Serverless Architecture in Cloud-based Applications

*1 What is this article about?*

The idea of this article is to give you a very understanding about Serverless architecture by using one of the top 3 cloud provider Azure. At the end of this article, you will be able to understand about the serverless architecture with the help of Azure function app.

*2 What is a serverless architecture?*

A serverless architecture is use to build and run applications and services without manage infrastructure. Our application still runs on servers, but all the server management is done by Azure/AWS/GCP cloud providers. we no longer provision like scale up or scale down, and maintain servers to run your applications, databases, and storage systems.

**2.1 Example services across different cloud providers**

* Azure Function App in Azure
* AWS Lambda in AWS
* [Google Cloud Functions](https://www.techtarget.com/searchcloudcomputing/definition/Google-Cloud-Functions) in GCP

*3 Why use serverless architectures?*

As developers focus on core product/project development instead of worrying about managing infrastructure etc.,

So, serverless architecture is a way to build your [cloud-based application](https://relevant.software/blog/modern-cloud-based-application-development/) without managing infrastructure. Still, serverless does not mean there are *no servers* at all. Servers are simply eliminated from the app development since they are managed automatically whenever it required.

*4 How Azure Functions works*

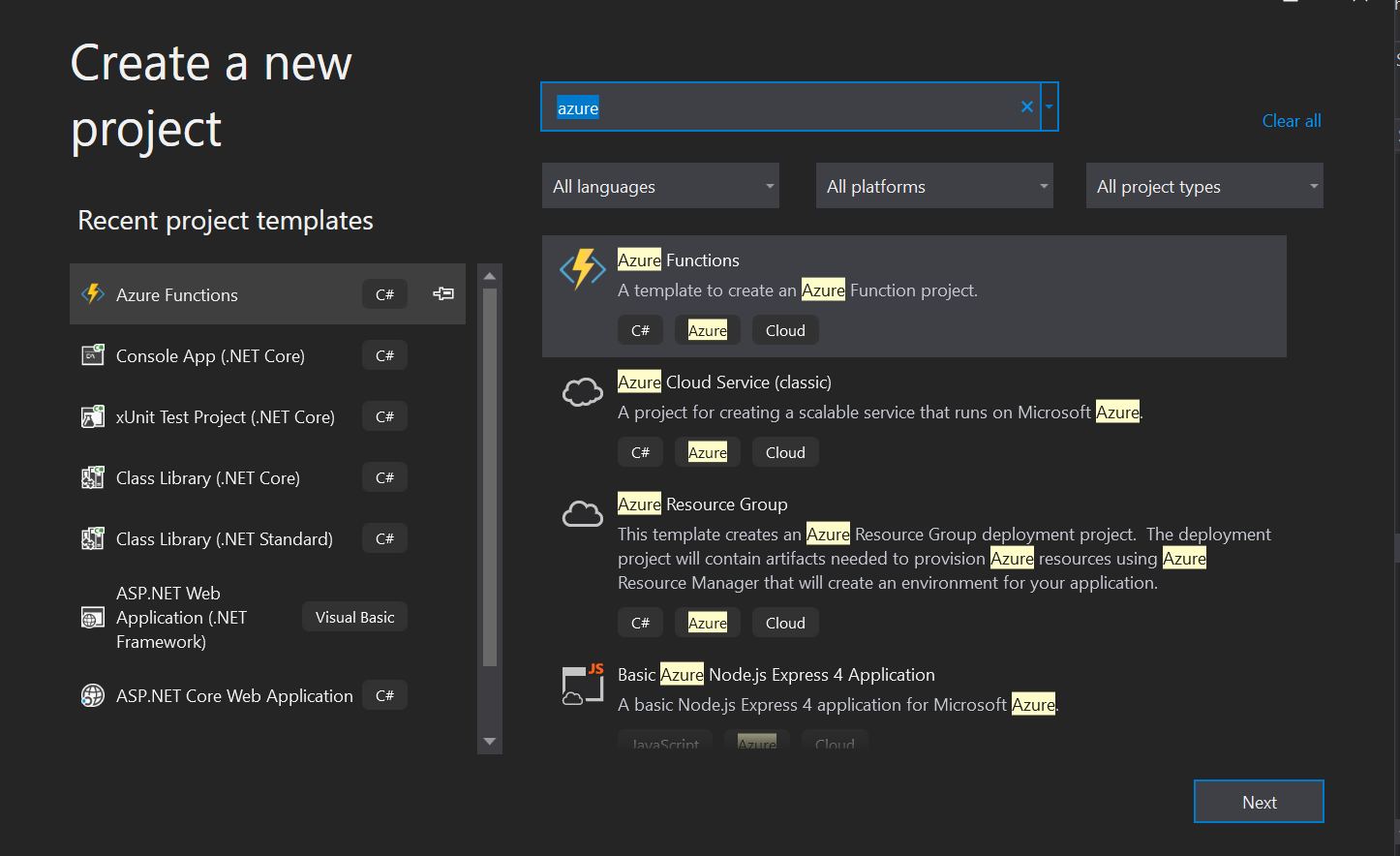
The Azure Functions serverless is to build event-driven apps that run code whenever triggered by azure resources events/Http/timer. The platform automatically manages all the computing resources required in those processes.

