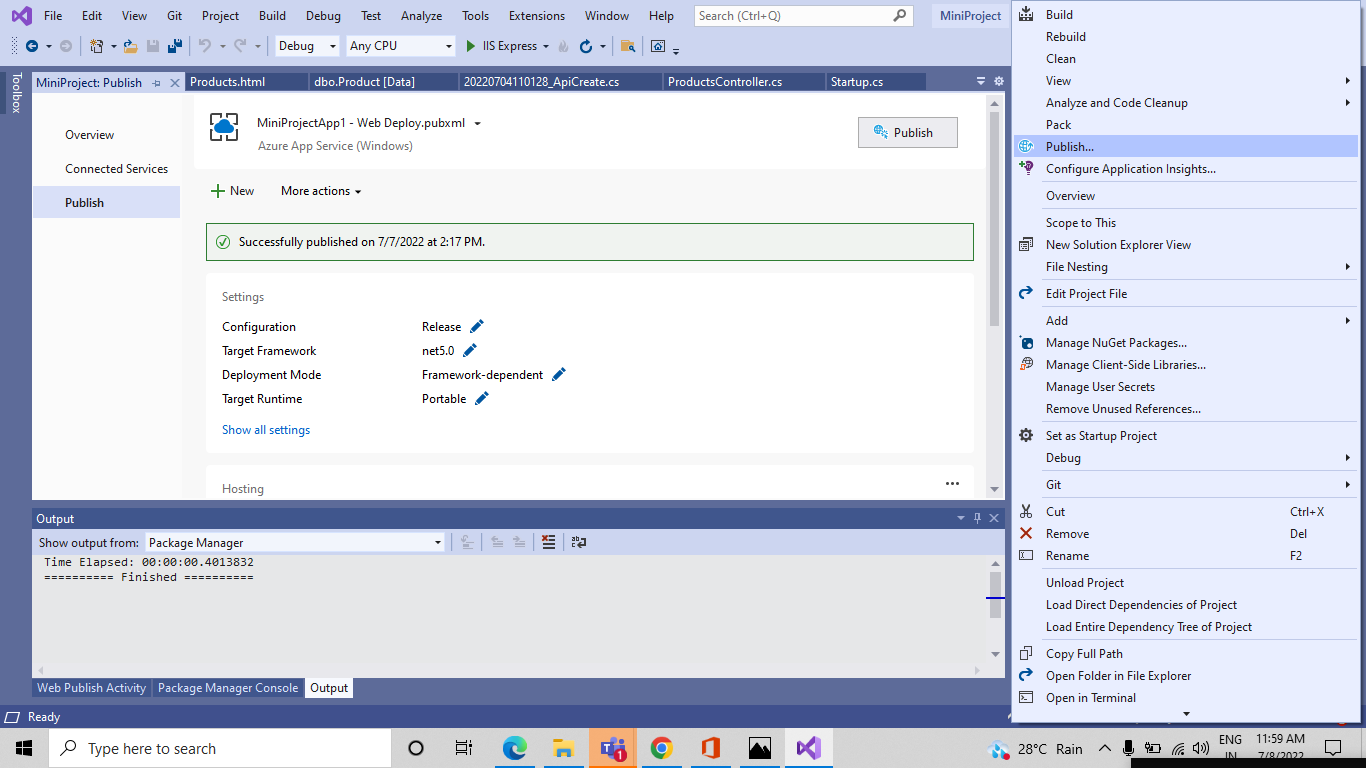
Create a JQuery and AJAX Client to consume the Web API and show the result.

**Azure Hosting:**

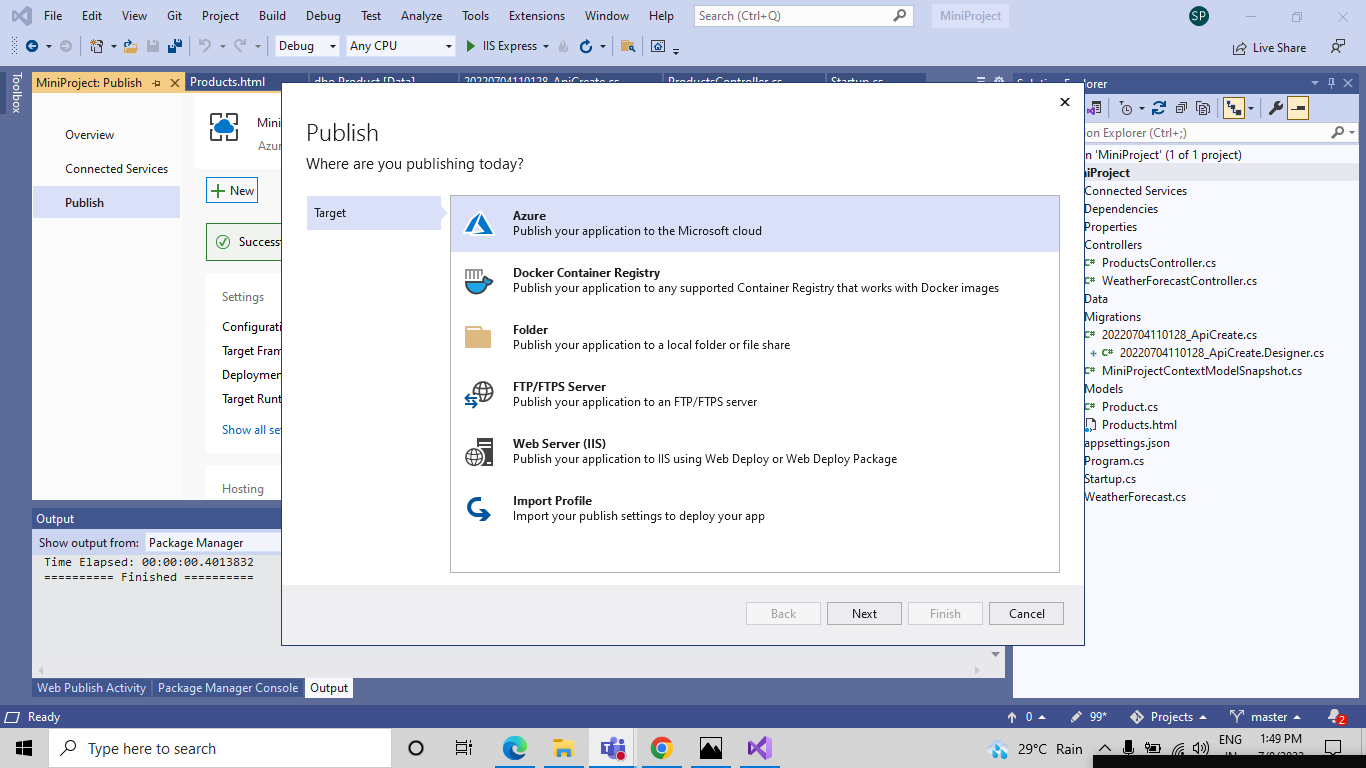
* Host the web api in azure and consume the same using JQuery Client.
* Configure Scale out by adding rules for custom scaling
* Configure Deployment slots for staging and production
* Configure Application Insights for the project
* Configure Swagger for the api
* Work with Log Analytics with the sample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

