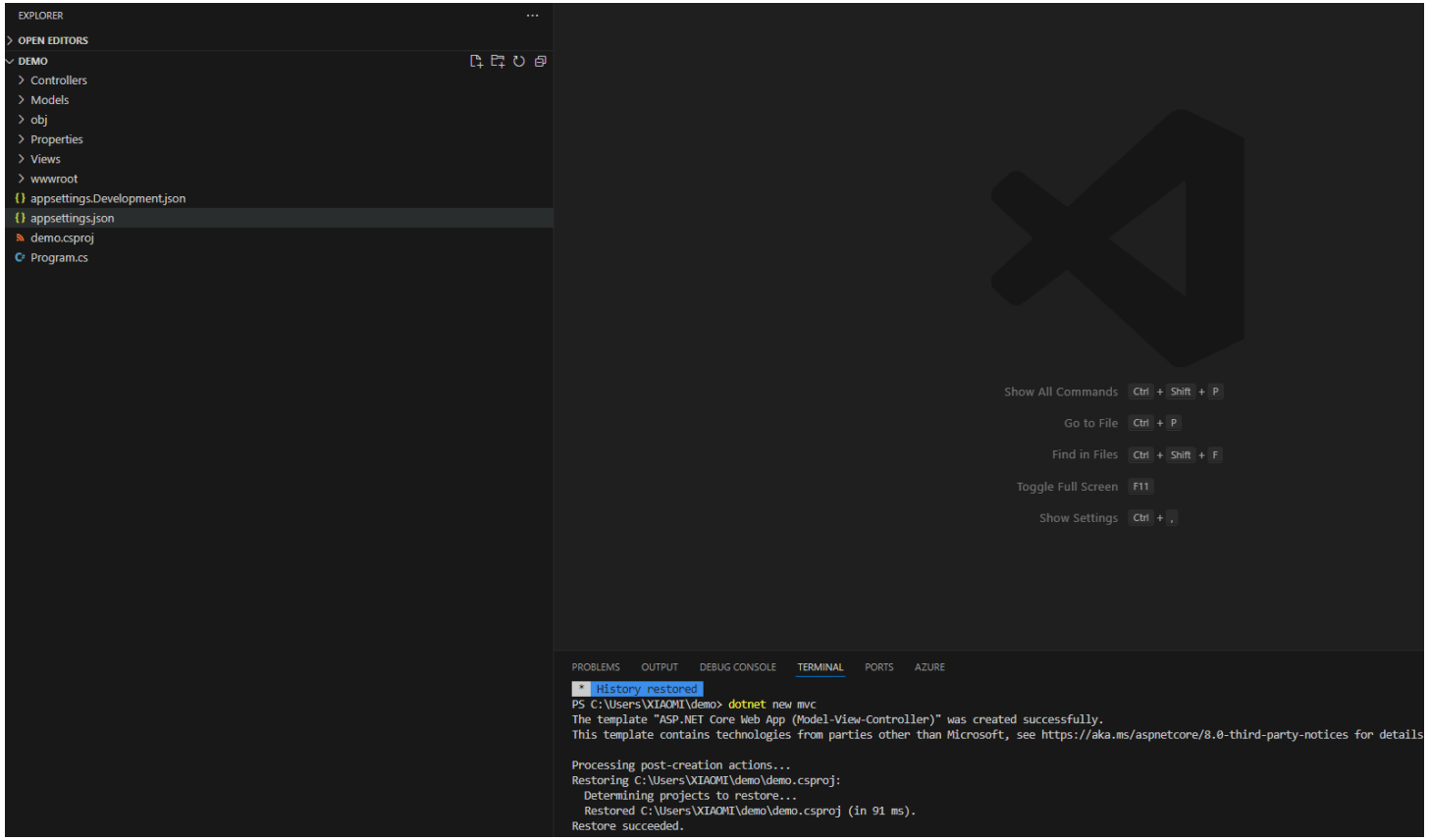
Лекція 8

dotnet new mvc

Команда dotnet new mvc використовується для створення нового проекту на базі шаблону ASP.NET Core MVC (Model-View-Controller). Це стандартна архітектурна модель для веб-прикладень в ASP.NET Core, яка розділяє логіку додатків на три основні частини:
Модель — відповідає за дані і бізнес-логіку.
View — відповідає за інтерфейс користувача та представлення даних.
Controller — відповідає за зв’язок між Model і View, а також за обробку запитів користувачів.

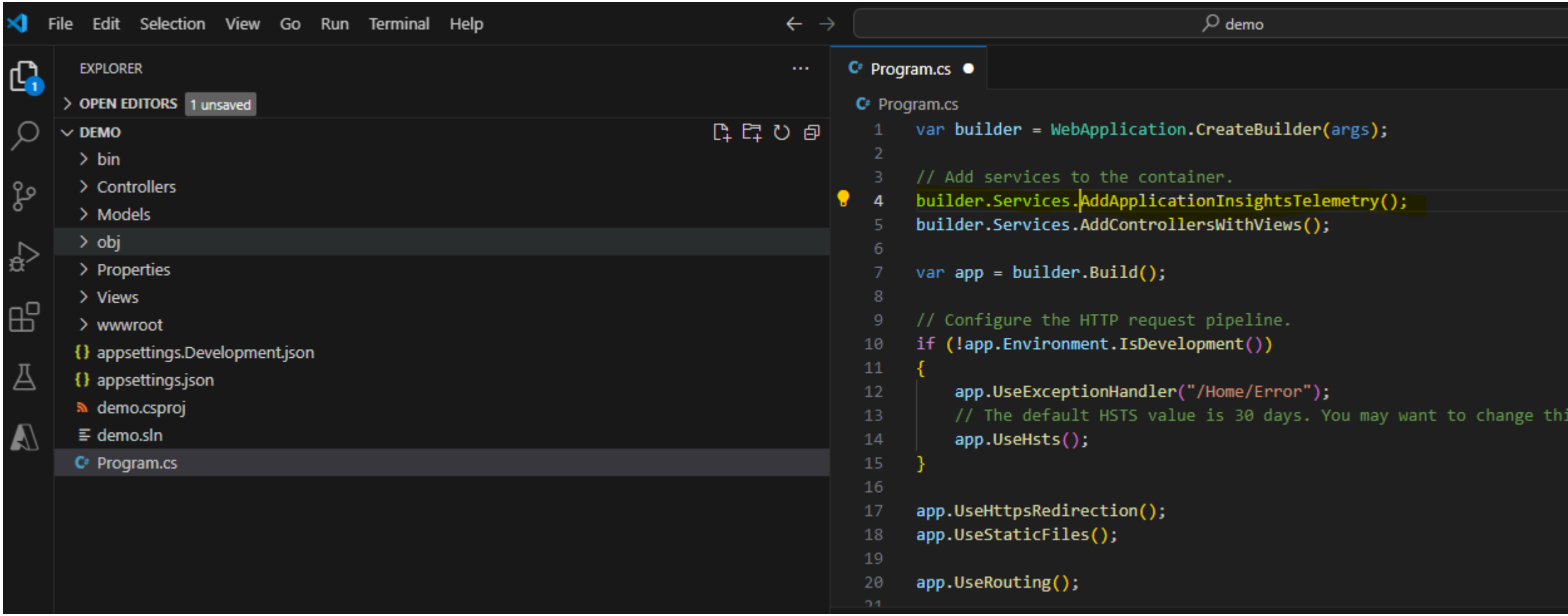


dotnet add package Microsoft.ApplicationInsights.AspNetCore

Microsoft.ApplicationInsights.AspNetCore використовується для додавання в проект пакета Application Insights, який надає можливості для моніторингу та телеметрії додатків ASP.NET Core.
Задачі пакету Microsoft.ApplicationInsights.AspNetCore:
Збір телеметрії: Пакет збирає дані про роботу вашого додатка, такі як виключення, запити, метрики продуктивності, залежності, журнал і користувацькі події.
Моніторинг продуктивності: Допомагає контролювати швидкість роботи, час виконання запитів та інші показники продуктивності.
Відслежування помилок і файлів: Автоматично реєструє непередбачені виключення та файли в робочих додатках.
Інтеграція з Azure: дозволяє надсилати зібрані дані на платформу Application Insights в Azure для подальшого аналізу.

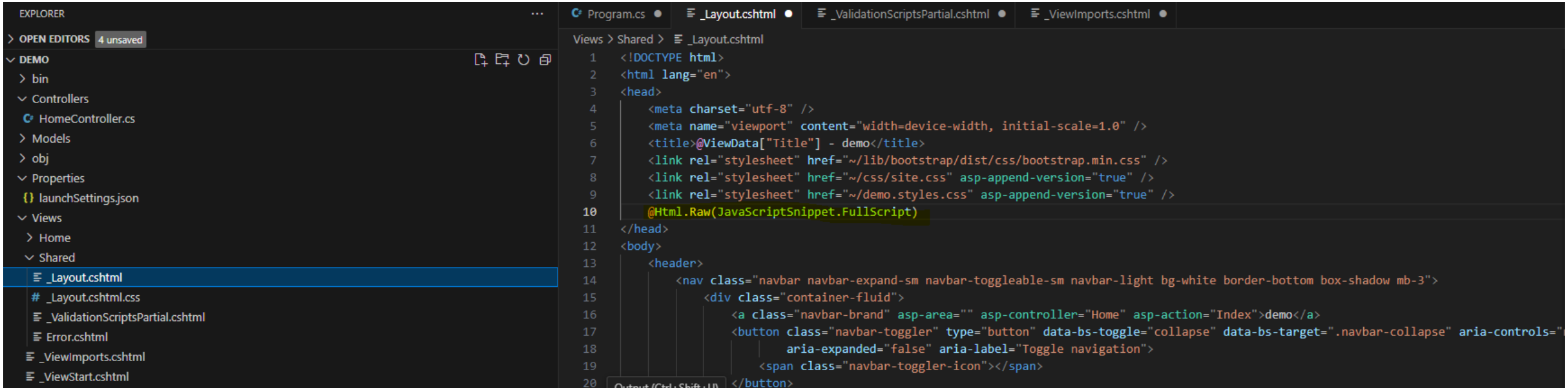
```
PS C:\Users\XIAOMI\demo> dotnet add package Microsoft.ApplicationInsights.AspNetCore
  Determining projects to restore...
  Writing C:\Users\XIAOMI\AppData\Local\Temp\tmpg4duqd.tmp
info : X.509 certificate chain validation will use the default trust store selected by .NET for code signing.
info : X.509 certificate chain validation will use the default trust store selected by .NET for timestamping.
info : Adding PackageReference for package 'Microsoft.ApplicationInsights.AspNetCore' into project 'C:\Users\XIAOMI\demo\demo.csproj'.
info :   GET https://api.nuget.org/v3/registration5-gz-semver2/microsoft.applicationinsights.aspnetcore/index.json
info :   OK https://api.nuget.org/v3/registration5-gz-semver2/microsoft.applicationinsights.aspnetcore/index.json 160ms
info : Restoring packages for C:\Users\XIAOMI\demo\demo.csproj...
info :   GET https://api.nuget.org/v3/vulnerabilities/index.json
info :   OK https://api.nuget.org/v3/vulnerabilities/index.json 116ms
info :   GET https://api.nuget.org/v3-vulnerabilities/2024.10.23.05.45.30/vulnerability.base.json
info :   GET https://api.nuget.org/v3-vulnerabilities/2024.10.23.05.45.30/2024.10.23.05.45.30/vulnerability.update.json
info :   OK https://api.nuget.org/v3-vulnerabilities/2024.10.23.05.45.30/2024.10.23.05.45.30/vulnerability.update.json 91ms
info :   OK https://api.nuget.org/v3-vulnerabilities/2024.10.23.05.45.30/vulnerability.base.json 123ms
info : Package 'Microsoft.ApplicationInsights.AspNetCore' is compatible with all the specified frameworks in project 'C:\Users\XIAOMI\demo\demo.csproj'.
info : PackageReference for package 'Microsoft.ApplicationInsights.AspNetCore' version '2.22.0' added to file 'C:\Users\XIAOMI\demo\demo.csproj'.
info : Writing assets file to disk. Path: C:\Users\XIAOMI\demo\obj\project.assets.json
log  : Restored C:\Users\XIAOMI\demo\demo.csproj (in 826 ms).
PS C:\Users\XIAOMI\demo>
```