*5 Create Azure Functions with different methods*

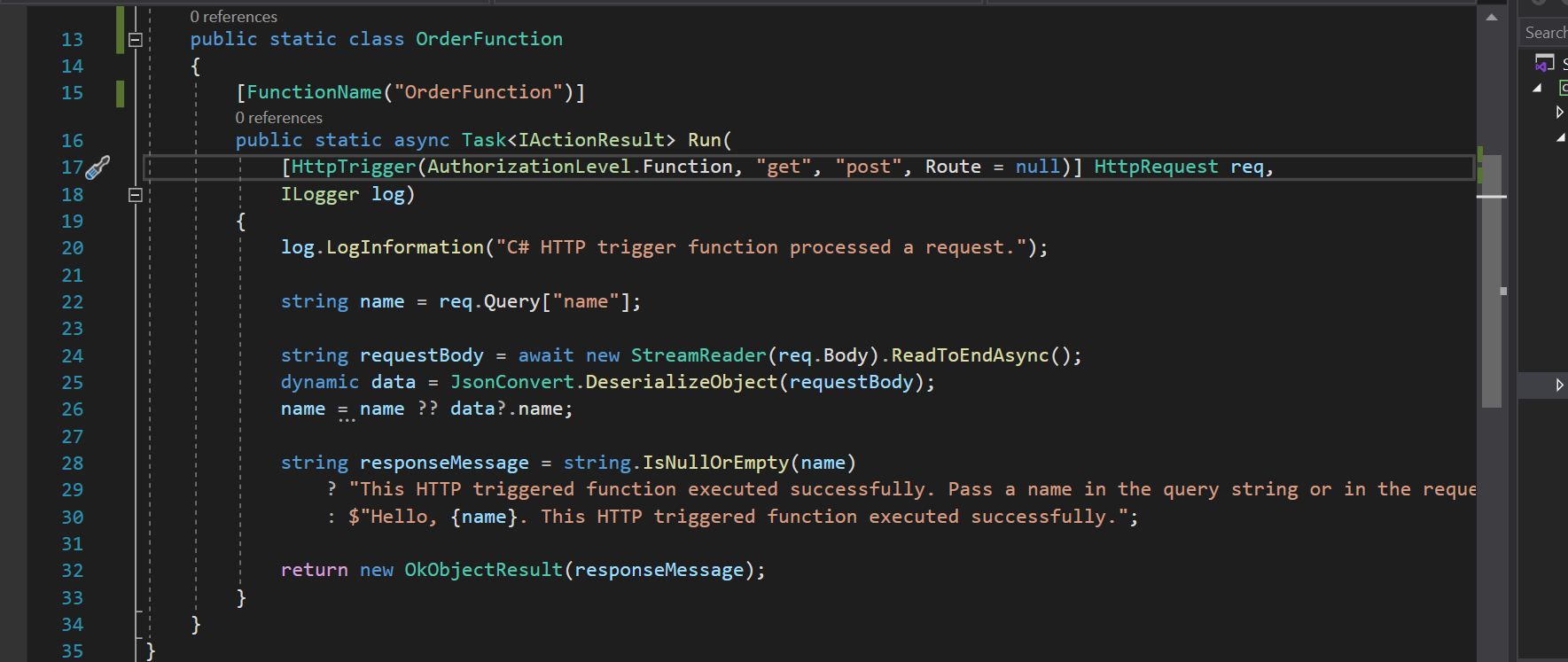
let's see how to create function app with different ways in the below sections.

*5.1 Visual studio*

* Create new project
* Search Azure function and create a new project with required information.
* Write our logic inside the function app Run/function attribute method (Screenshot 10.2).
* Right click on the function app project and choose publish option to Azure and then we need to provide the valid subscription and then storage account details (Storage account is mandatory when we create or deploy function app)
* Once done everything it will deploy into azure



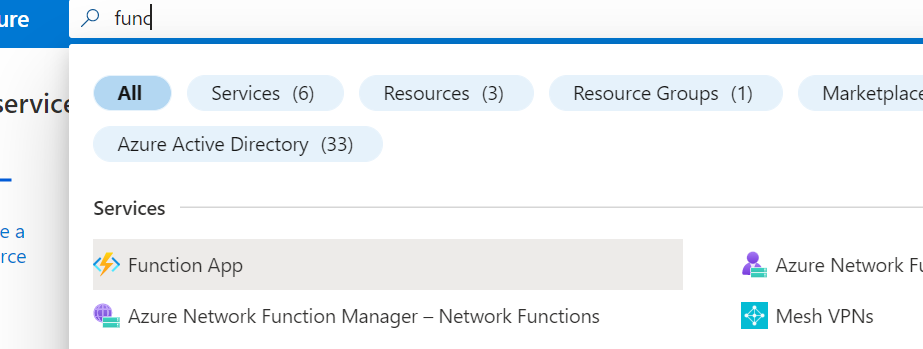
Screenshot 5.1.1



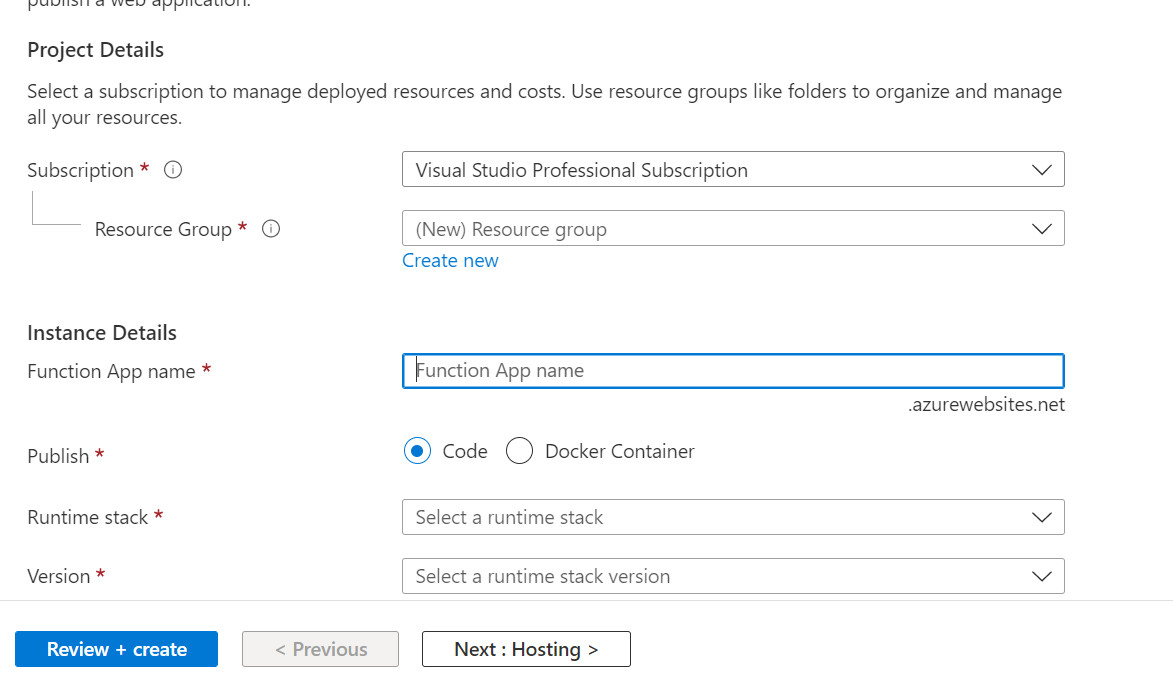
Screenshot 5.1.2

*5.2 Azure portal:*

* Search function app in the portal and we need to click function app in searched item