**PUBLISHING API ON AZURE :-**

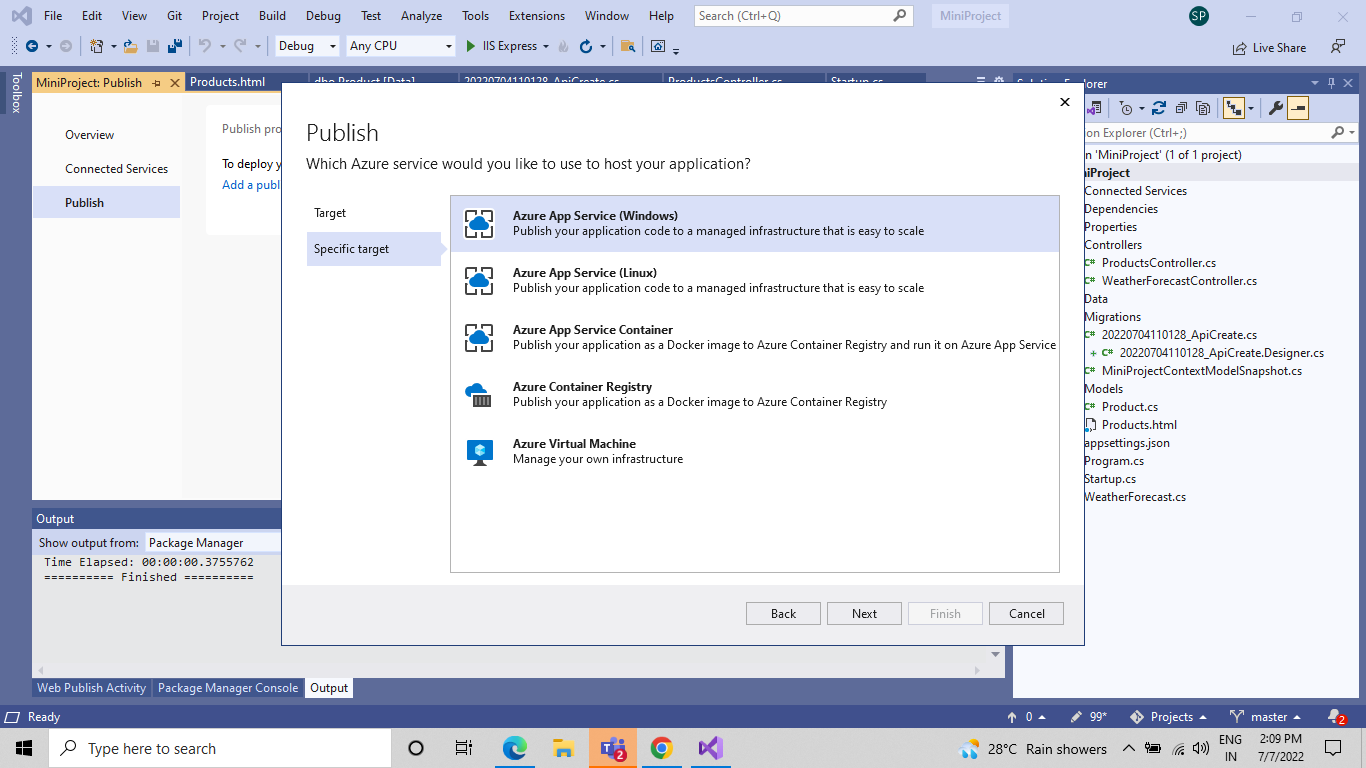
* In Solution Explorer, right-click the project and select Publish



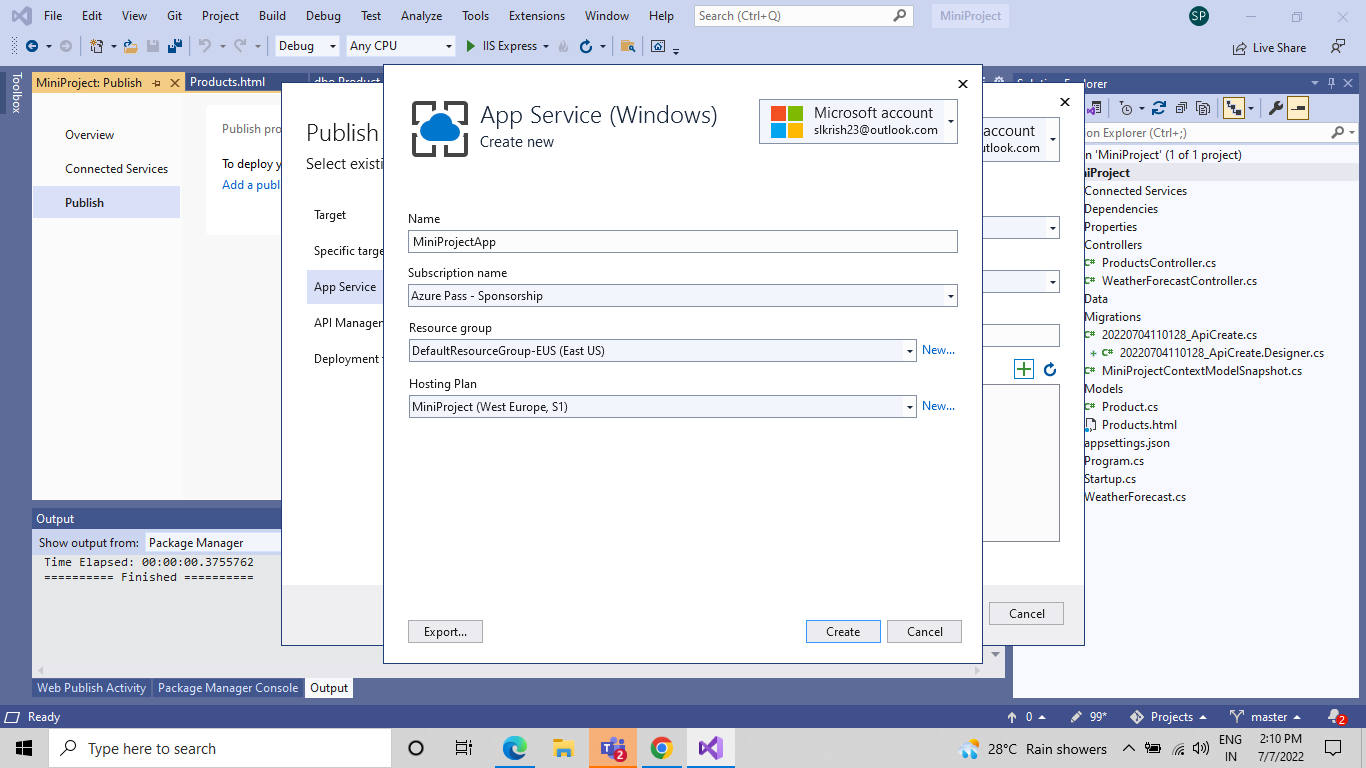
* In the Publish dialog, select Azure and select the Next button.