services.AddApplicationInsightsTelemetry() - використовується для налаштувань Application Insights у додатках ASP.NET Core, що дозволяє автоматично збирати дані про робочі додатки та передавати їх у сервіс Application Insights

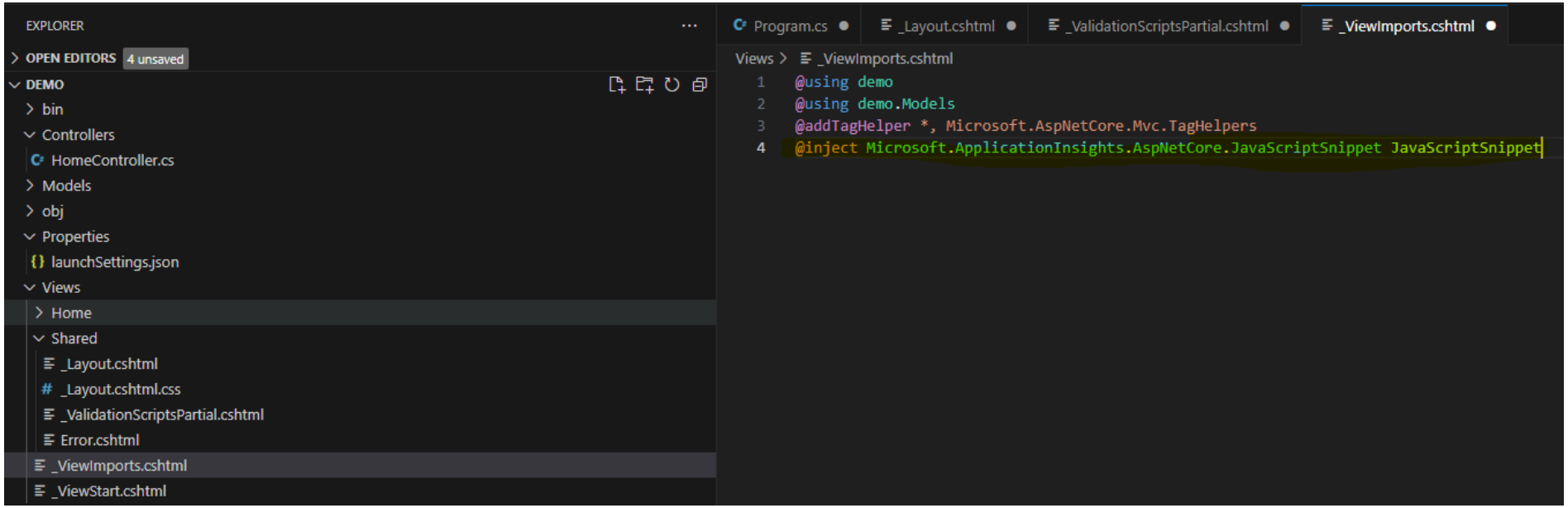


Додавання @Html.Raw(JavaScriptSnippet.FullScript) у файл _Shared/_Layout.cshtml (шаблон макета)

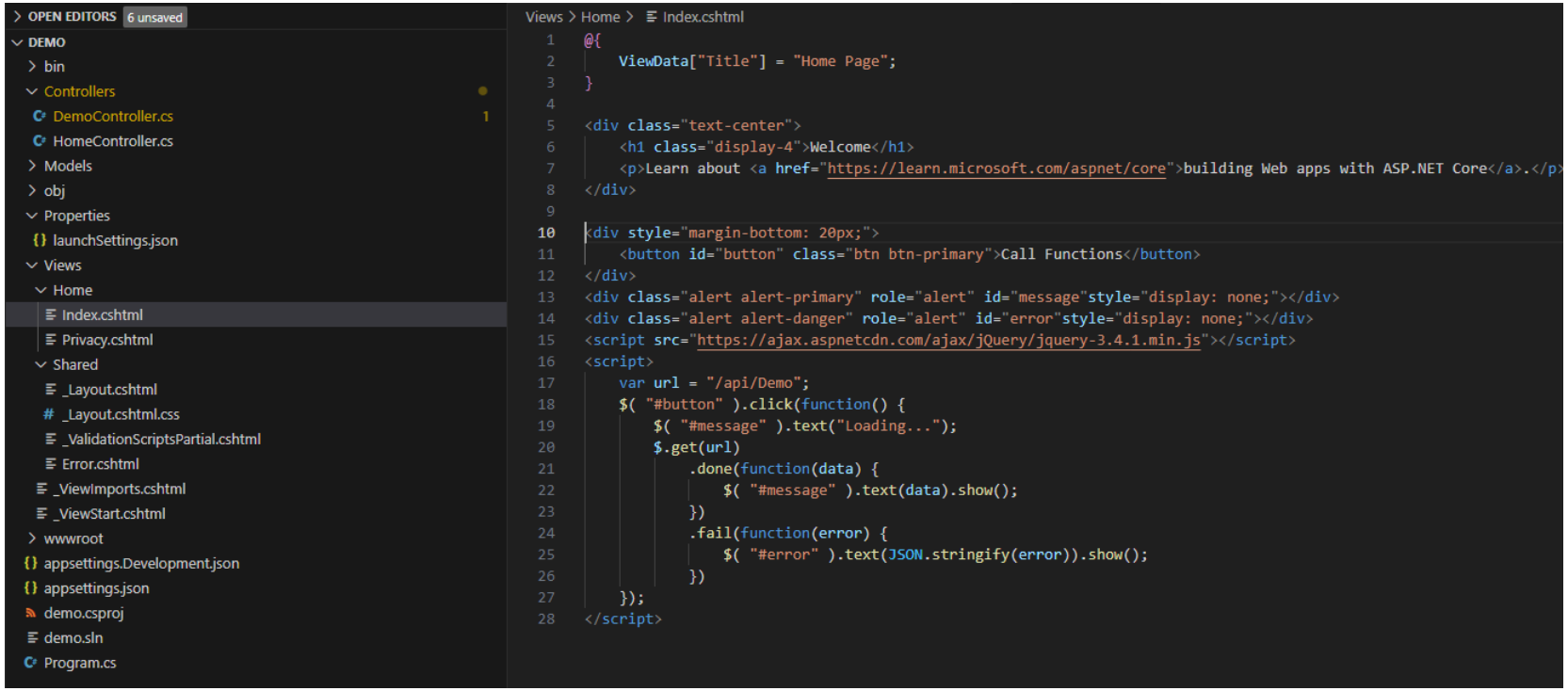
Коли JavaScript-код є частиною інтеграції зовнішнього сервісу (наприклад, Application Insights для моніторингу або аналітики), він повинен бути включений на всіх сторінках програми. Додавання @Html.Raw(JavaScriptSnippet.FullScript) у _Layout.cshtml, забезпечується його завантаження на кожній сторінці, які використовує макет.

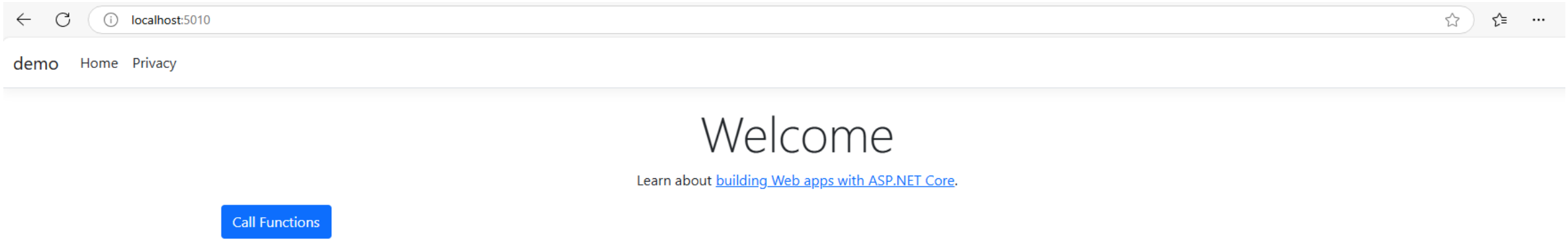


Директива @inject Microsoft.ApplicationInsights.AspNetCore.JavaScriptSnippet JavaScriptSnippet
Це клас, який відповідає за генерацію JavaScript-коду, необхідного для підключення Application Insights на клієнтській стороні. Він вставляє код для відстеження клієнтських даних, таких як час завантаження сторінки, помилки на клієнті, взаємодії з елементами інтерфейсу тощо.
Цей клас містить метод FullScript, який генерує і повертає фрагмент JavaScript для вбудовування на сторінку.