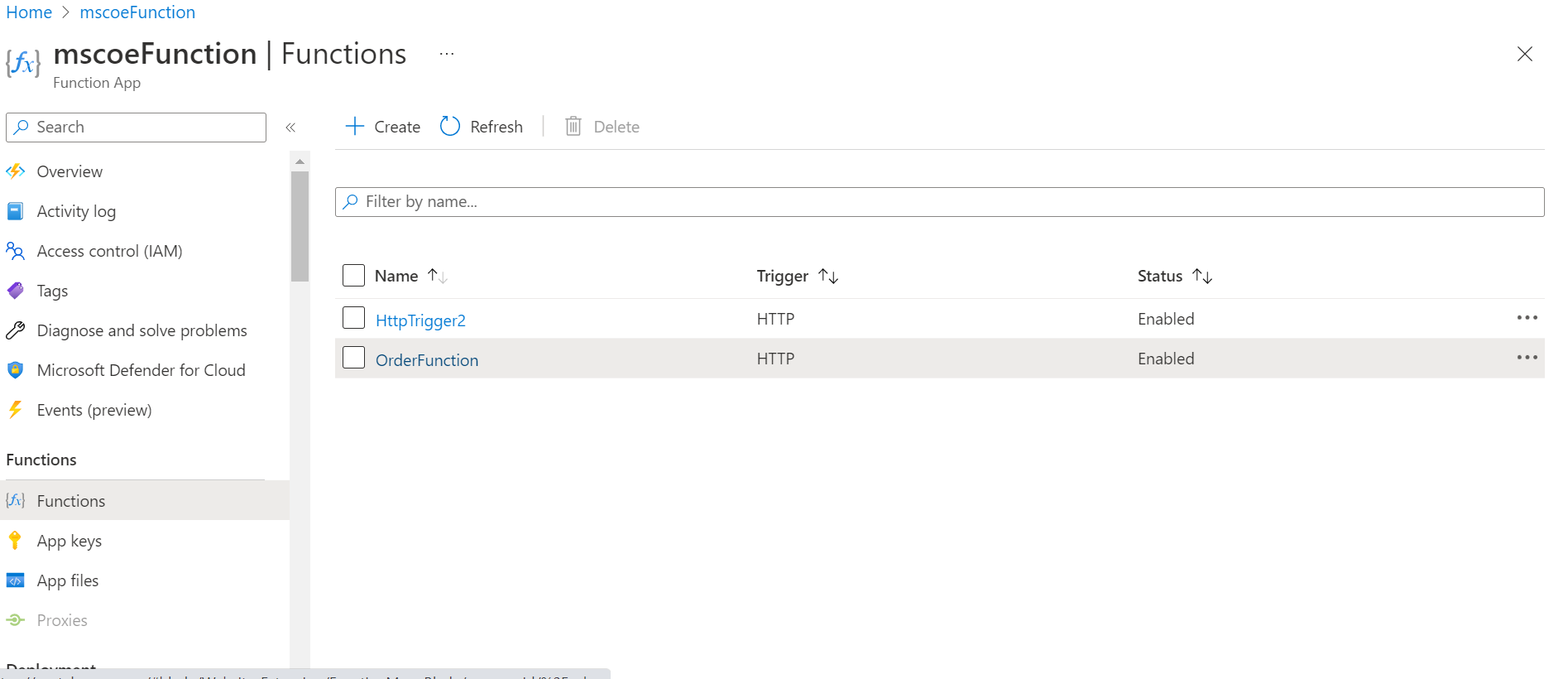


Screenshot 5.2.1



Screenshot 5.2.2

* We need to provide the valid subscription Function app name, Resource group and then then storage account details (Storage account is mandatory when we create or deploy function app) etc.,
* Finally review and create it.
* Write our logic inside the function app Run/function attribute method (Screenshot 10.2).
* Once we can create many functions inside the function app with different triggers



*5.3 Provisioning or managing Azure Functions Using Terraform (IAC)*

We need to make sure the files main.tf, variables.tf, and env.tfvars are together in a subfolder(terraform) of our project. To run this code, terraform needs to be installed and it will be available in the command line.

Code snippets available in section 5.3.2. Additionally Azure CLI also installed locally. In this section we just create or update the azure function app we are not doing scaling up and scaling out. It can be handled by itself (on demand basis).

Terraform is cloud-agnostic. One config file can manage multiple providers (AWS, GCP, Azure) and handle cross-cloud dependencies.

For e.g.:

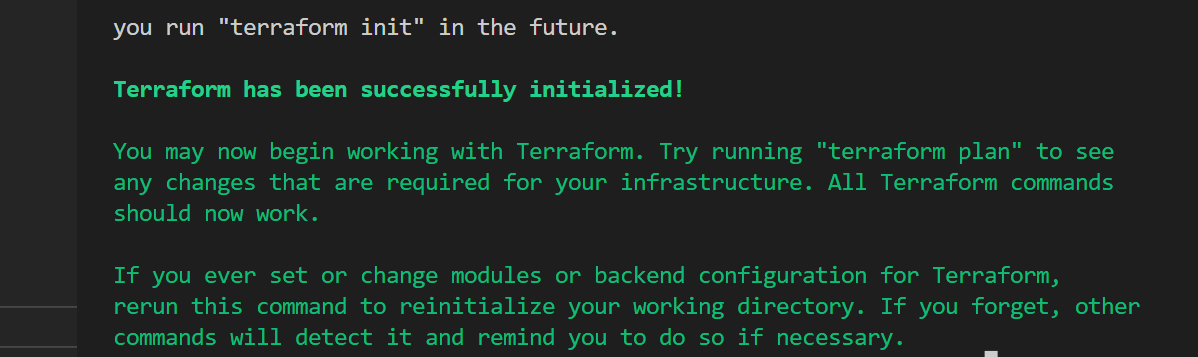
Cloud formation supports only AWS

ARM templates support only Azure

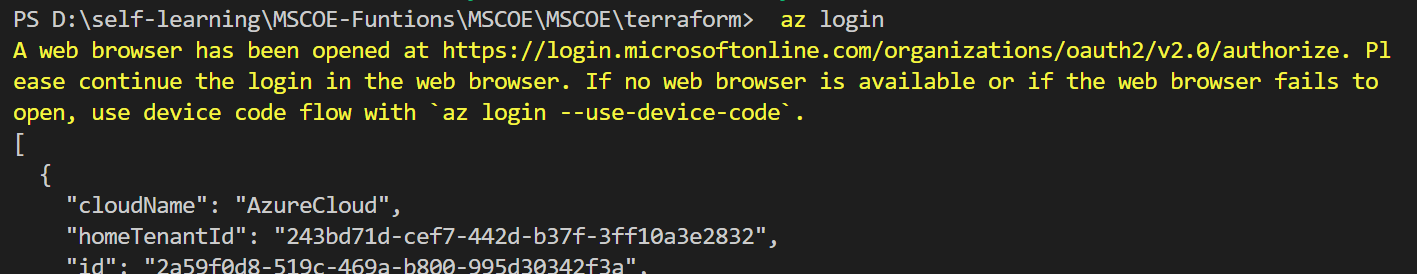
*5.3.1 Commands*

Step 1: Terraform init command initialize our working directory.