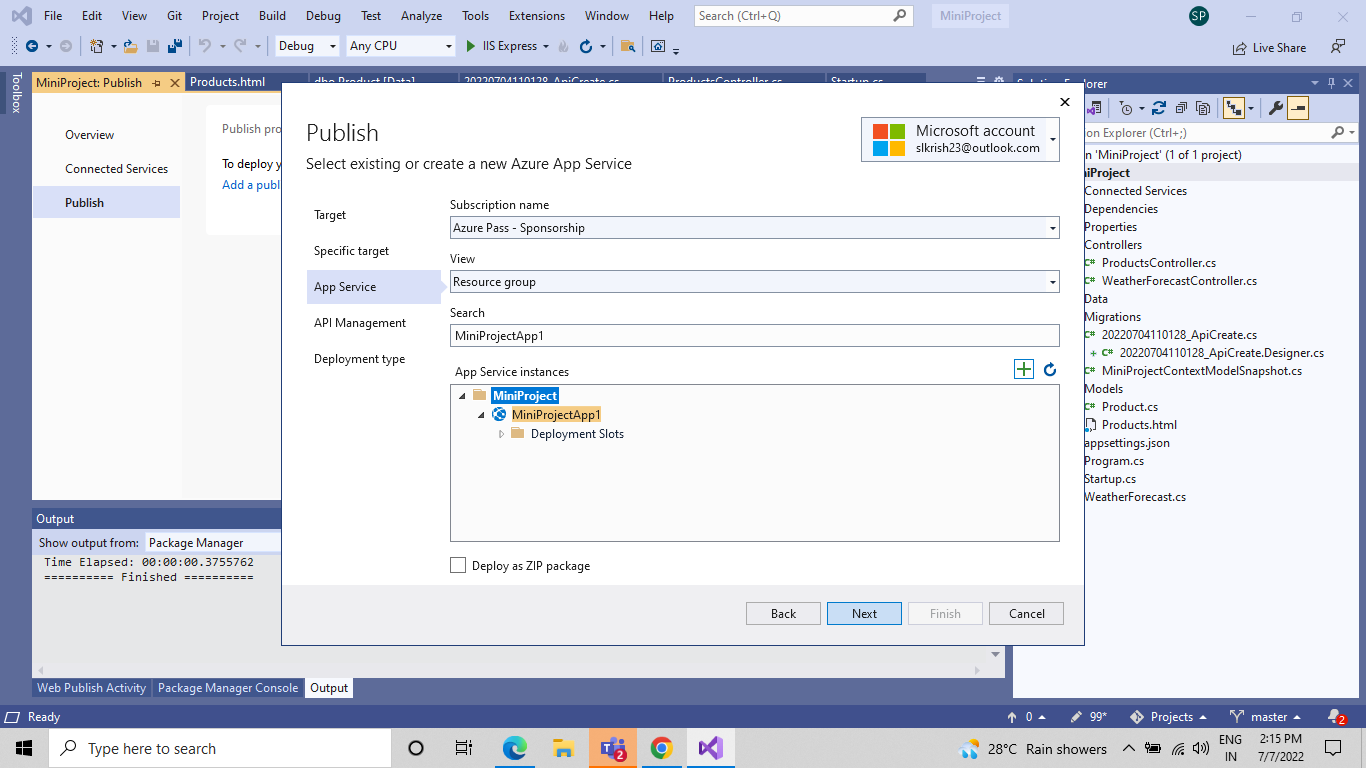
* Select Azure App Service (Windows) and select the Next button



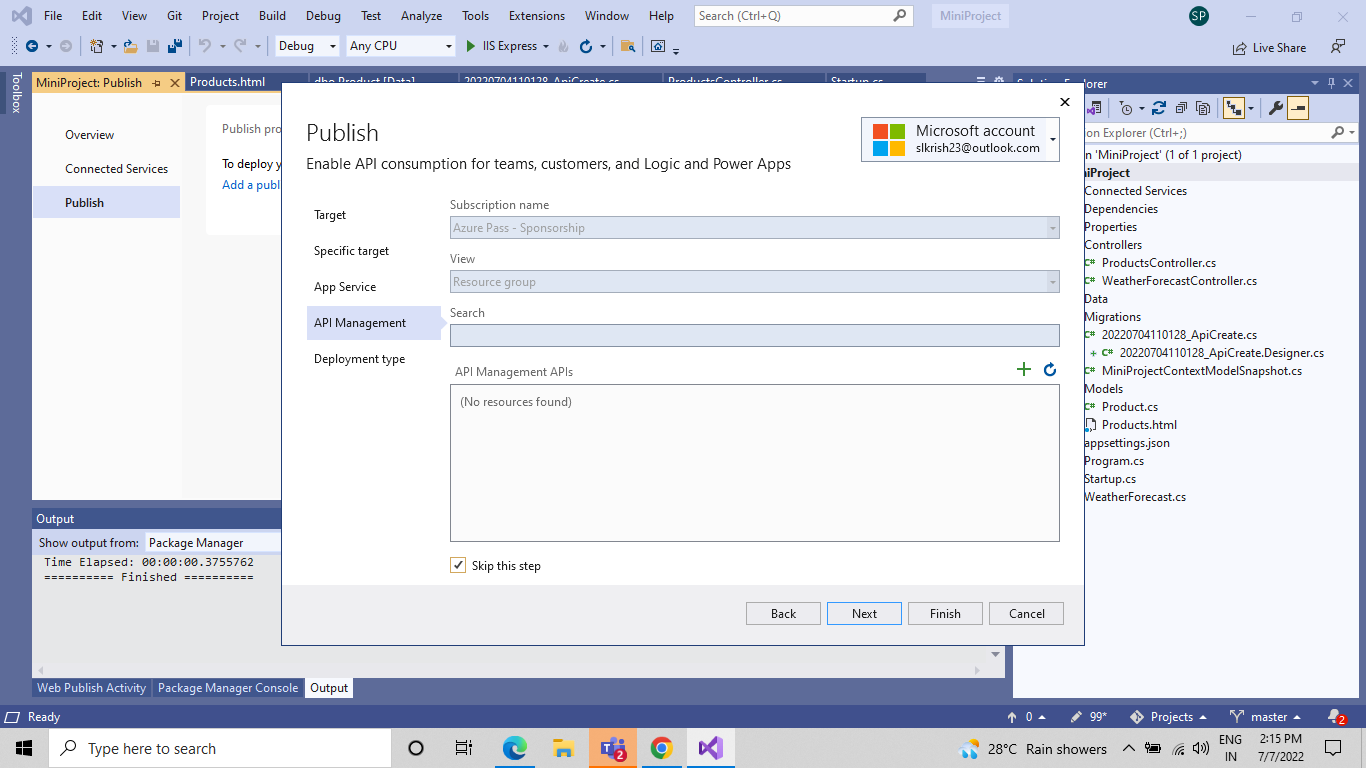
* Select Create a new Azure App Service, Select the Create button