Додаємо DemoController, модифікуємо index.html та компілюємо код





Створюємо Web App в хмарному сховище, розгортаємо веб застосунок

Home >

Create Web App ...

Basics | Database | Deployment | Networking | Monitor + secure | Tags | Review + create

Summary

Web App
by Microsoft

Free sku
Estimated price - Free

Basic authentication for this app is currently disabled and may impact deployments. Click to learn more.

Details

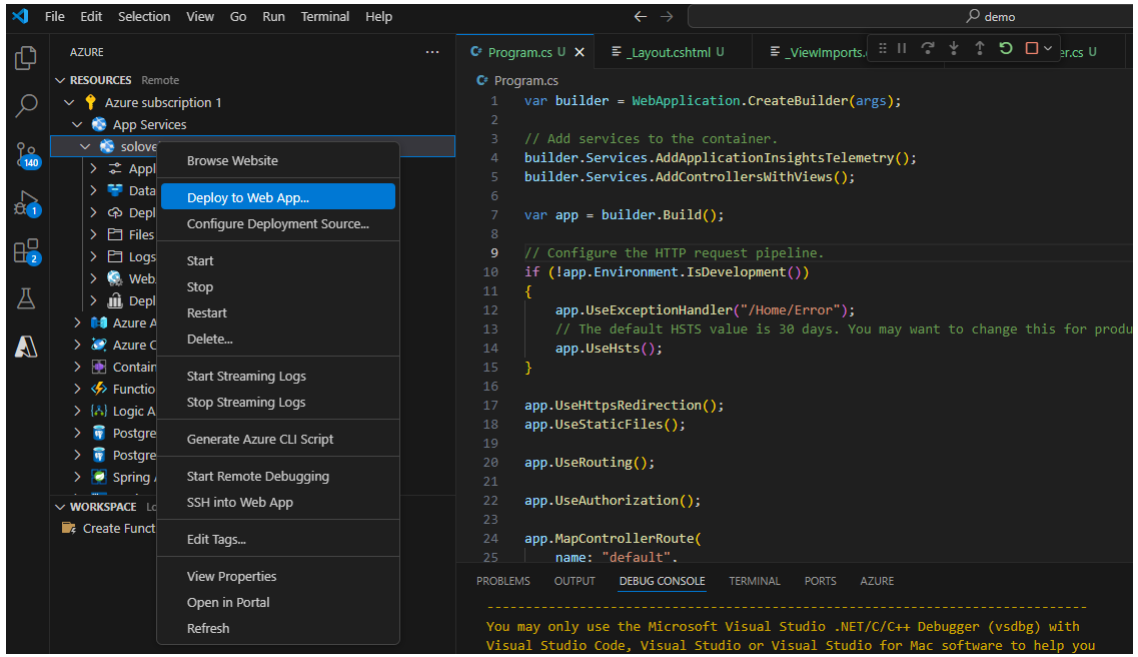
Subscription	601a84f6-579c-4ba6-ac84-00c597a85bd9
Resource Group	demo
Name	soloveidemo
Unique default hostname (preview)	Enabled
Publish	Code
Runtime stack	.NET 8 (LTS)

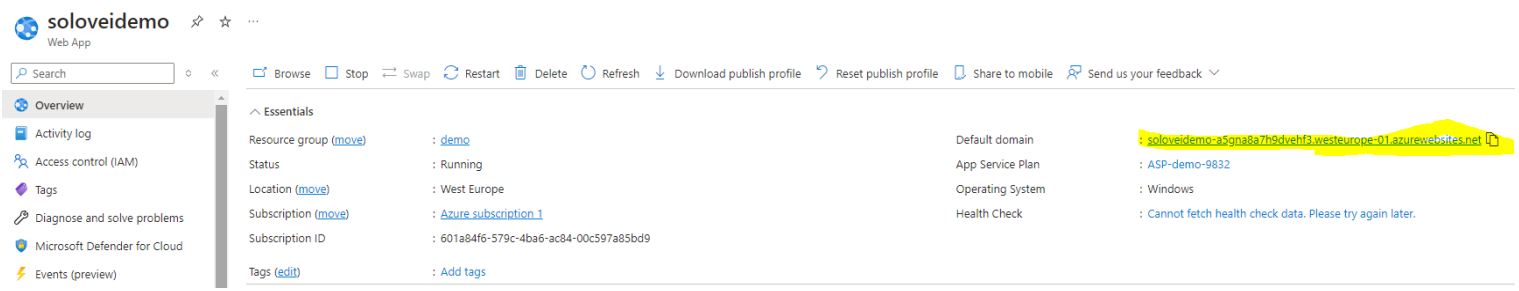
App Service Plan (New)

Name	ASP-demo-9832
Operating System	Windows
Region	West Europe
SKU	Free
ACU	Shared infrastructure
Memory	1 GB memory

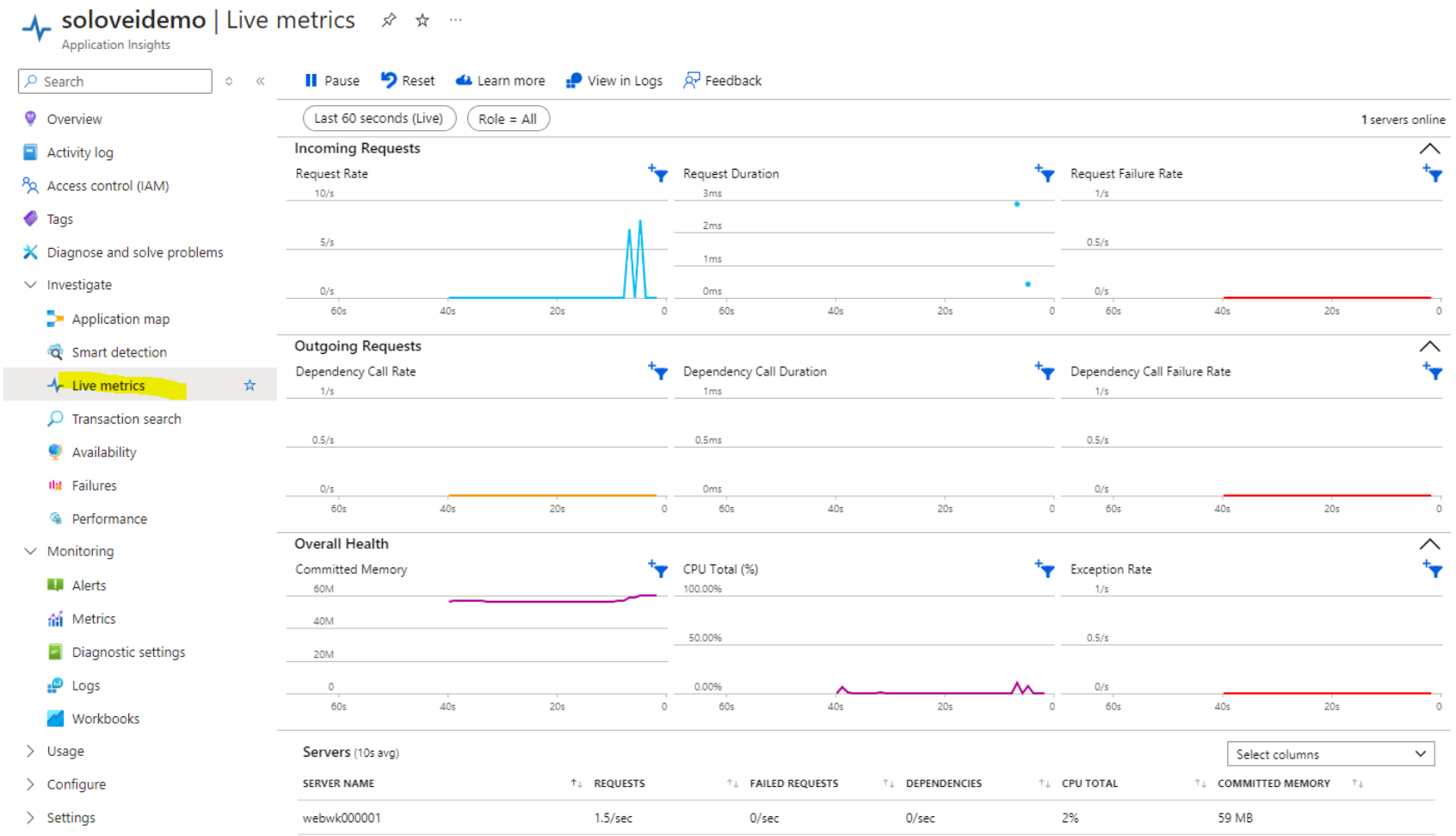
Monitor + secure (New)