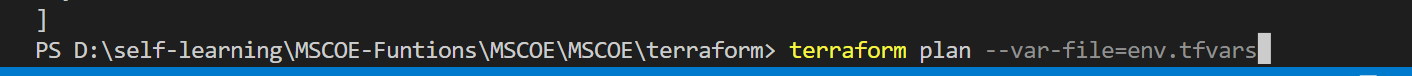


Step 2: Uses the following command for authorize our azure subscription through Azure command line interface(In order to run this command we need to install Azure CLI in our local).



Step 3: Uses the following command for creates the terraform plan based on files main.tf, variables.tf, and env.tfvars

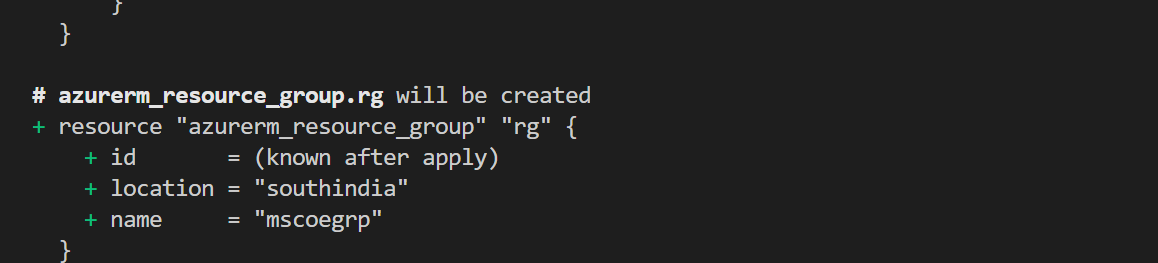
If any issues occurs/issues in terraform files it will indicate the errors.



Once plan command executed it will show the below screenshots with + or – symbols

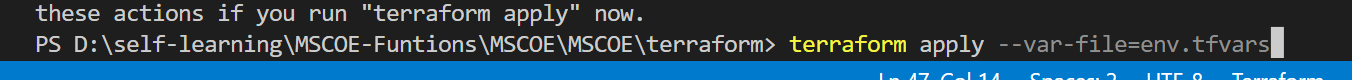
+ => Symbol indicates adding new resource

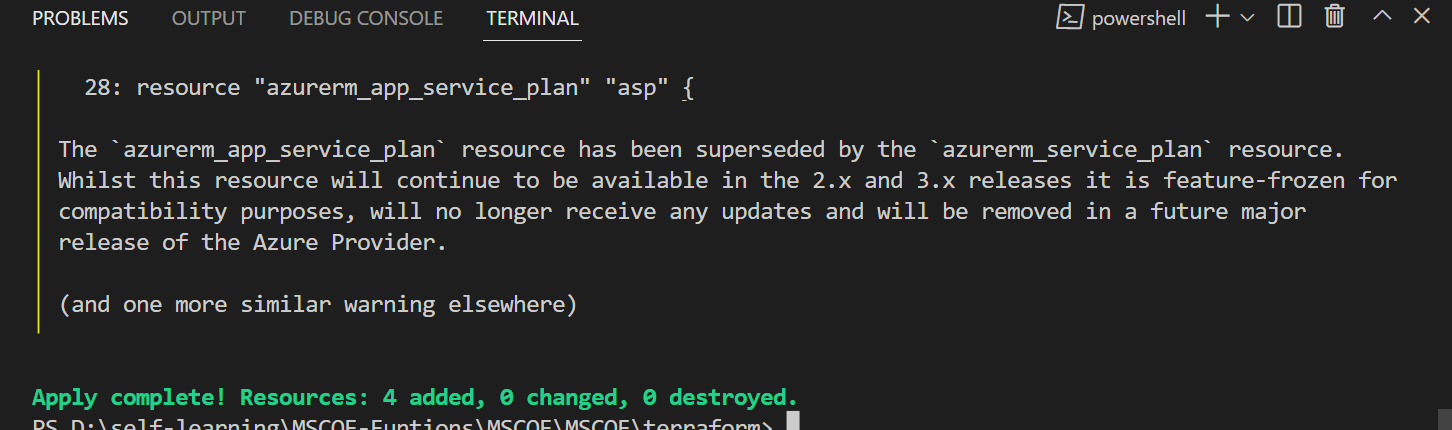
- => Symbol indicates removing/updating existing resource