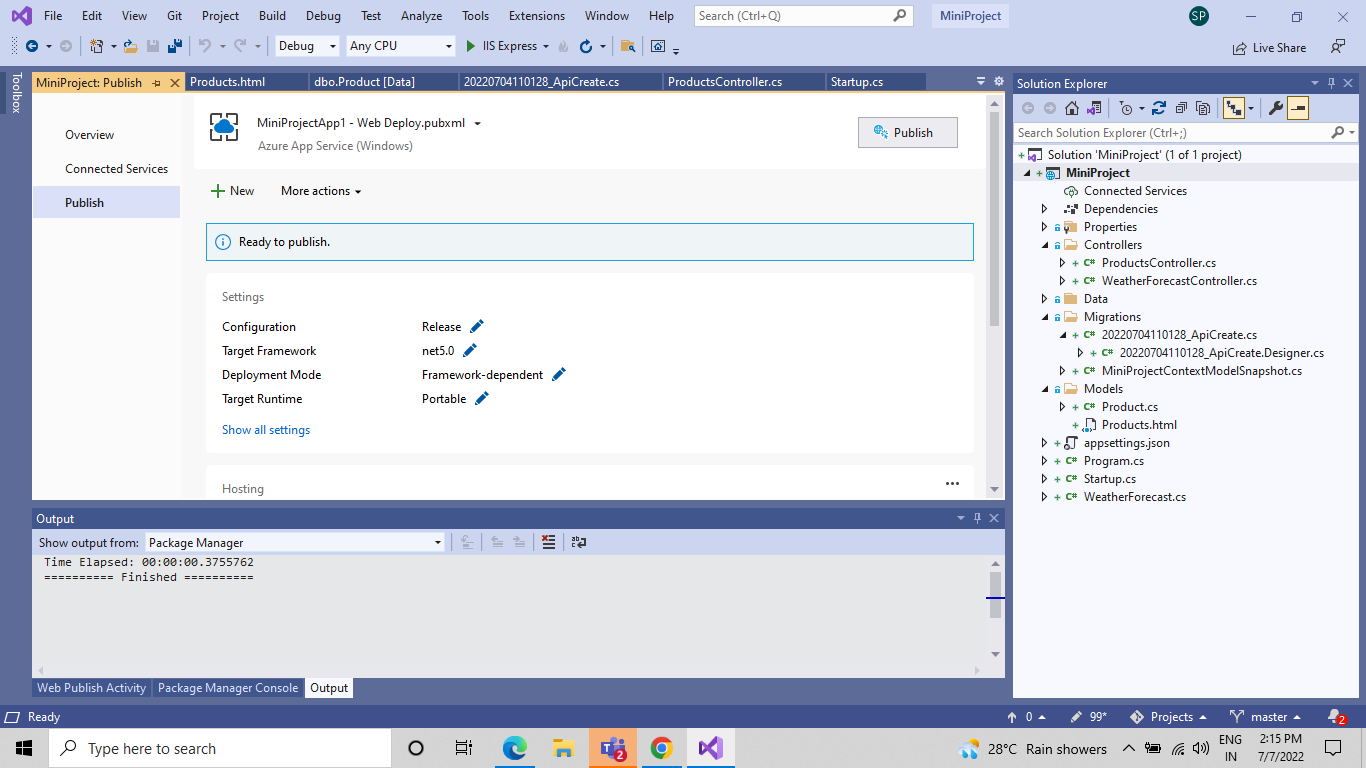
* Click On Next



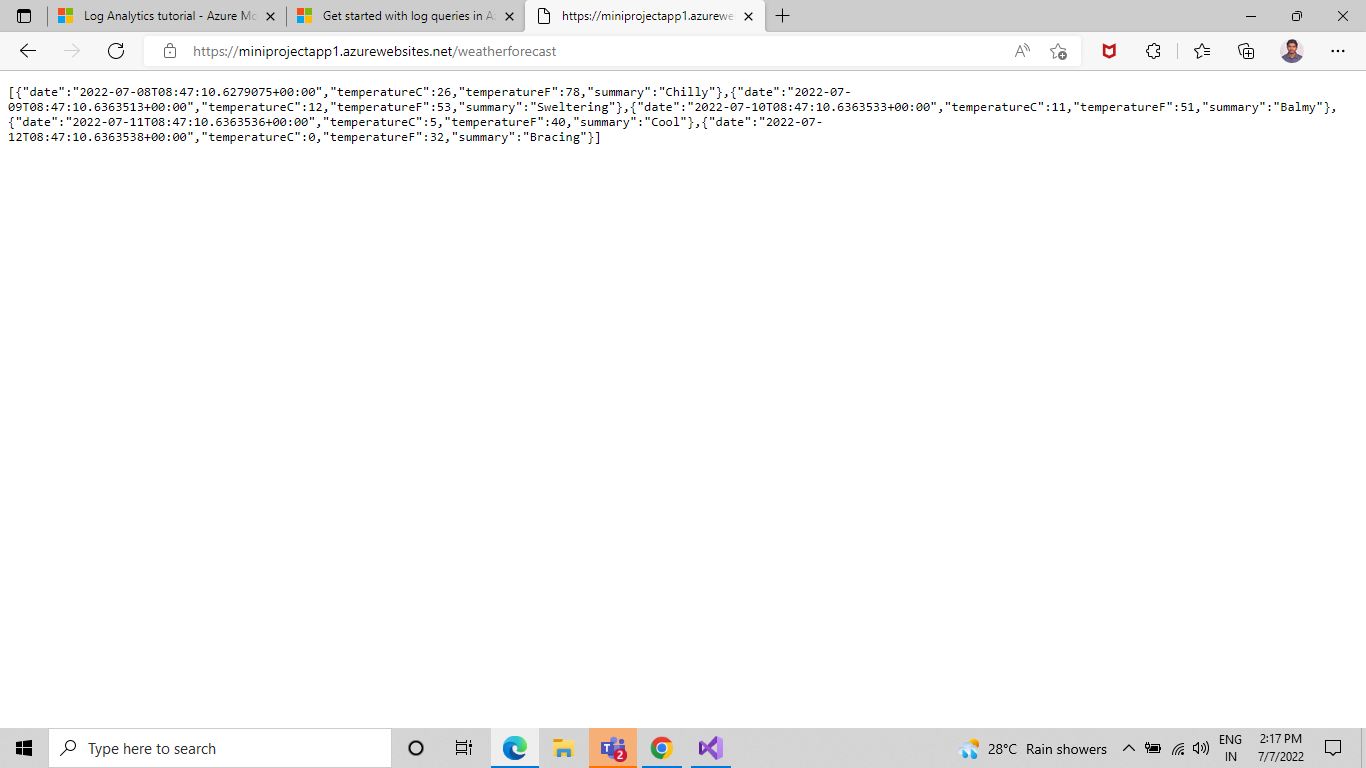
* And Click on Finish



* After creation is completed, the dialog is automatically closed and the Publish dialog gets focus again. The instance that was created is automatically selected.