Application Insights	Enabled
Name	soloveidemo

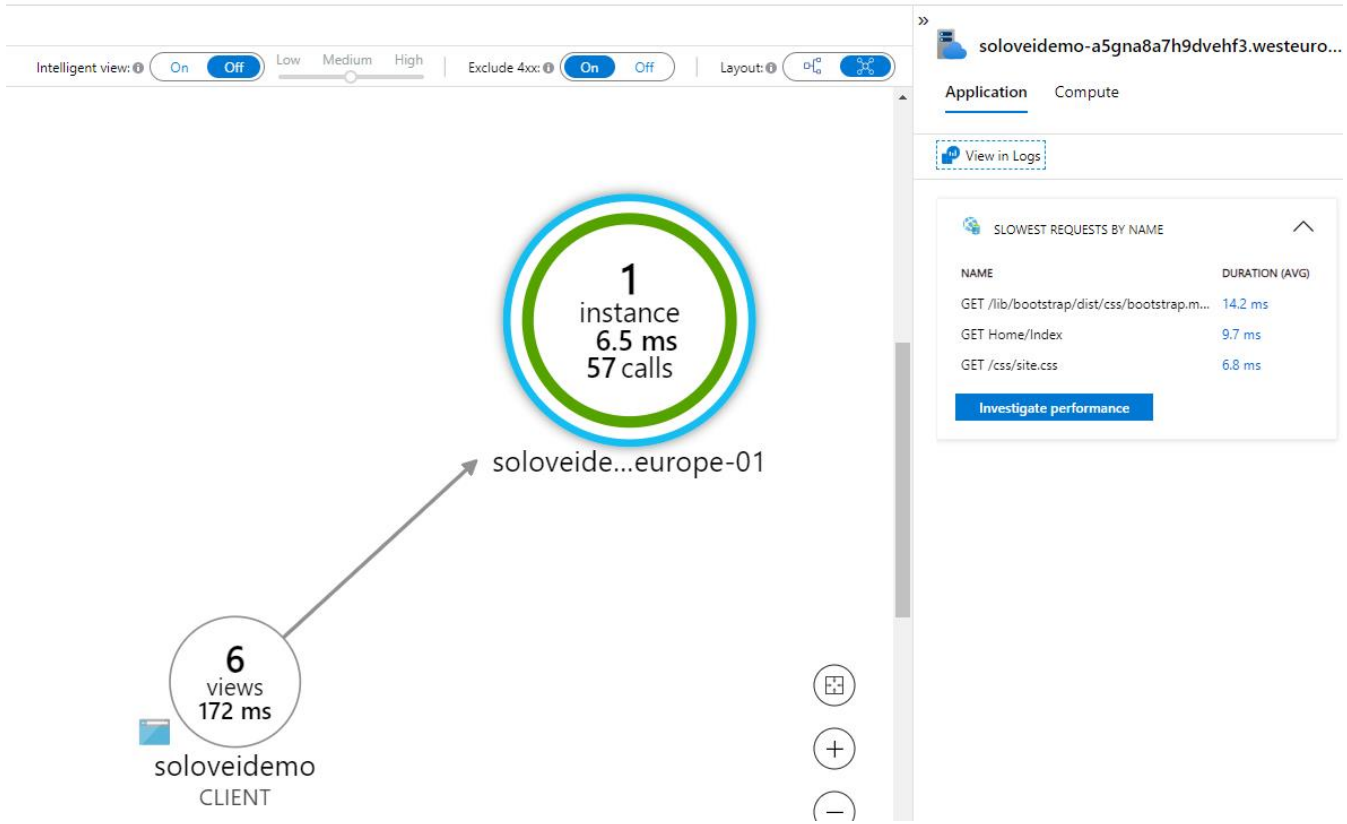




Перейдемо на Application Insights – Live metrics



Application map – веде журнал логів



Застосунок було викликано 57 разів

Azure Function App

Serverless computing — це хмарна модель, у якій хмарний провайдер автоматично керує інфраструктурою, потрібною для виконання вашого коду, а ви фокусуєтесь виключно на написанні й запуску функцій або додатків. Важливою особливістю цієї моделі є те, що сервери дійсно існують, але їх обслуговування та управління приховане від користувача.

Основні принципи Serverless Computing:

Автоматичне масштабування: Сервіси автоматично масштабуються залежно від навантаження. Наприклад, якщо ваш додаток отримує більше запитів, інфраструктура адаптується без необхідності втручання.

Оплата за використання: Ви платите тільки за час виконання вашого коду. Якщо код не виконується, ви нічого не платите.

Відсутність необхідності управління інфраструктурою: Вам не потрібно турбуватися про налаштування серверів, оновлення або резервне копіювання.

Приклади Serverless Computing:

AWS Lambda (Amazon Web Services): Дозволяє запускати код у відповідь на події без потреби в управлінні серверами.

Azure Functions: Функції Microsoft Azure, які працюють як реакція на події.

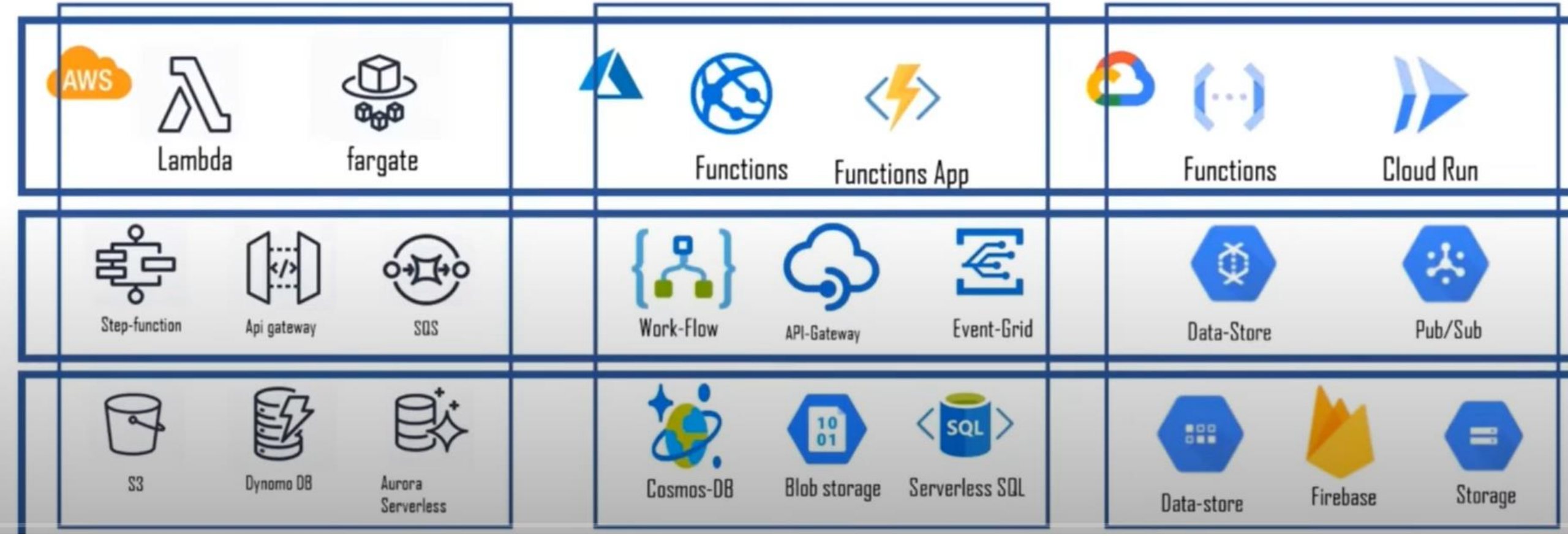
Google Cloud Functions: Аналогічна послуга від Google Cloud.

Приклади Serverless Computing:

AWS Lambda (Amazon Web Services): Дозволяє запускати код у відповідь на події без потреби в управлінні серверами.

Azure Functions: Функції Microsoft Azure, які працюють як реакція на події.

Google Cloud Functions: Аналогічна послуга від Google Cloud.



Cold starts — це затримки, що виникають під час запуску функцій у серверлес обчисленнях (serverless computing) при першому зверненні або після тривалого періоду бездіяльності. В основному, це пов'язано з тим, що інфраструктура повинна спочатку завантажити й ініціалізувати середовище для виконання вашого коду.

Як це працює:

Cold start трапляється, коли функція вперше викликається або довго не використовувалась, і хмарний провайдер повинен:

- Виділити новий контейнер або середовище для виконання функції.
- Завантажити код функції та ініціалізувати залежності (наприклад, бібліотеки).
- Запустити функцію, виконуючи її код.

Warm start — це протилежний стан, коли функція вже виконувалась нещодавно і середовище для її запуску готове. Це дозволяє уникнути додаткової затримки, оскільки функція одразу починає виконуватись.

Чому cold starts відбуваються:

Serverless платформи, такі як AWS Lambda, Azure Functions чи Google Cloud Functions, вимикають середовище виконання, якщо функція тривалий час не використовується, щоб заощадити ресурси.

Це особливо помітно, коли функція має складні або великі залежності, що вимагають більше часу для ініціалізації.

Наслідки cold starts:

Затримка: Cold start може додавати затримку від кількох мілісекунд до кількох секунд, залежно від складності функції та ресурсів.

Проблеми з продуктивністю: Якщо cold start стається під час важливих запитів користувачів, це може погіршити користувацький досвід.

Як зменшити cold starts:

Оптимізація розміру функції: Зменшення розміру залежностей та спрощення коду.

Підтримання функцій у warm стані: Використання спеціальних пінг-запитів для регулярного виклику функцій, що дозволяє тримати середовище активним.

Використання налаштованих контейнерів: Деякі серверлес платформи дозволяють запускати функції у вже налаштованих середовищах для зменшення затримок.

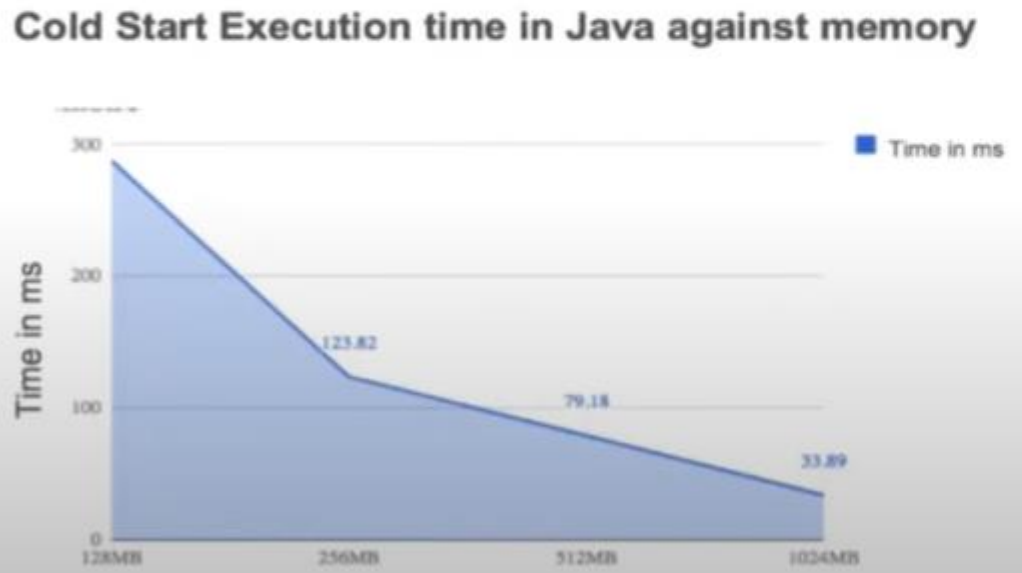
Основні фактори, що впливають на тривалість cold start