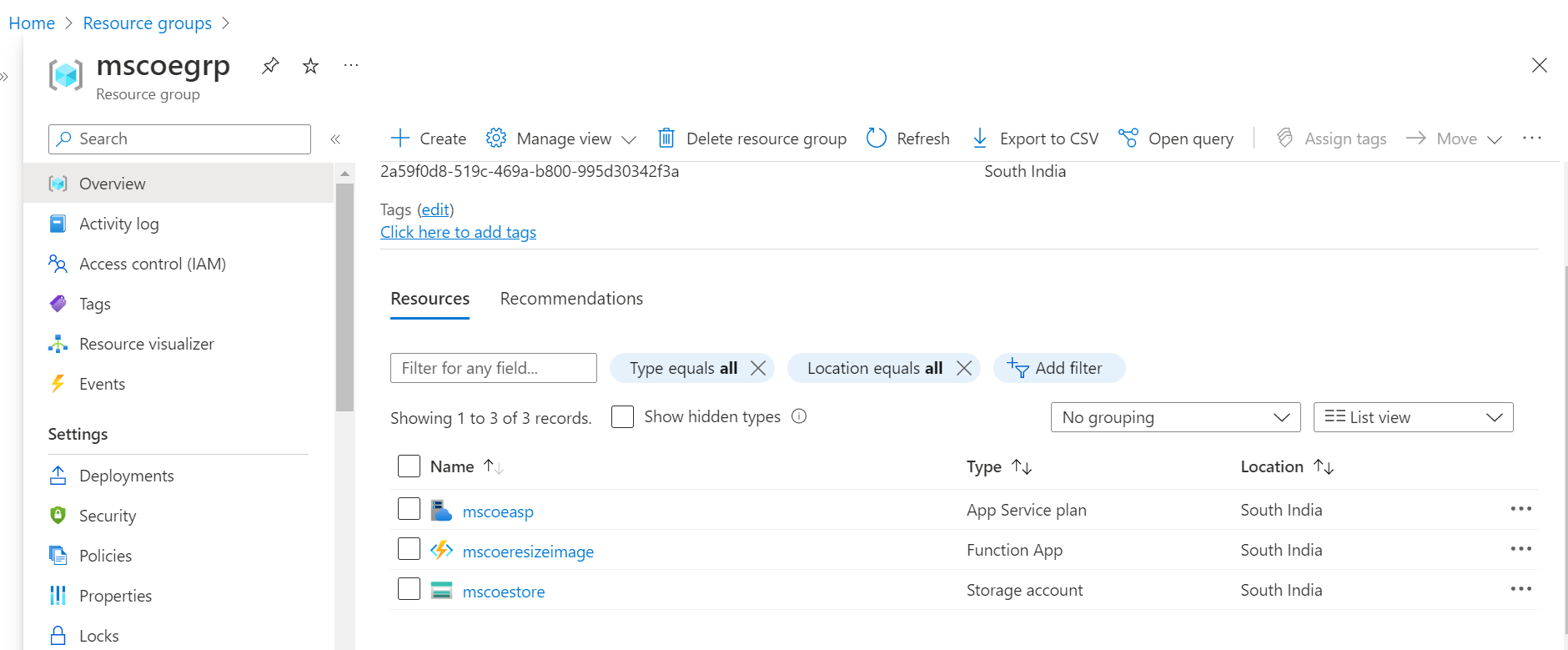


Step 4: Once terraform plan is completed and apply command used for creating and updating/deleting resources in Azure (or other cloud providers)





The above screenshot mentions apply command created/updated resources successfully.



The above screenshots show 4 resources created successfully in Azure (Resource group, App service plan, functions and storage account)

*5.3.2 Code snippet:*

Before proceeding our terraform commands we need the below files with code snippets. This contains how to create function app resource along with other required resources like storage account and etc.,

*main.tf*

terraform {

  required\_providers {

    azurerm = {

      source  = "hashicorp/azurerm"

      version = ">= 2.26"

    }

  }

}

provider "azurerm" {

  features {}

}

resource "azurerm\_resource\_group" "rg" {

  name     = var.resource\_group\_name

  location = var.location

}

resource "azurerm\_storage\_account" "sa" {

  name                      = var.storage\_account\_name

  resource\_group\_name       = azurerm\_resource\_group.rg.name

  location                  = azurerm\_resource\_group.rg.location

  account\_tier              = "Standard"

  account\_replication\_type  = "LRS"

  account\_kind              = "StorageV2"

}

resource "azurerm\_app\_service\_plan" "asp" {

  name                = var.app\_service\_plan\_name

  resource\_group\_name = azurerm\_resource\_group.rg.name

  location            = azurerm\_resource\_group.rg.location

  kind                = "functionapp"

  sku {

    size = "Y1"

    tier = "Dynamic"

  }

}

resource "azurerm\_function\_app" "function" {

  name                       = var.function\_name

  resource\_group\_name        = azurerm\_resource\_group.rg.name

  location                   = azurerm\_resource\_group.rg.location

  app\_service\_plan\_id        = azurerm\_app\_service\_plan.asp.id

  storage\_account\_name       = azurerm\_storage\_account.sa.name

  storage\_account\_access\_key = azurerm\_storage\_account.sa.primary\_access\_key

  https\_only                 = true

  version                    = "~3"

  app\_settings = {

    "FUNCTIONS\_WORKER\_RUNTIME" = "dotnet"

  }

}

*env.tfvars*

location              = "southindia"

resource\_group\_name   = "mscoegrp"

storage\_account\_name  = "mscoestore"

app\_service\_plan\_name = "mscoeasp"

function\_name         = "mscoeresizeimage"

*variable.tf*

variable "location" {

  description = "Name of the location where the resources will be provisioned"

  type = string

}

variable "resource\_group\_name" {

  description = "Name of the resource group"

  type = string

}

variable "storage\_account\_name" {

  description = "Name of the storage account"

  type = string

}

variable "app\_service\_plan\_name" {

  description = "Name of the application service plan"

  type = string

}

variable "function\_name" {

  description = "Name of the function"

  type = string

}

*6 Host.json settings reference for Azure Functions*

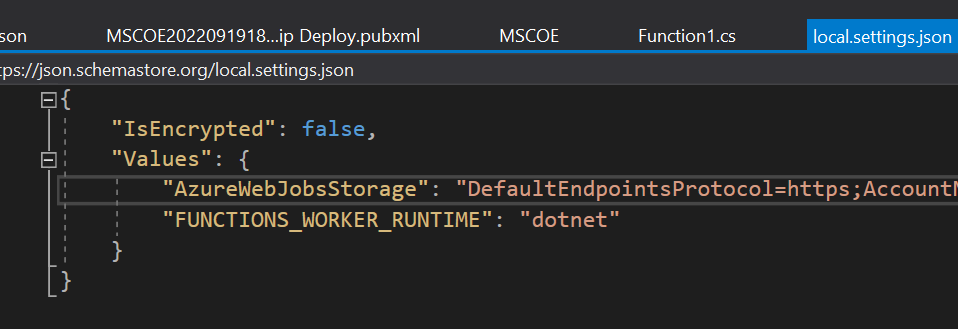
The host.json file contains configuration options it affects all functions in a function app instance.  host.json file contains logger/appinsights related information.



Screenshot 6.1

*7 Local settings reference for Azure Functions*

Local settings in a function app contain configuration options that it all functions for that function app. When we run locally, these configurations will be useful.