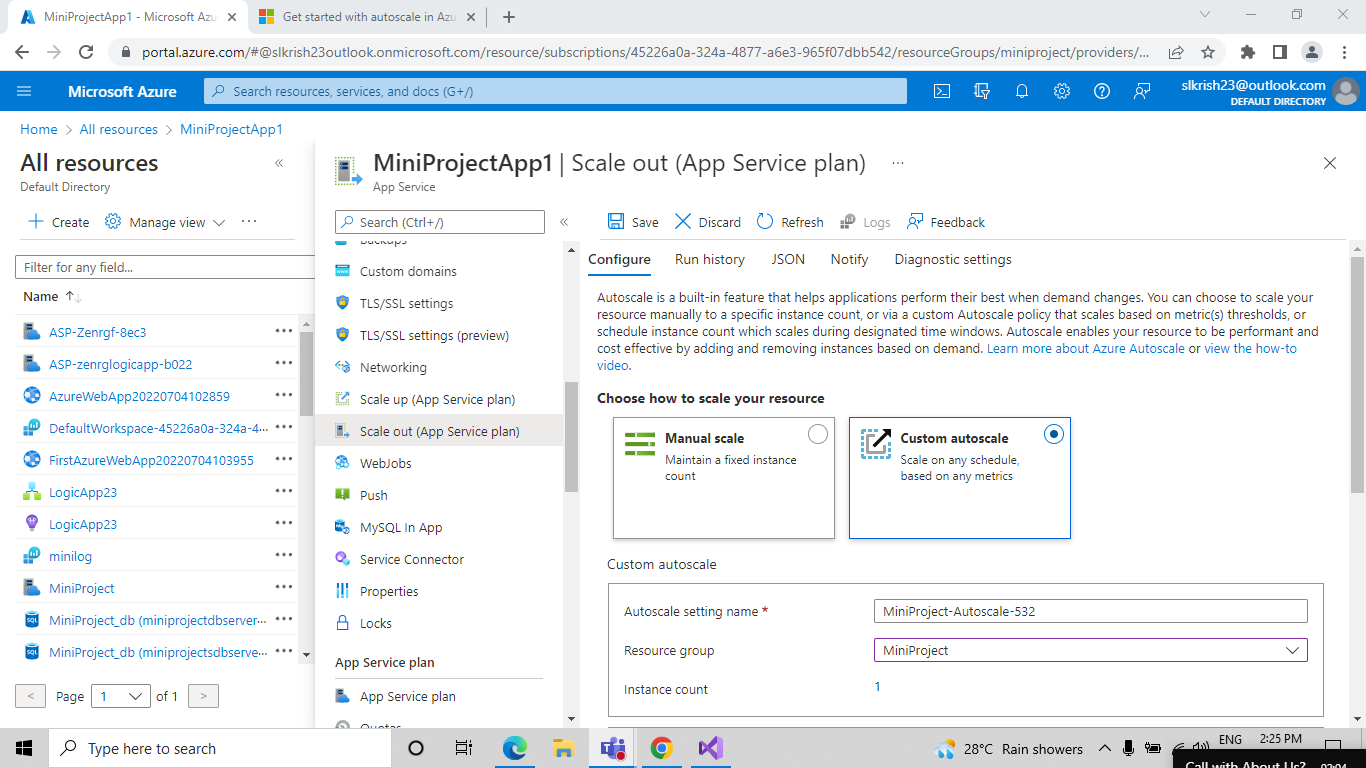


* Then if you click on Publish button you'll get JSON result

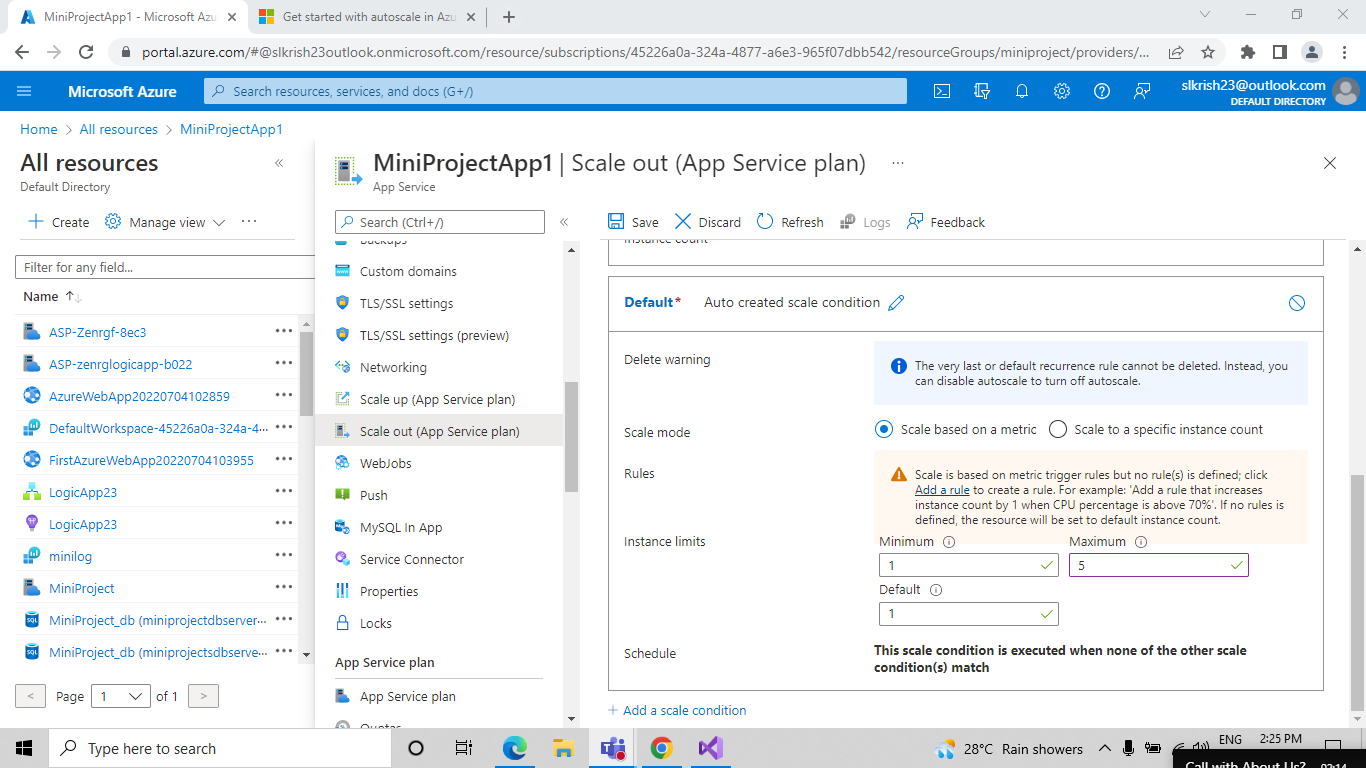


**Configuration of Scale Out :-**

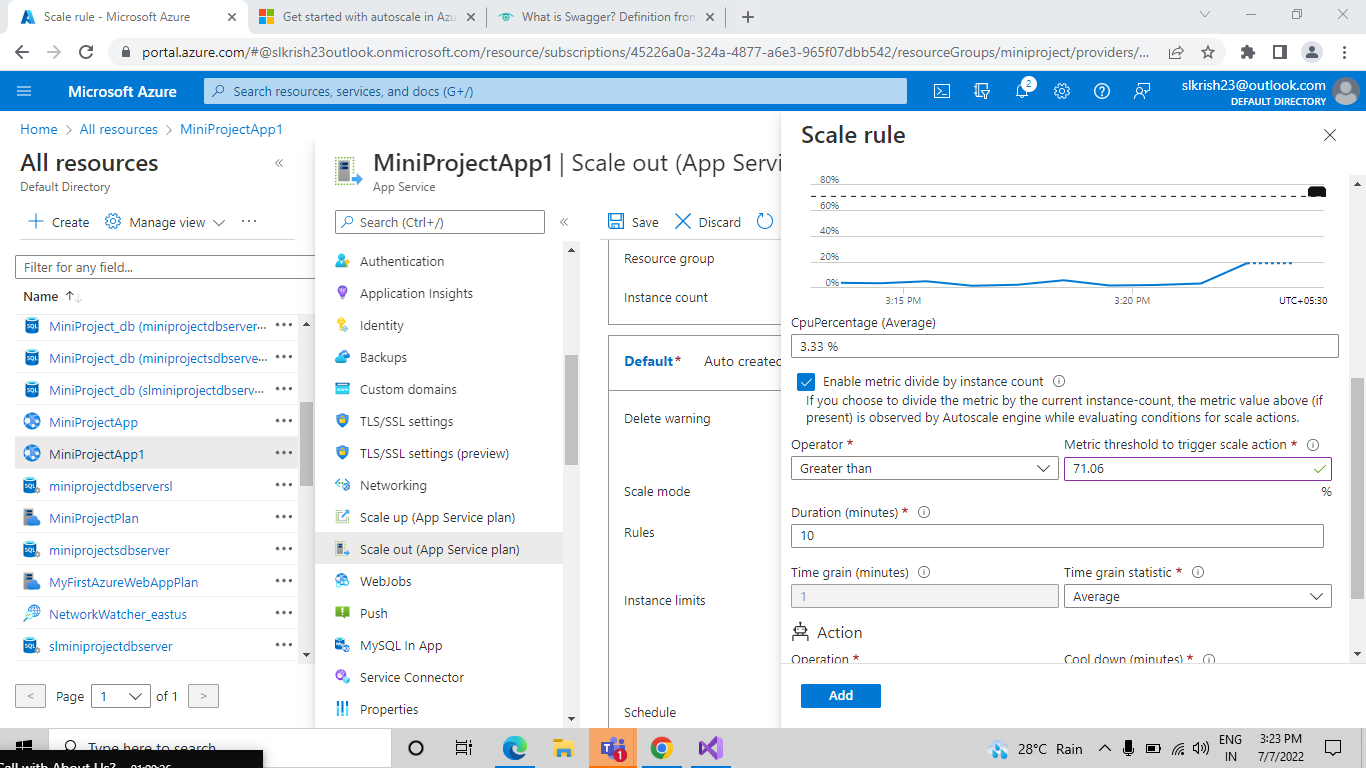
* Open Azure portal and open App Service go to settings and select scale out (App Service Plan), Select Custom autoscale.



* And then click Add a rule. This opens 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 values and click Add.