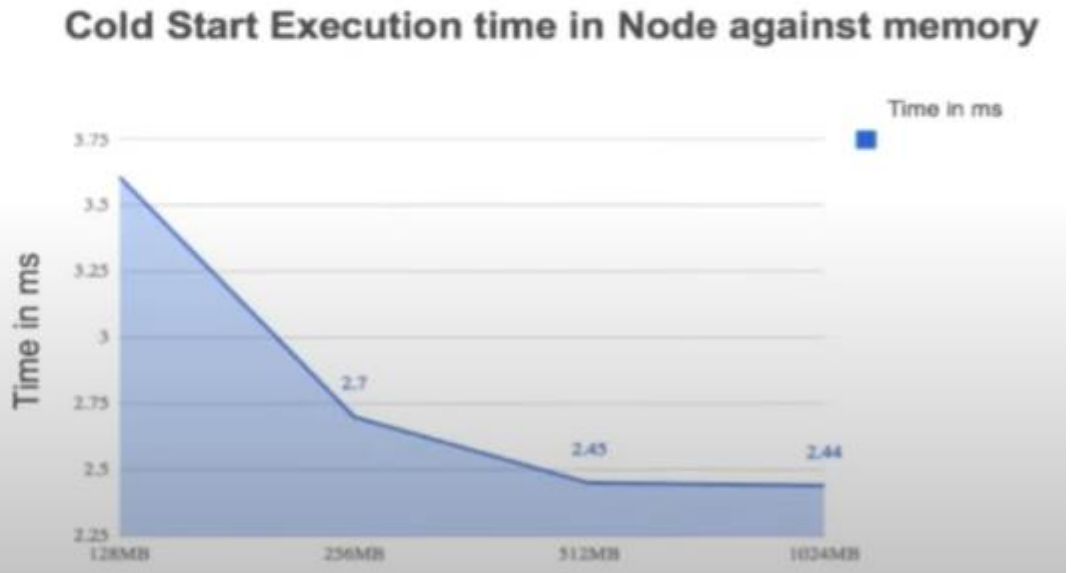
Розмір функції та її залежності; Тип і конфігурація середовища; Виділені ресурси (пам'ять і CPU); Мова програмування; Час завантаження та ініціалізації функції;

Наприклад: Java – 300ms; Node – 3.5ms

Complied Language

Interpreted Language





Розмір функції та її залежності:

Чим більший обсяг коду та кількість бібліотек, тим більше часу займає їх завантаження та ініціалізація.
Великі залежності, особливо важкі бібліотеки, можуть значно збільшувати час cold start.

Тип і конфігурація середовища:
Серверлес провайдери використовують віртуальні машини (VM) або контейнери для виконання функцій.
Налаштування цих середовищ, а також тип використовуваної віртуалізації (наприклад, lightweight контейнери або повноцінні VM) впливають на швидкість запуску.
Менші, оптимізовані середовища запускаються швидше.

Виділені ресурси (пам'ять і CPU):
Функції з більшим обсягом виділеної пам'яті та більш потужними процесорами можуть ініціалізуватись швидше, оскільки такі середовища отримують більше обчислювальних ресурсів.
Наприклад, функції на AWS Lambda з більшою кількістю пам'яті мають швидший cold start, оскільки разом з пам'яттю виділяється більше процесорних ресурсів.

Мова програмування:
Мови, що вимагають компіляції або мають складні середовища виконання (наприклад, Java або .NET), можуть потребувати більше часу на cold start через ініціалізацію JVM (Java Virtual Machine) або CLR (Common Language Runtime).
Легші мови, як Python або Node.js, зазвичай запускаються швидше, оскільки потребують менше ресурсів для старту.

Час завантаження та ініціалізації функції:
Якщо функція взаємодіє з зовнішніми ресурсами (наприклад, базами даних або API), час на cold start також залежатиме від швидкості підключення до цих сервісів та їх ініціалізації.

Consumption Plan (План споживання)

Характеристики:

•**Автоматичне масштабування**: Інфраструктура автоматично масштабується залежно від навантаження. Якщо функція не викликається, ресурси не використовуються.

•**Оплата за використання**: Ви платите лише за обчислювальні ресурси, які споживаються під час виконання функцій. Це включає час виконання функцій і кількість виконаних запитів.

•**Обмеження часу виконання**: Максимальний час виконання функції становить 5 хвилин за замовчуванням, з можливістю збільшити цей ліміт до 10 хвилин.

•**Cold starts**: Під час першого виклику функції (або після тривалої бездіяльності) відбувається "cold start", що може призвести до певної затримки.

•**Без фіксованих ресурсів**: Виділені ресурси, такі як CPU та пам'ять, не є фіксованими, а динамічно виділяються під час виклику функції.

•**Підтримка тільки HTTP-триггерів та деяких подій**: Не всі сценарії можуть бути оброблені ефективно через обмеження по тривалій роботі з підключеннями.

Ідеальні сценарії:

•Нерегулярні або спорадичні запити.

•Коли важливо мінімізувати витрати і функція виконується рідко або швидко.

•Невеликі або менш критичні додатки.

Premium Plan (Преміум план)

Характеристики:

•**Постійні ресурси**: Функції завжди мають виділені ресурси (CPU і пам'ять), що дозволяє уникати cold starts і гарантує швидкий час відповіді навіть після тривалої бездіяльності.

•**Підтримка тривалих виконань**: На відміну від Consumption Plan, Premium Plan дозволяє функціям працювати без обмеження часу виконання.

•**Масштабування на основі попиту**: Масштабування відбувається на основі попиту, але з мінімальним і максимальним обмеженням кількості інстанцій. Це дозволяє краще контролювати використання ресурсів і уникати надмірних витрат.

•**Оптимізована продуктивність**: Завдяки відсутності cold starts і стабільним ресурсам, функції працюють стабільніше і з меншими затримками.

•**Підтримка VNET**: Функції в Premium Plan можуть бути підключені до віртуальних мереж (VNET), що важливо для сценаріїв з підвищеними вимогами до безпеки та конфіденційності.

•**Гнучка конфігурація**: Premium Plan дозволяє налаштувати потужність інфраструктури, виділивши певні ресурси для кожної функції.

Ідеальні сценарії:

•Високонавантажені додатки або додатки з критичною продуктивністю.

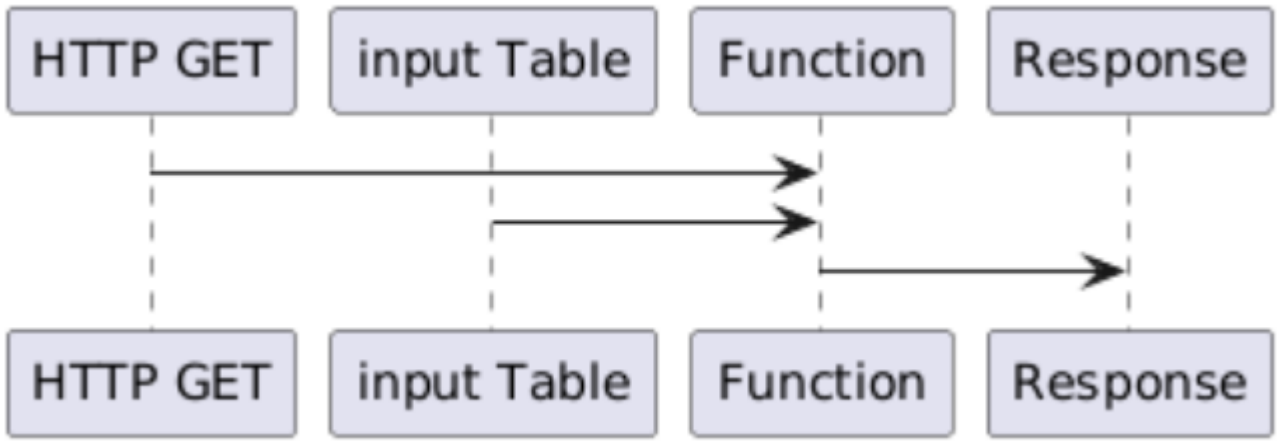
•Функції, які повинні бути завжди готові до виконання (без cold starts).

•Додатки з великими вимогами до часу виконання або з складними залежностями.

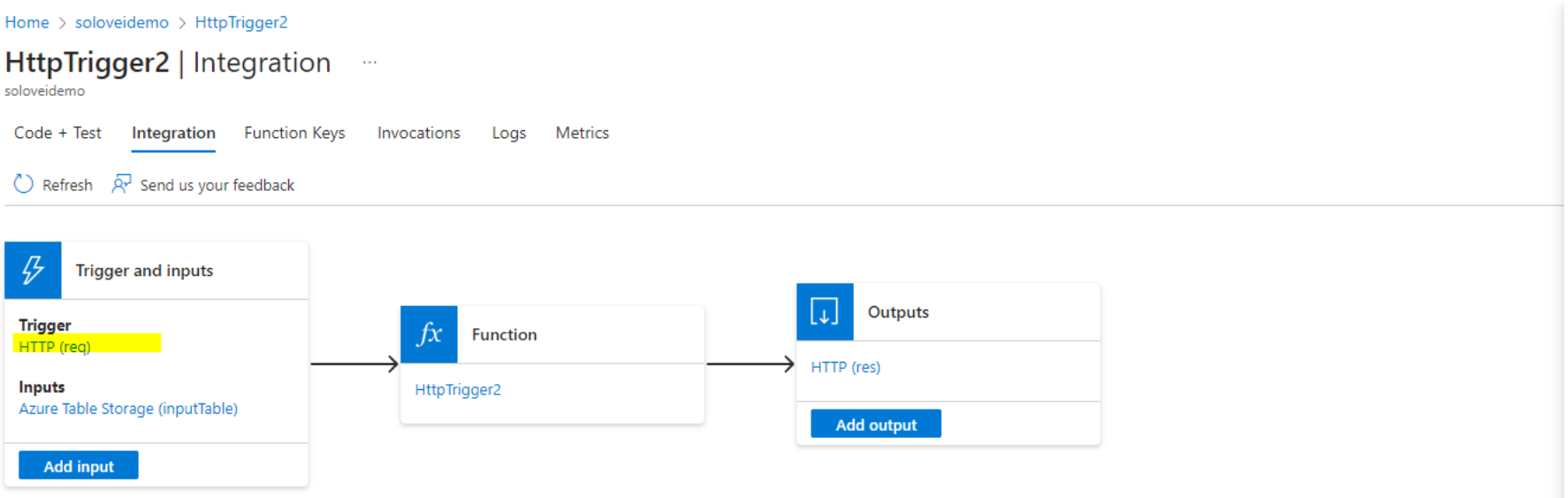
•Додатки, що вимагають підключення до віртуальних мереж або доступу до захищених ресурсів.