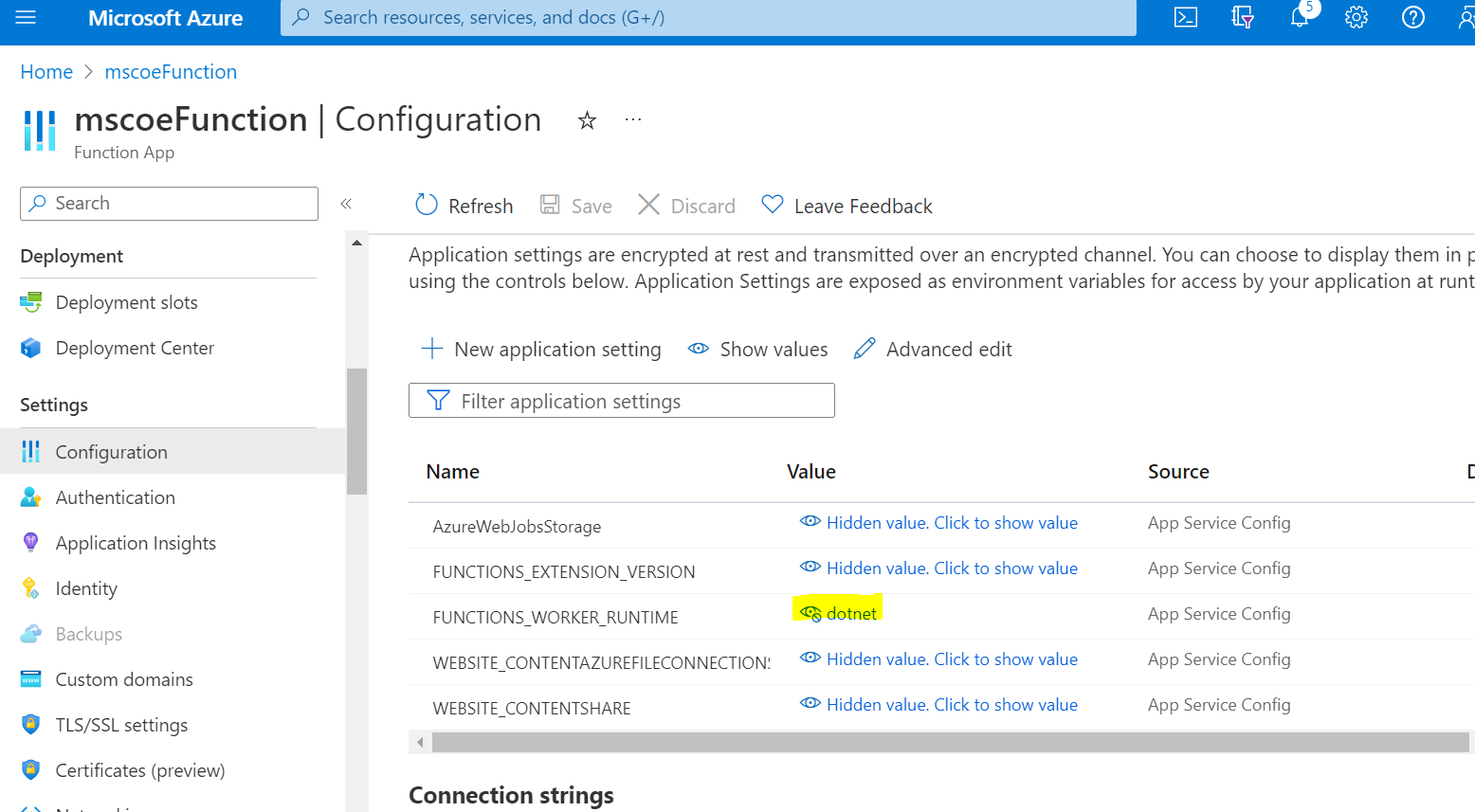
Screenshot 7.1

*8 App settings reference for Azure Functions*

Application settings in a function app contain configuration blade/section and it affect all functions in particular function app. When we run locally, these settings are accessed as local [environment variables](https://learn.microsoft.com/en-us/azure/azure-functions/functions-develop-local#local-settings-file)/local.settings.json.

There are several ways that you can add, update, and delete function app settings in Azure function configuration:

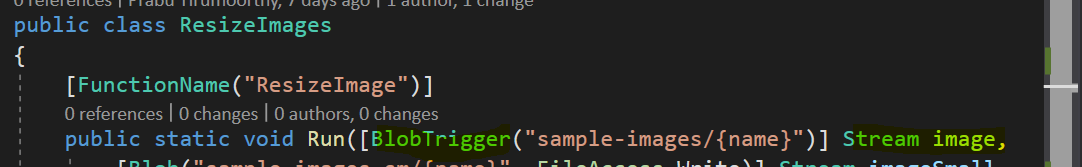
* [*In the Azure portal.*](https://learn.microsoft.com/en-us/azure/azure-functions/functions-how-to-use-azure-function-app-settings#settings)
* [*By using the Azure CLI.*](https://learn.microsoft.com/en-us/cli/azure/functionapp/config/appsettings#az-functionapp-config-appsettings-set)
* [*By using Azure PowerShell.*](https://learn.microsoft.com/en-us/powershell/module/az.functions/update-azfunctionappsetting)



Screenshot 8.1

*9 Azure Functions triggers*

A trigger used to call a function and a function must have exactly one trigger. Triggers have associated data, which is often provided as the payload of the function.