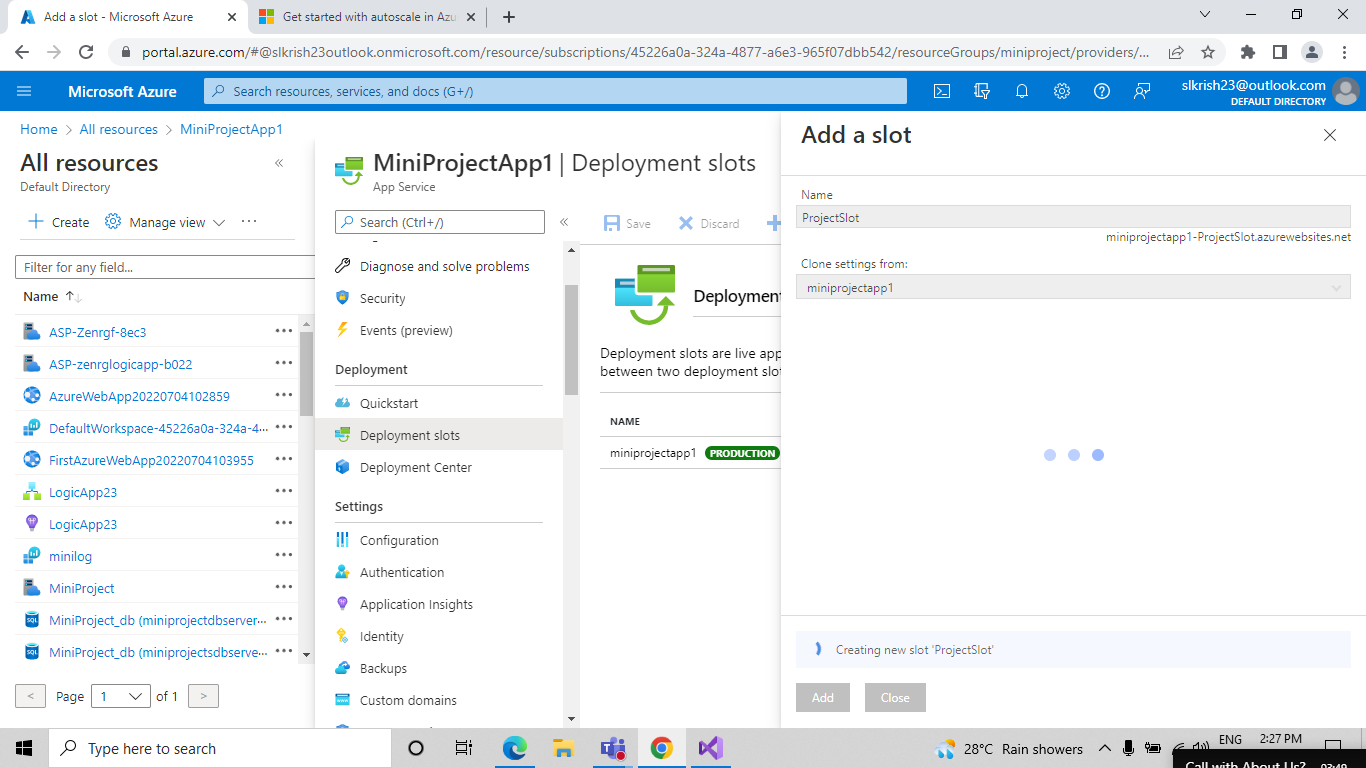


**Scale-out** – Upgrade the capacity of the app by increasing the number of host instances (PAAS Environment). Ex: Having a Load Balancer where your app is hosted on multiple instances. You can do a scale-out in any of the following two ways based on the Tier of your App Service Plan.

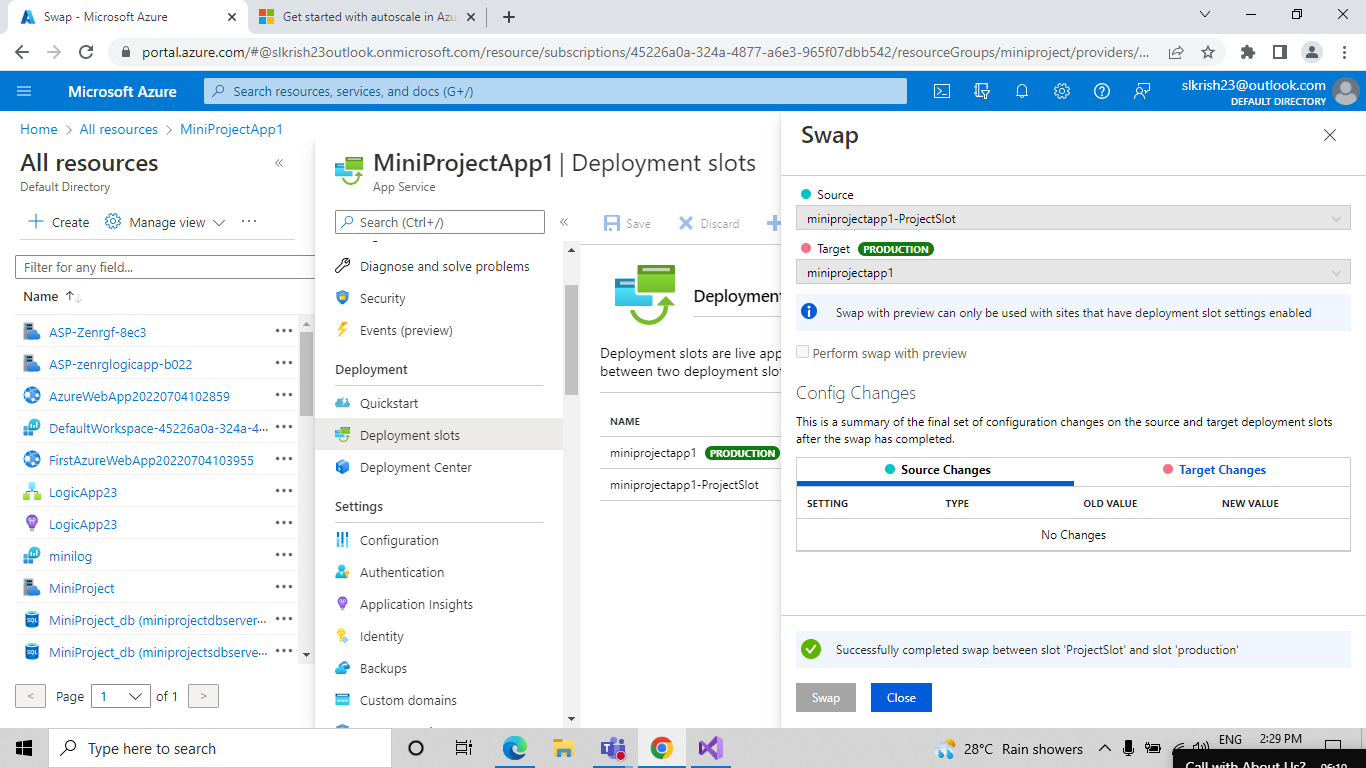
Autoscale is a built-in feature that helps applications perform their best when demand changes. You can choose to scale your resource manually to a specific instance count, or via a custom Autoscale policy that scales based on metric(s) thresholds, or schedule instance count which scales during designated time windows. Autoscale enables your resource to be performant and cost-effective by adding and removing instances based on demand.

**Configuration of Deployment Slots :-**

* Select Deployment slots, and then select **+** Add Slot.