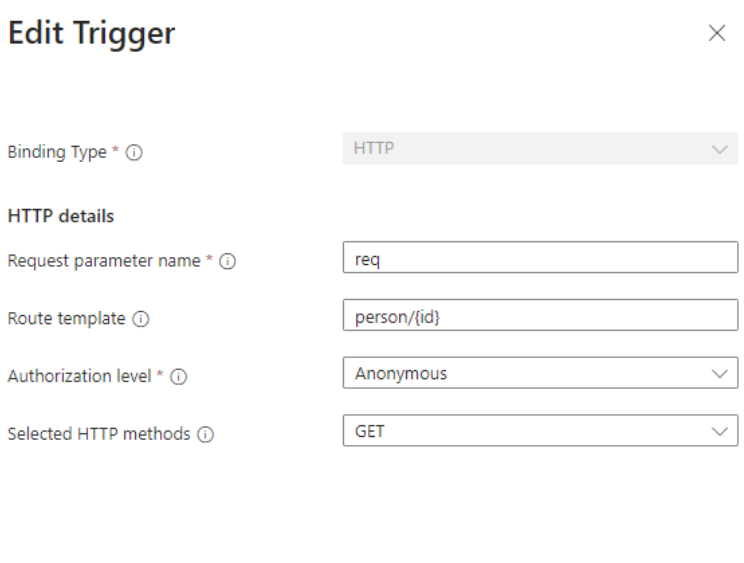
Характеристика	Consumption Plan	Premium Plan
Масштабування	Автоматичне, без обмежень	Автоматичне, але з контролем кількості інстанцій
Оплата	Тільки за використані ресурси	Оплата за зарезервовані ресурси
Cold starts	Присутні	Відсутні
Максимальний час виконання	5-10 хвилин	Без обмежень
Постійні ресурси	Ні	Так
Підтримка VNET	Ні	Так
Продуктивність	Може бути нерівномірною через cold starts	Гарантована стабільність

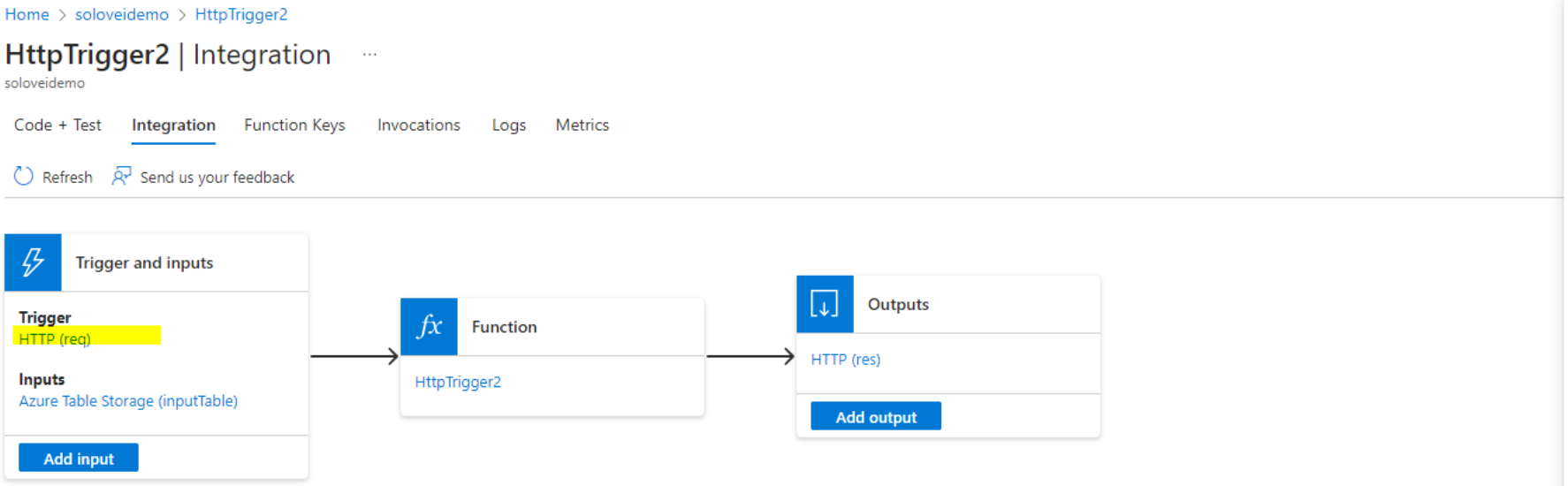
Процес для отримання даних з хмарного сховища типу Таблиця