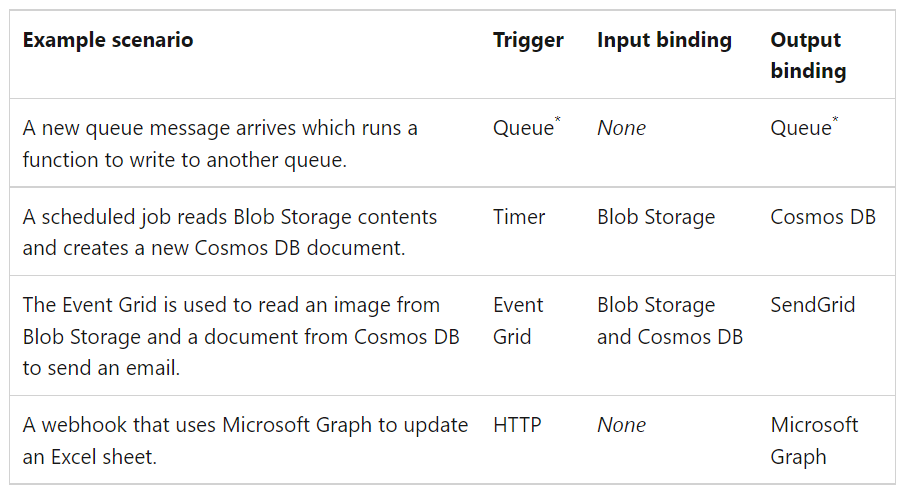
Screenshot 9.1

*10 Bindings (Input and Output)*

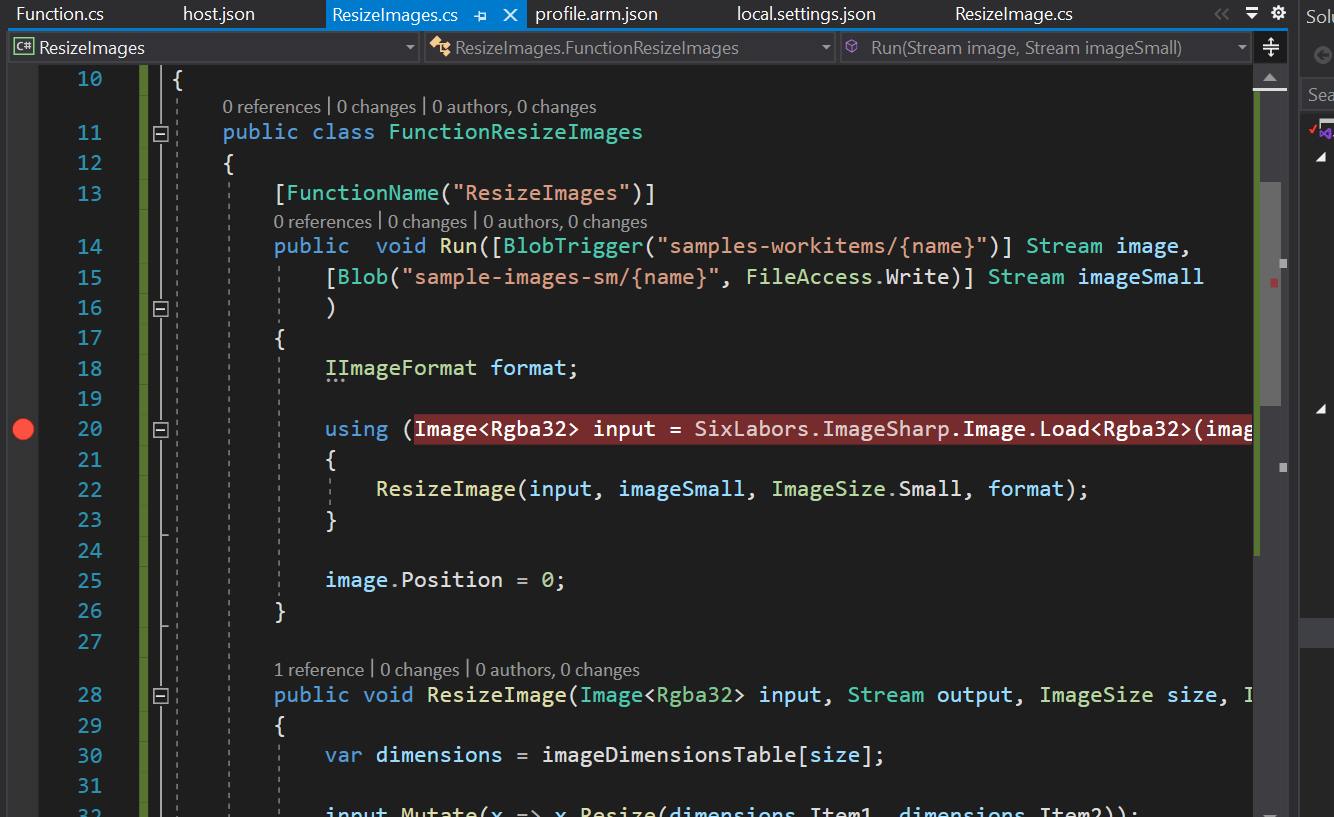
Data coming from or to a function is called Bindings There are two types of bindings *input bindings*, *output bindings*.

Bindings are optional and a function might have one or multiple input and/or output bindings.

There are many types of triggers. Few of the below mentioned triggers and its use cases