* Verify the configuration settings for your swap and select Swap

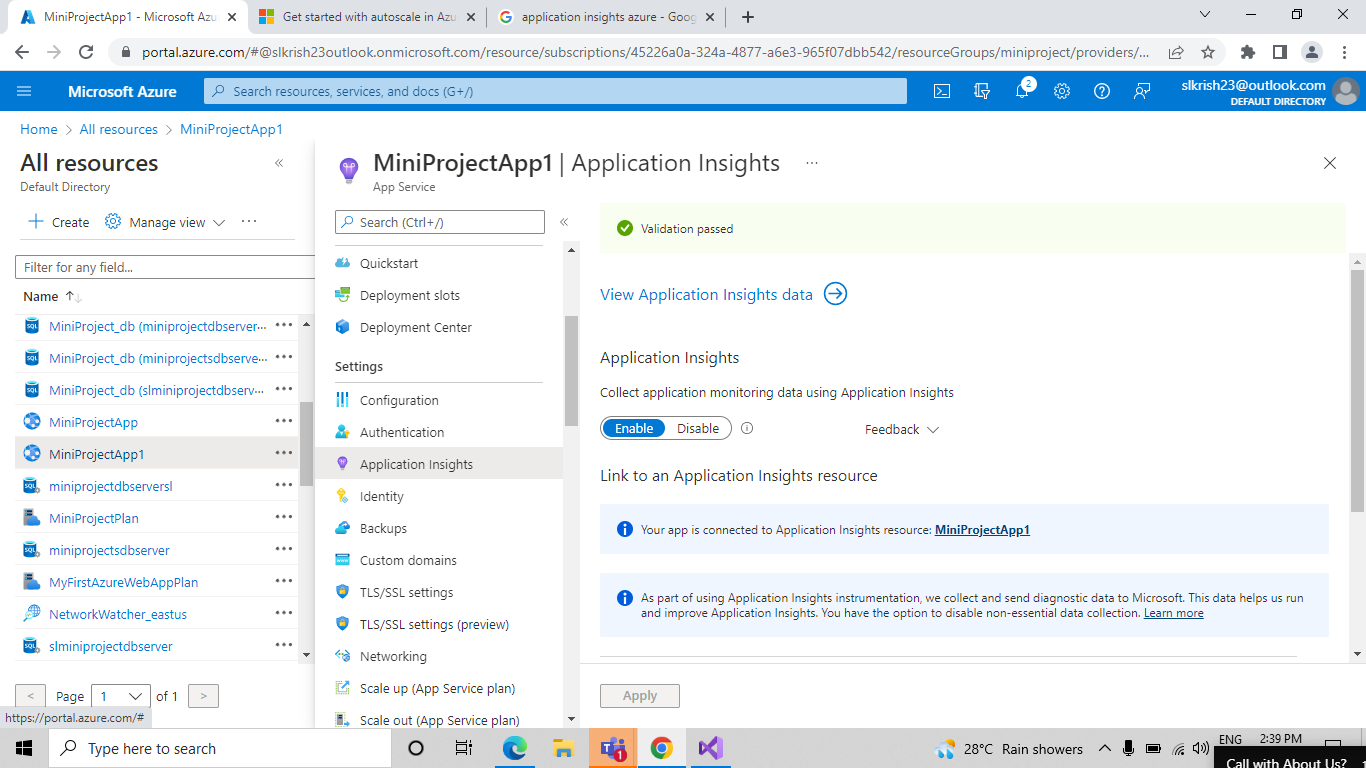


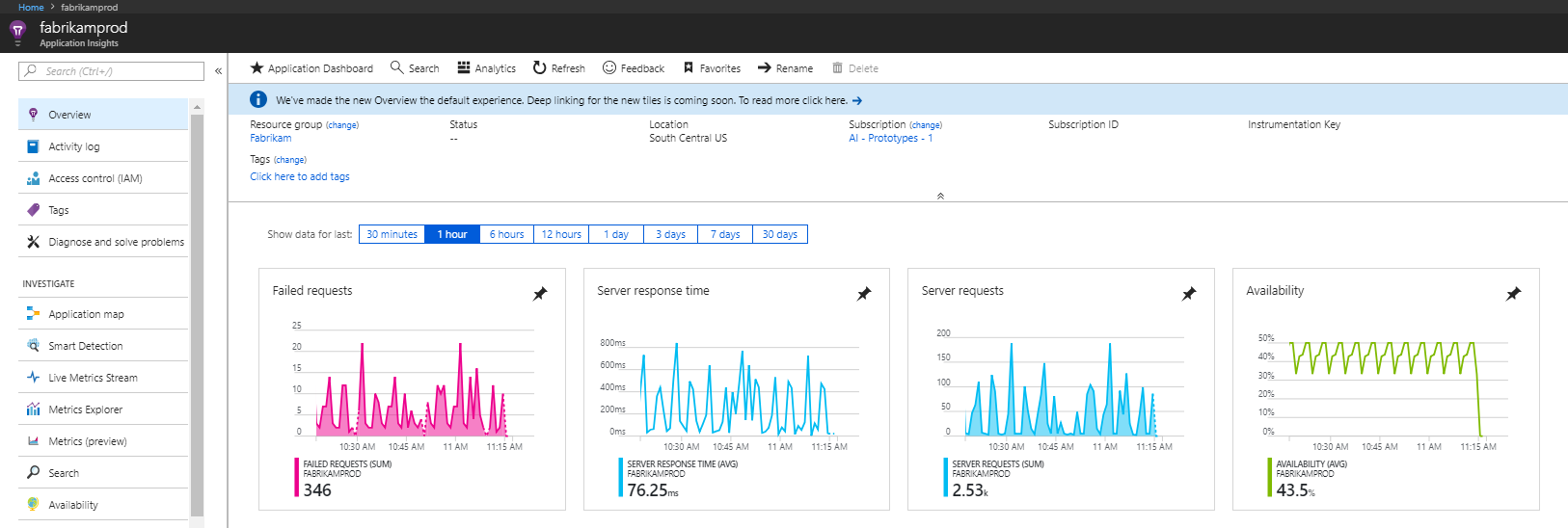
**Deployment Slots** **-** Azure Functions deployment slots allow your function app to run different instances called "slots". Slots are different environments exposed via a publicly available endpoint. One app instance is always mapped to the production slot, and you can swap instances assigned to a 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.

The following reflect how functions are affected by swapping slots:

* Traffic redirection is seamless; no requests are dropped because of a swap. This seamless behavior is a result of the next function triggers being routed to the swapped slot.
* Currently executing function are terminated during the swap. Please review [Improve the performance and reliability of Azure Functions](https://docs.microsoft.com/en-us/azure/azure-functions/performance-reliability#write-functions-to-be-stateless) to learn how to write stateless and defensive functions.