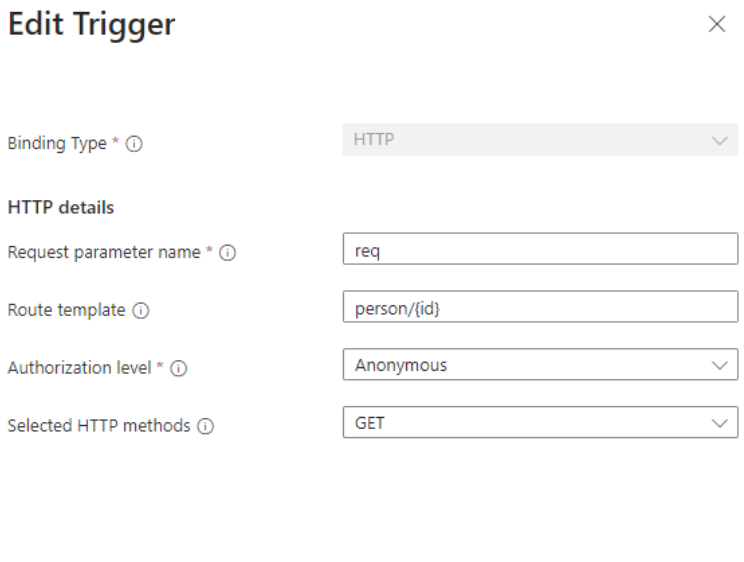


Вхідні змінні

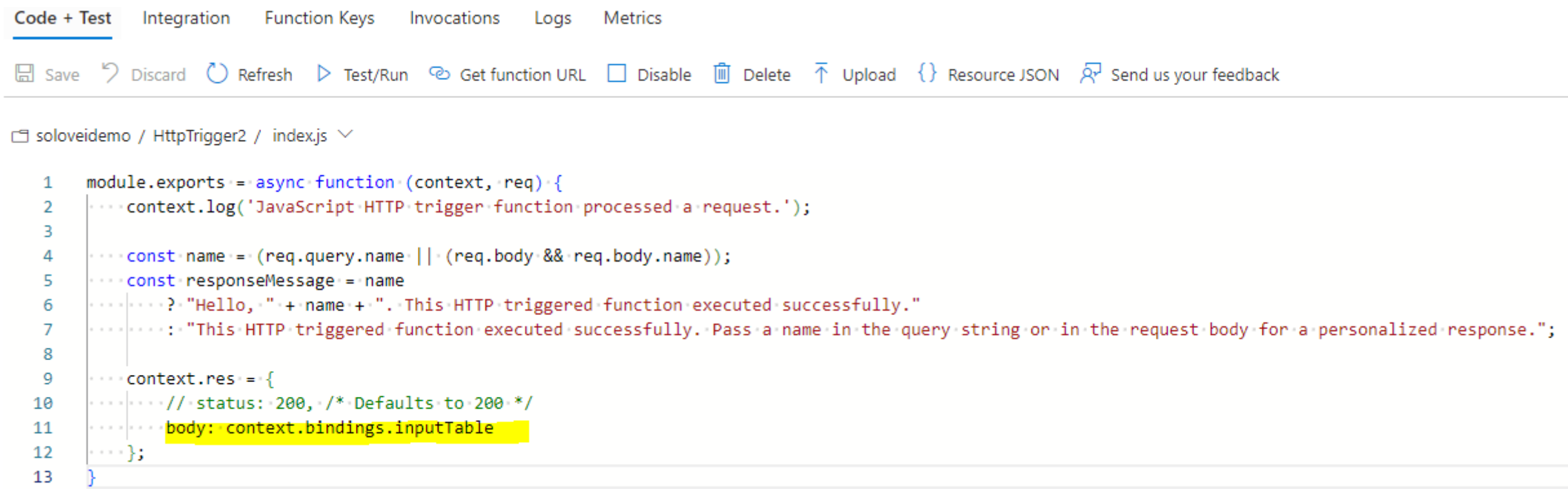




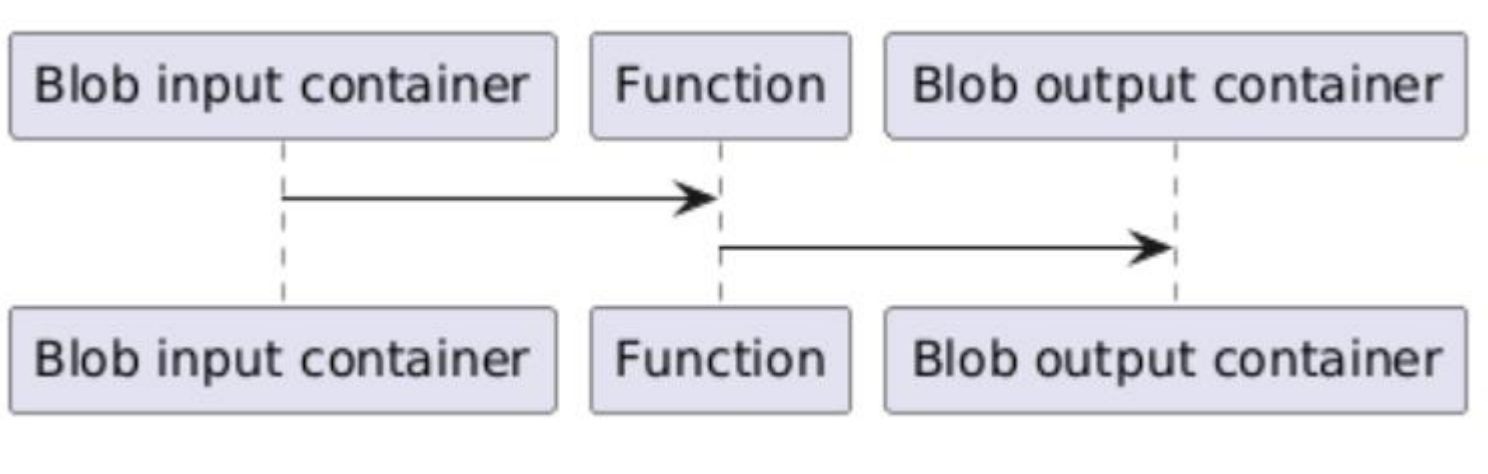




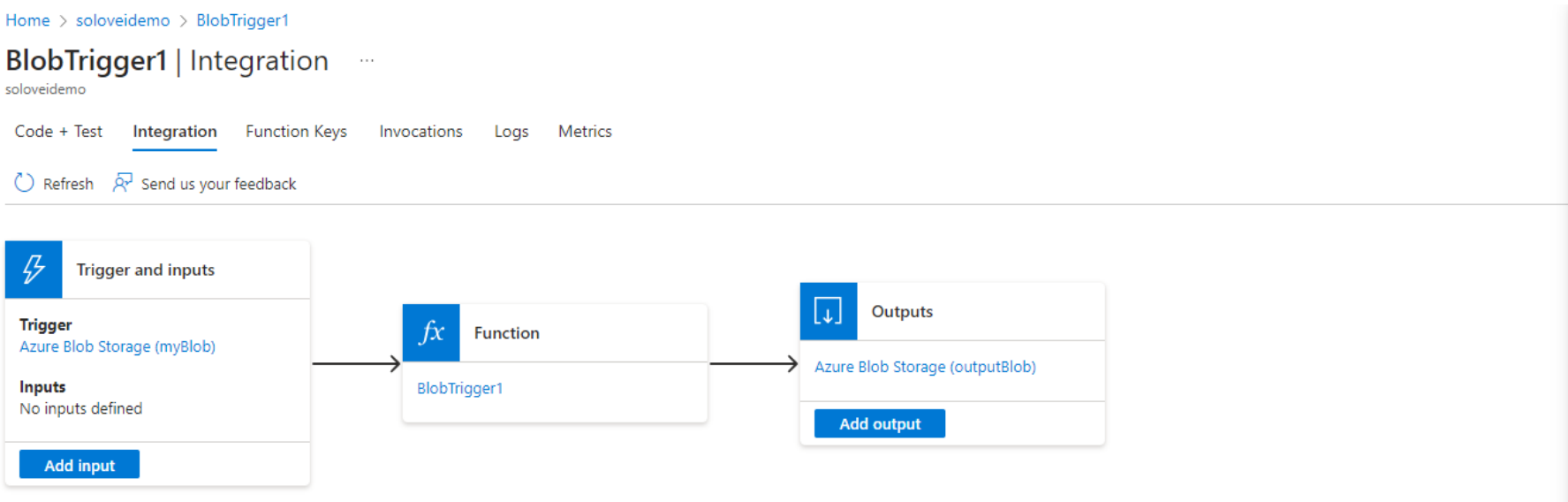
Функція

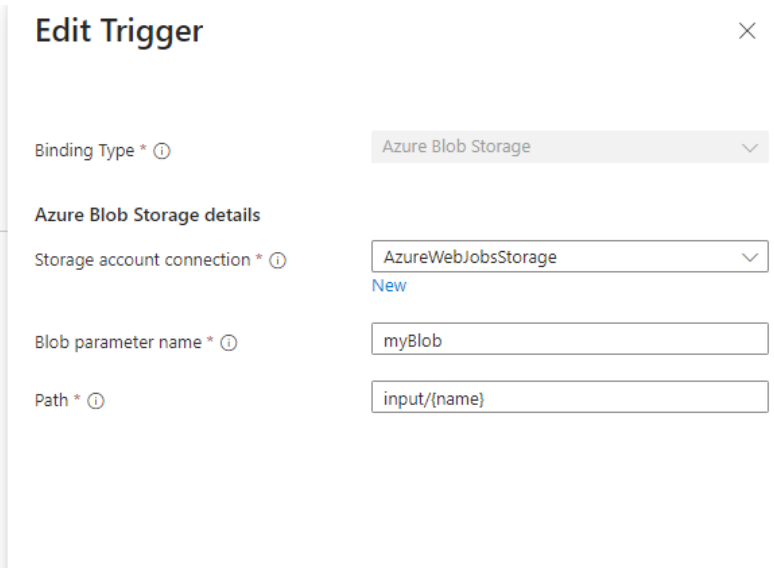

Code + Test
Integration
Function Keys
Invocations
Logs
Metrics
Save
Discard
Refresh
Test/Run
Get function URL
Disable
Delete
Upload
Resource JSON
Send us your feedback
soloveidemo / HttpTrigger2 / index.js
1 module.exports = async function (context, req) {
2 context.log('JavaScript HTTP trigger function processed a request.');
3
4 const name = (req.query.name || (req.body && req.body.name));
5 const responseMessage = name
6 ? "Hello, " + name + ". This HTTP triggered function executed successfully."
7 : "This HTTP triggered function executed successfully. Pass a name in the query string or in the request body for a personalized response.";
8
9 context.res = {
10 // status: 200, /* Defaults to 200 */
11 body: context.bindings.inputTable
12 };
13 }

Процес для копіювання файлів між контейнерами двійкового сховища

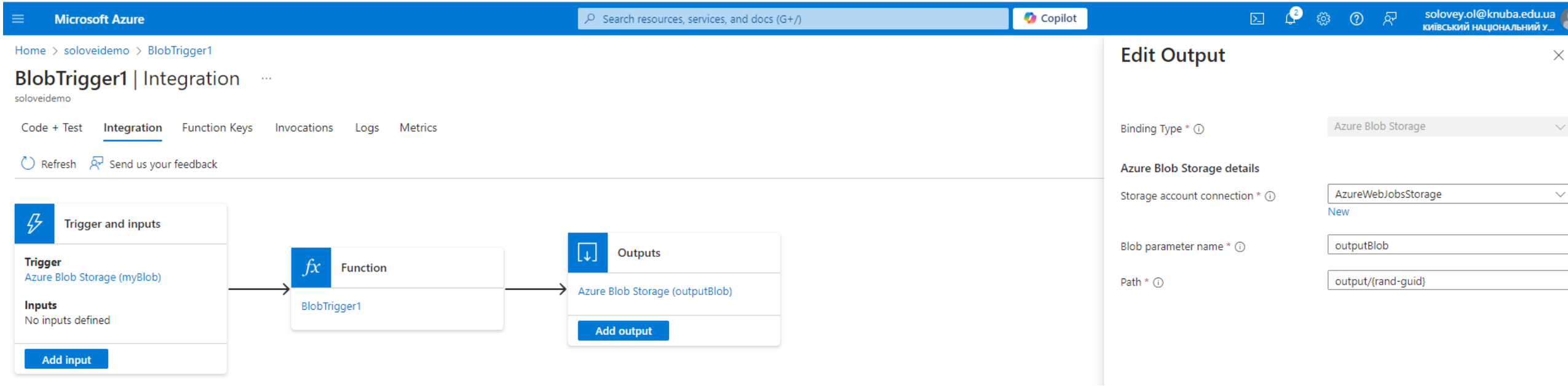


Вхідні дані

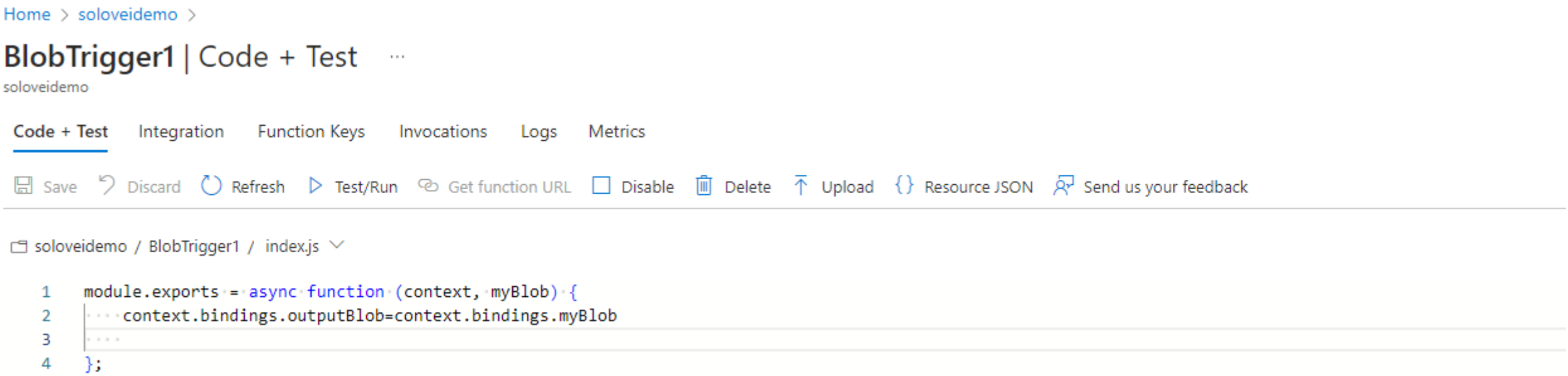

Home > soloveidemo > BlobTrigger1
BlobTrigger1 | Integration
soloveidemo
Code + Test
Integration
Function Keys
Invocations
Logs
Metrics
Refresh
Send us your feedback
Trigger and inputs
Trigger
Azure Blob Storage (myBlob)
Inputs
No inputs defined
Add input
Function
BlobTrigger1
Outputs
Azure Blob Storage (outputBlob)
Add output


Edit Trigger
Binding Type *
Azure Blob Storage
Azure Blob Storage details
Storage account connection *
AzureWebJobsStorage
New
Blob parameter name *
myBlob
Path *
input/{name}

Вихідні дані



Функція


Home > soloveidemo >
BlobTrigger1 | Code + Test
soloveidemo
Code + Test
Integration
Function Keys
Invocations
Logs
Metrics
Save
Discard
Refresh
Test/Run
Get function URL
Disable
Delete
Upload
Resource JSON
Send us your feedback
soloveidemo / BlobTrigger1 / index.js
1 module.exports = async function (context, myBlob) {
2 context.bindings.outputBlob=context.bindings.myBlob
3
4 };