Screenshot 10.1

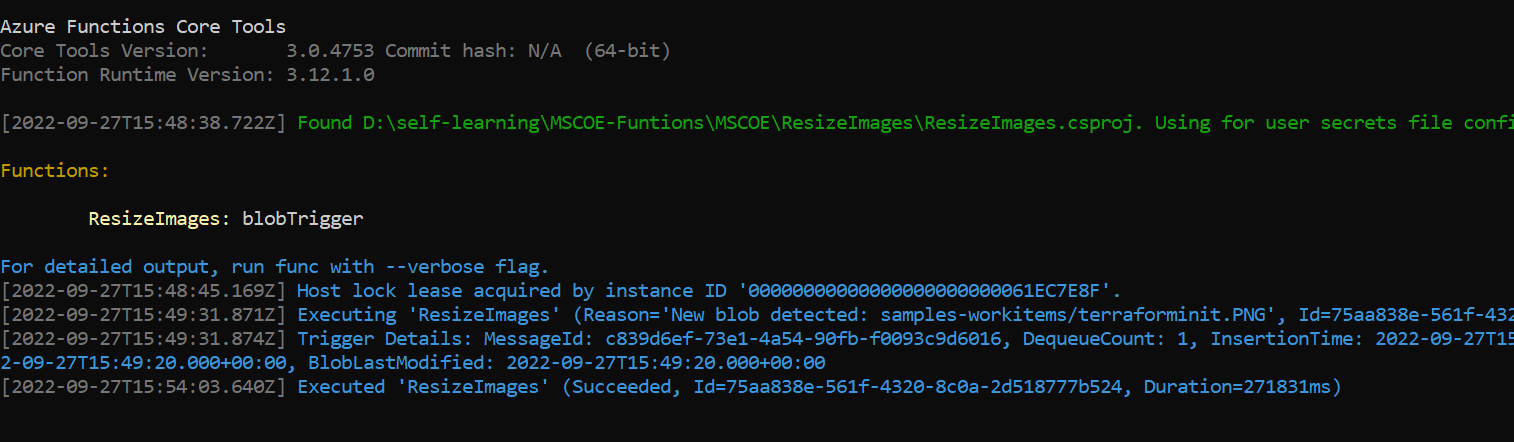


Screenshot 10.2

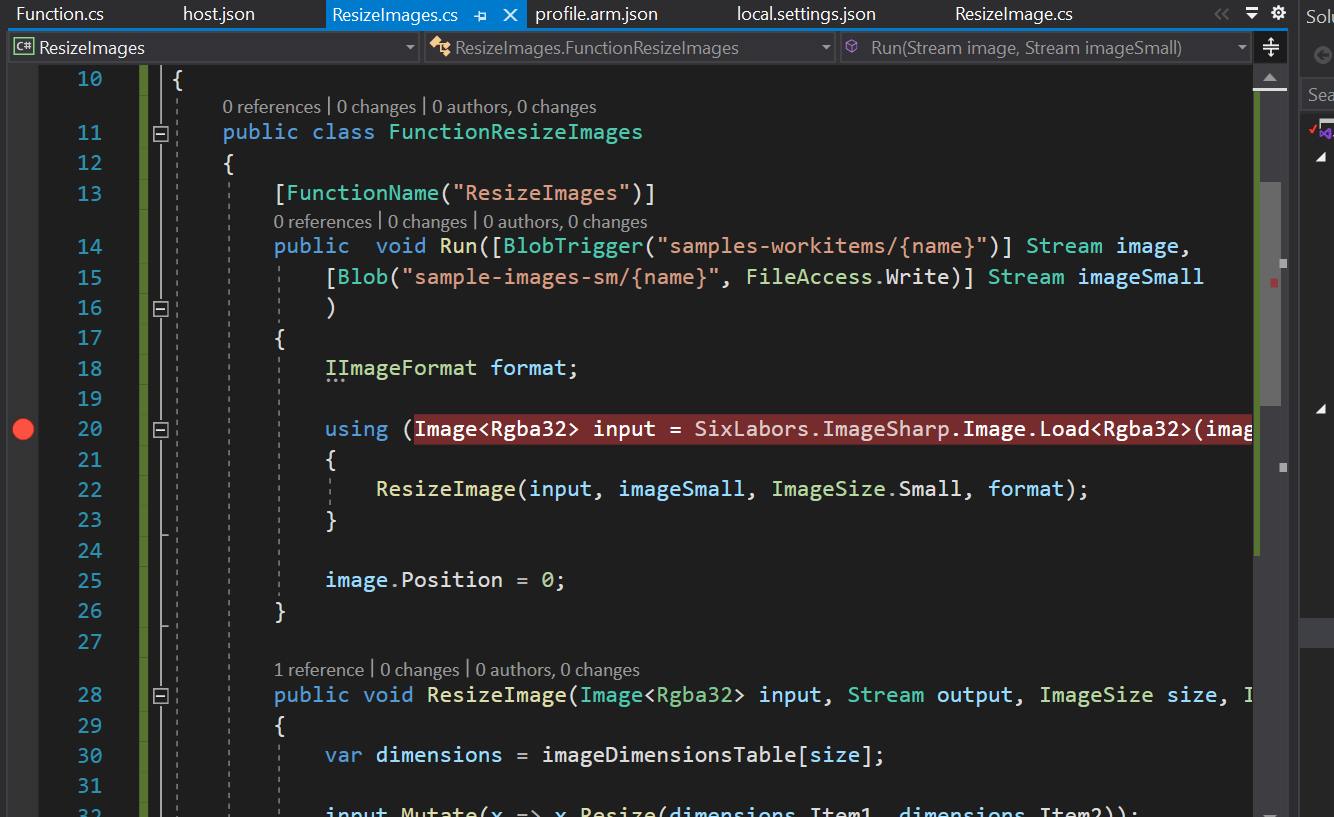
*11 Run Resize Image Azure function:*

In screenshot 10.2 “sample-workitems” is container of storage account “mscoestore”.

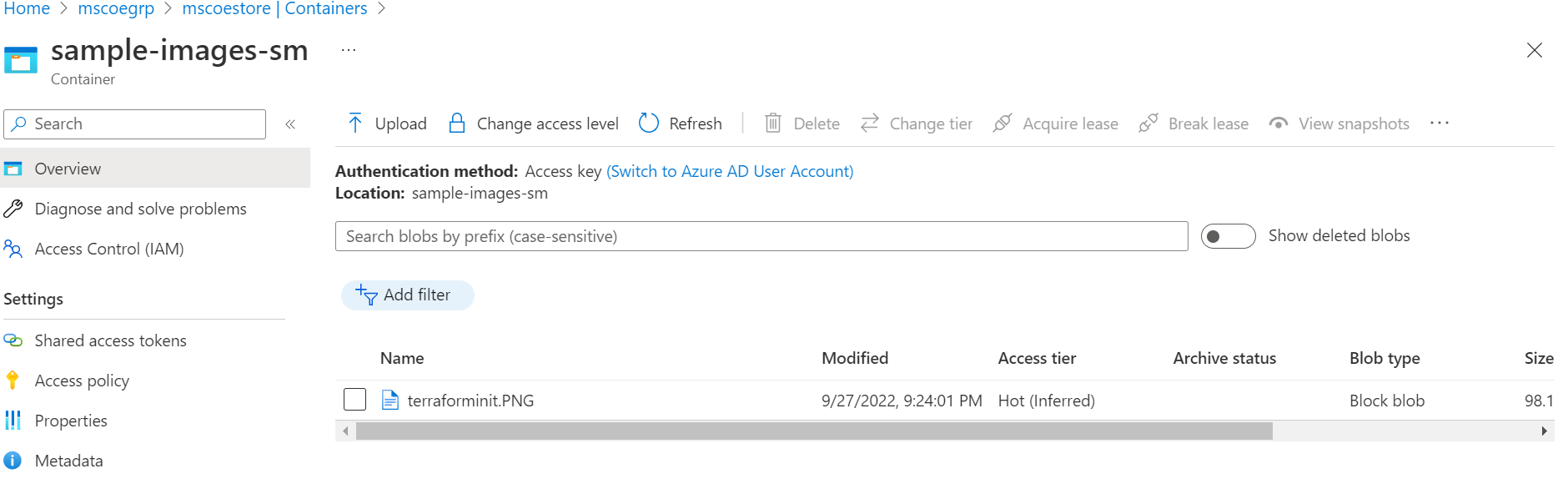
Whenever I upload image file to “sample-workitems” container, It will resize the image and then upload into “sample-images-sm” container of storage account “mscoestore”.



Screenshot 11.1



Screenshot 11.2



Screenshot 11.3

*12 Use cases of Azure function:*

* Reminders and notifications
* Scheduled /Background tasks
* File processing
* Event driven architecture asynchronous calls.
* Message processing (Storage queues/Service bus)
* Azure Functions are NOT a replacement for Web APIs. Web APIs usually lots of business logic in it. Functions usually receives input and processing and send as output. It will run on-demand.

*13 Benefits of Azure function:*

* Pay-as-You-Use (Pay based on our usage) and Cost-Efficient
* Azure functions are lightweight and serverless
* Supports Different Programming Languages like Java, C#, JavaScript, Python, TypeScript, and even PowerShell.
* Azure functions can easily communicate with other APIs, library functions, and databases.
* Trigger based execution – function will execute based on other service triggers like Cosmos DB trigger, Queue trigger, EventHub trigger. It runs on demand because it is a trigger-based service

## *13. Conclusion*

With Azure function blog, how Azure functions and what benefits it can bring to the users/customers/developers. This is quite a different from standard application development because we need to do extensive IT infrastructure that is sometimes expensive, and time delay. We developed or covered stateless functions. But Azure provides us additional option stateful function concept called Durable functions. I will cover it in future blogs. But I hope this blog will help us that when we go for function, benefits and use cases.