**Configuration of Application Insights :-**





**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:

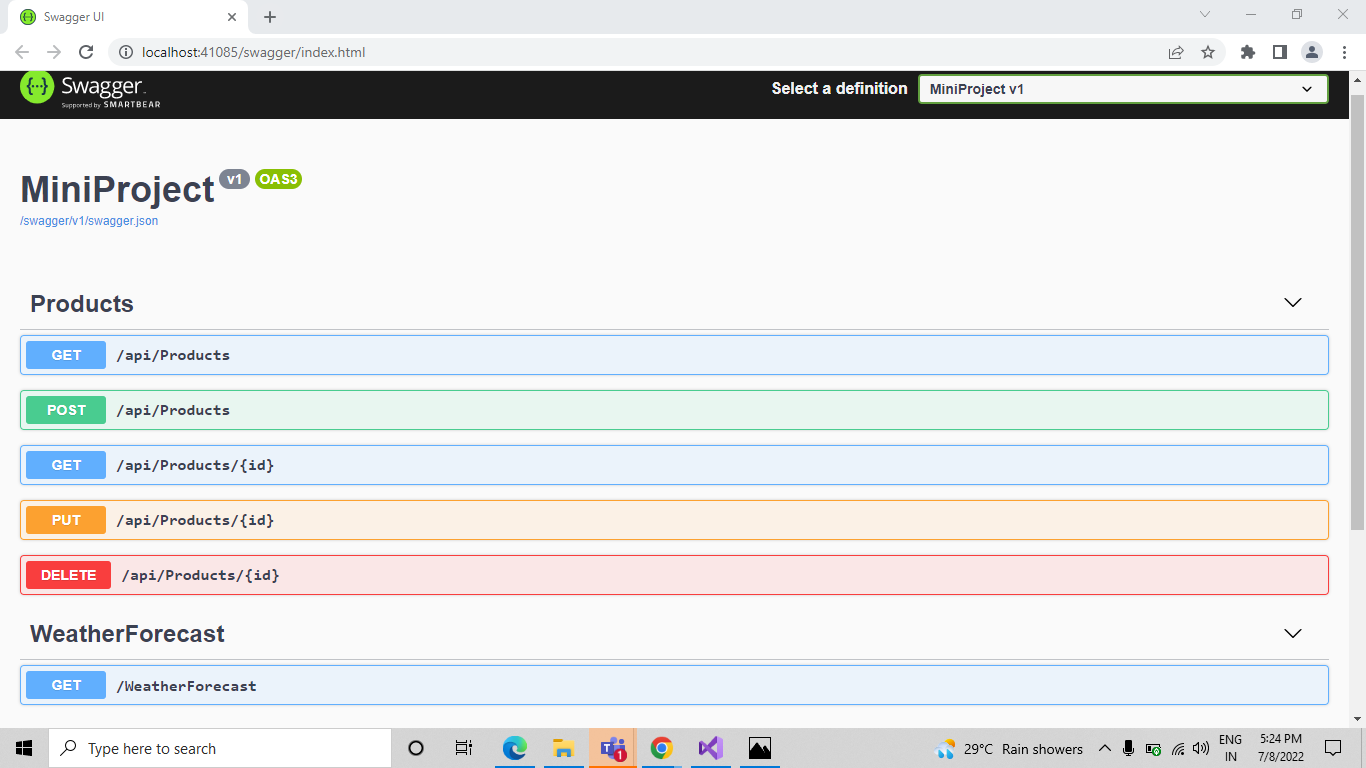
* Supports a wide variety of platforms, including .NET, Node.js, Java, and Python.
* Works for apps hosted on-premises, hybrid, or on any public cloud.
* Integrates with DevOps processes.
* Has connection points to many development tools.
* Can monitor and analyze telemetry from mobile apps by integrating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

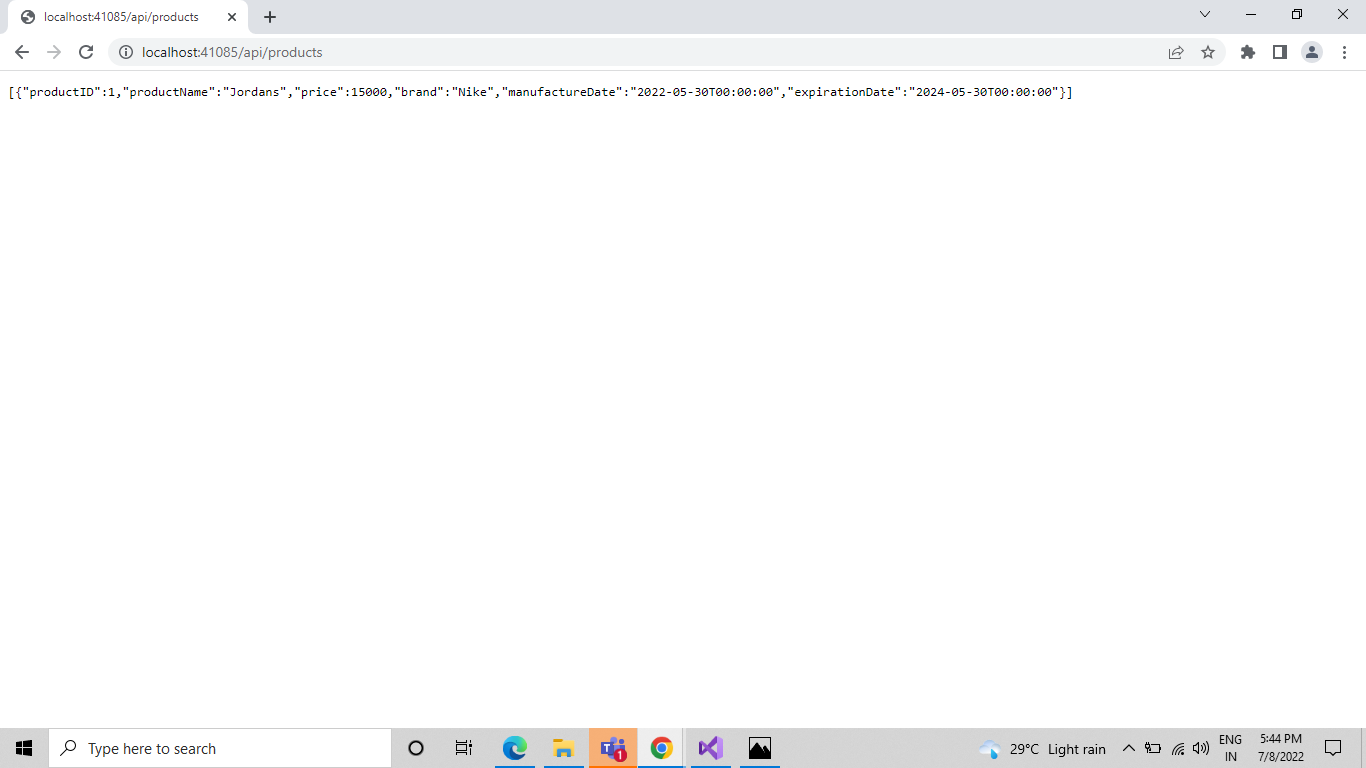
**Configuration Swagger for API :-**

**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. Why is it so great? Well, 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.](https://github.com/OAI/OpenAPI-Specification/blob/master/versions/2.0.md)

There are a few ways in which Swagger can help drive your API development further:

* Design-first users: use [Swagger Codegen](https://swagger.io/swagger-codegen/) togenerate a server stub for your API. The only thing left is to implement the server logic – and your API is ready to go live!
* Use [Swagger Codegen](https://swagger.io/swagger-codegen/) to generate client libraries for your API in over 40 languages.
* Use [Swagger UI](https://swagger.io/swagger-ui/) to generate interactive API documentation that lets your users try out the API calls directly in the browser.
* Use the spec to connect API-related tools to your API. For example, import the spec to [SoapUI](https://soapui.org/) to create automated tests for your API.



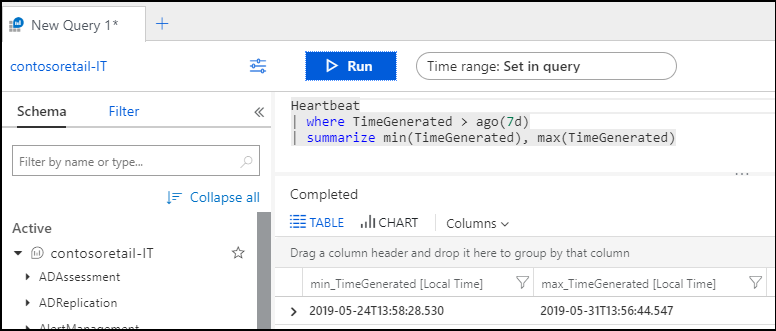


# **Log Analytics :-**

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.

## **Query scope :-**

The query scope defines the records that are evaluated by the query. This will usually include all records in a single Log Analytics workspace or Application Insights application. Log Analytics also allows you to set a scope for a particular monitored Azure resource. This allows a resource owner to focus only on their data, even if that resource writes to multiple workspaces.



## **Time range :-**

The time range specifies the set of records that are evaluated for the query based on when the record was created. This is defined by the TimeGenerated column on every record in the workspace or application as specified in the following table. For a classic Application Insights application, the timestamp column is used for the time range.

Set the time range by selecting it from the time picker at the top of the Log Analytics window. You can select a predefined period or select Custom to specify a specific time range.