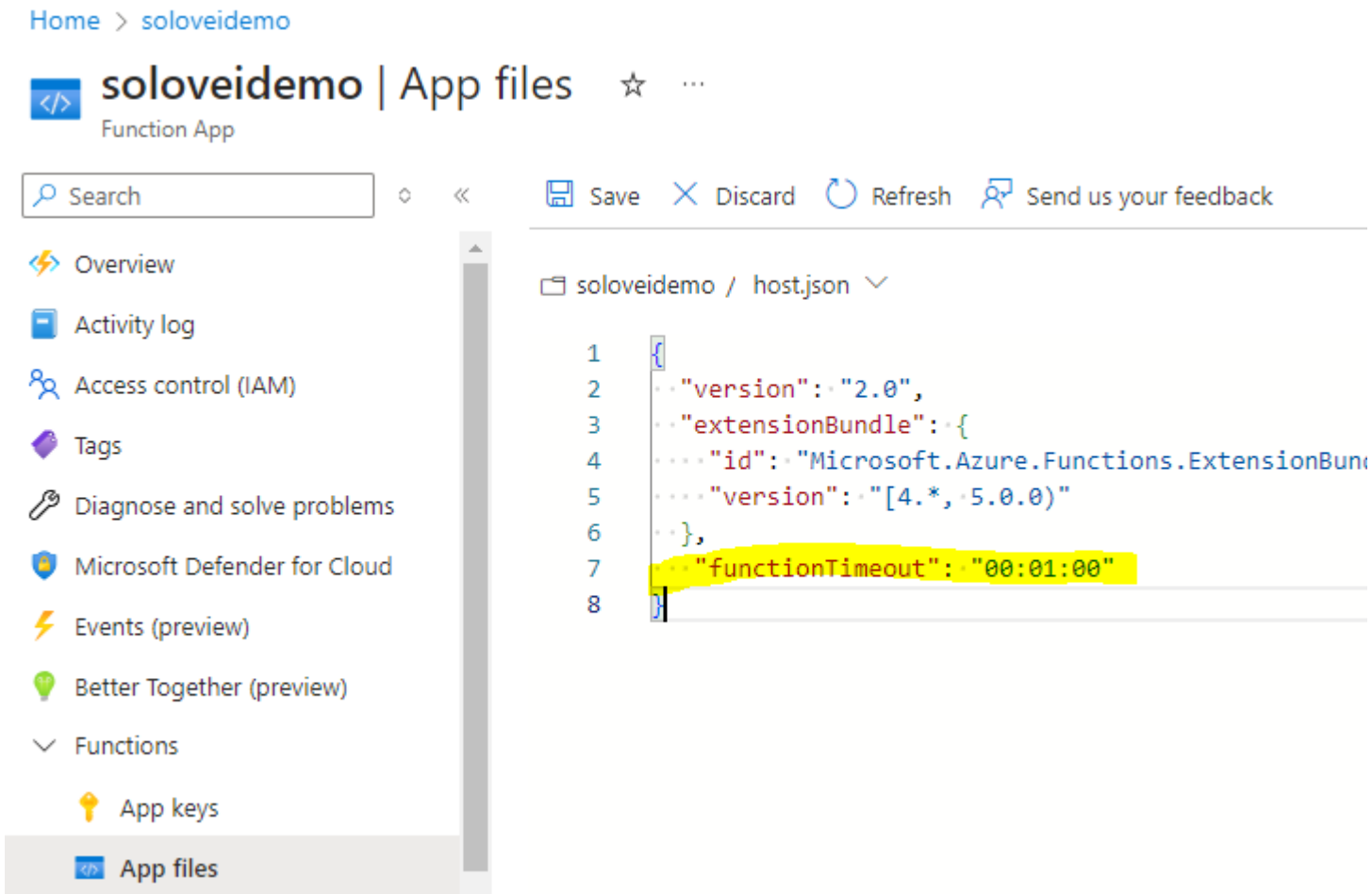
App file – host.json - https://learn.microsoft.com/en-us/azure/azure-functions/functions-host-json

Файл host.json є конфігураційним файлом для Azure Functions, який визначає загальні налаштування хоста для функціонального додатка. Основні характеристики та параметри, які можна налаштувати в цьому файлі:

1. “version”: Визначає версію схеми host.json. Це визначає сумісність файлу з версією платформи Azure Functions. Наприклад, версія 2.0 підтримується для більшості нових функцій.
2. functionTimeout - Встановлює максимальний час виконання функції перед її примусовим завершенням. Цей параметр важливий для функцій, які можуть виконуватися довго. Якщо значення не вказане, то тайм-аут за замовчуванням для консумпційного плану становить 5 хвилин.
3. Logging - Налаштовує параметри логування та діагностики.


```
{
    "version": "2.0",
    "aggregator": {
        "batchSize": 1000,
        "flushTimeout": "00:00:30"
    },
    "concurrency": {
            "dynamicConcurrencyEnabled": true,
            "snapshotPersistenceEnabled": true
        },
    "extensions": {
        "blobs": {},
        "cosmosDb": {},
        "durableTask": {},
        "eventHubs": {},
        "http": {},
        "queues": {},
        "sendGrid": {},
        "serviceBus": {}
    },
    "extensionBundle": {
        "id": "Microsoft.Azure.Functions.ExtensionBundle",
        "version": "[4.0.0, 5.0.0)"
    },
    "functions": [ "QueueProcessor", "GitHubWebHook" ],
    "functionTimeout": "00:05:00",
    "healthMonitor": {
        "enabled": true,
        "healthCheckInterval": "00:00:10",
        "healthCheckWindow": "00:02:00",
        "healthCheckThreshold": 6,
        "counterThreshold": 0.80
    },
    "logging": {
        "fileLoggingMode": "debugOnly",
        "logLevel": {
          "Function.MyFunction": "Information",
          "default": "None"
        },
        "applicationInsights": {
            "samplingSettings": {
              "isEnabled": true,
              "maxTelemetryItemsPerSecond" : 20,
              "evaluationInterval": "01:00:00",
              "initialSamplingPercentage": 100.0,
              "samplingPercentageIncreaseTimeout" : "00:00:01",
              "samplingPercentageDecreaseTimeout" : "00:00:01",
              "minSamplingPercentage": 0.1,
```

Визначити максимальний час виконання функції



Секція extensions

Logging - Налаштовує параметри логування та діагностики

Визначає фільтрацію за категорією

defaultLevel - Для будь-яких категорій, не вказаних у масиві categoryLevels, надсилайте журнали цього рівня та вище до Application Insights.

categoryLevels - Масив категорій, який визначає мінімальний рівень журналу для надсилання до Application Insights для кожної категорії. Зазначена тут категорія контролює всі категорії, які починаються з однакового значення, а довші значення мають пріоритет. У попередньому прикладі файлу host.json усі категорії, які починаються з «Host.Aggregator», реєструються на рівні інформації. Усі інші категорії, які починаються з «Host», наприклад «Host.Executor», реєструються на рівні помилки.

```
{ "logger":
{ "categoryFilter":
{ "defaultLevel": "Information",
"categoryLevels": { "Host": "Error", "Function": "Error", "Host.Aggregator":
"Information" } } } }
```

Секція extensions

Секція extensions у файлі host.json Azure Functions використовується для налаштування різних розширень, які дозволяють функціям взаємодіяти з різними службами або зовнішніми ресурсами, такими як Azure Storage, Service Bus, Event Hubs тощо. Кожне з цих розширень відповідає за певний тригер або прив'язку в Azure Functions.

1. blobs (Azure Storage Blobs) - Конфігурація може містити параметри, такі як maxDegreeOfParallelism (максимальна кількість одночасних операцій), batchSize (розмір пакета для обробки об'єктів) тощо.
2. cosmosDb (Azure Cosmos DB)- дозволяє функціям взаємодіяти з базою даних Azure Cosmos DB. Функції можуть реагувати на зміни у записах або виконувати запити до Cosmos DB.
3. Конфігурація може включати налаштування, такі як leaseCollectionThroughput (пропускна здатність для колекцій лізингів).
4. durableTask (Durable Functions) - Durable Functions — це надбудова над Azure Functions для створення оркестрацій та інших компонентів Durable Functions.
5. eventHubs (Azure Event Hubs) - Це розширення дозволяє функціям працювати з Event Hubs, які є ***системами потокової передачі даних для обробки великих обсягів подій***.
6. http (HTTP Triggers and Bindings) – для налаштування конфігурацій HTTP-запитів такі як routePrefix (префікс для маршрутів HTTP-запитів);

maxOutstandingRequests (максимальна кількість активних запитів).

Секція extensions
http (HTTP Triggers and Bindings)

1. dynamicThrottlesEnabled - коли цей параметр увімкнено, цей параметр змушує конвеєр обробки запитів періодично перевіряти лічильники продуктивності системи, наприклад з'єднання/потоки/процеси/пам'ять/процесор тощо. і якщо будь-який із цих лічильників перевищує встановлений високий поріг (80%), запити відхиляються відповіддю 429 «Занадто зайнятий», доки лічильник(и) не повернуться до нормального рівня.
2. maxConcurrentRequests - Максимальна кількість HTTP-функцій, які виконуватимуться паралельно. Це дозволяє контролювати паралельність, що може допомогти керувати використанням ресурсів. Наприклад, у вас може бути функція HTTP, яка використовує багато системних ресурсів (пам'ять/процесор/сокети), що спричиняє проблеми, коли паралелізм надто високий. Або у вас може бути функція, яка надсилає вихідні запити сторонній службі, і ці виклики мають бути обмежені швидкістю. У цих випадках може допомогти застосування дросельної заслінки.
3. routePrefix - Префікс маршруту, який застосовується до всіх маршрутів. Порожній рядок, щоб видалити префікс за замовчуванням.
4. maxOutstandingRequests - Максимальна кількість нерозглянутих запитів, які зберігаються в будь-який момент часу. Цей ліміт включає запити, які поставлено в чергу, але ще не почали виконувати, а також будь-які поточні виконання. Будь-які вхідні запити, що перевищують цей ліміт, відхиляються відповіддю 429 "Занадто зайнятий". Це дозволяє абонентам використовувати стратегії повторних спроб на основі часу, а також допомагає контролювати максимальні затримки запитів. Це лише керування чергуванням, яке відбувається в межах шляху виконання сценарію. Інші черги, як-от черга запитів ASP.NET, залишатимуться в силі й на них цей параметр не впливатиме.

```
{
  "http": {
    "routePrefix": "api",
    "maxOutstandingRequests": 200,
    "maxConcurrentRequests": 100,
    "dynamicThrottlesEnabled": true
  }
}
```