## Учреждение образования

## Белорусский государственный технологический университет

Кафедра информационных систем и технологий

**ЛАБОРАТОРНАЯ РАБОТА № 11**

по дисциплине «Корпоративные информационные системы»

Тема

«Мониторинг и оптимизация решений Azure»

Выполнил студент

4 курса 5 группы, Демьянов В.Р.

Минск 2023

**Цель работы:** изучение возможностей Azure для мониторинга и оптимизации решений.

**Ход работы**

**1 Создание ресурса для аналитики приложения**

На рисунке 1 представлен процесс создания ресурса для аналитики приложения.

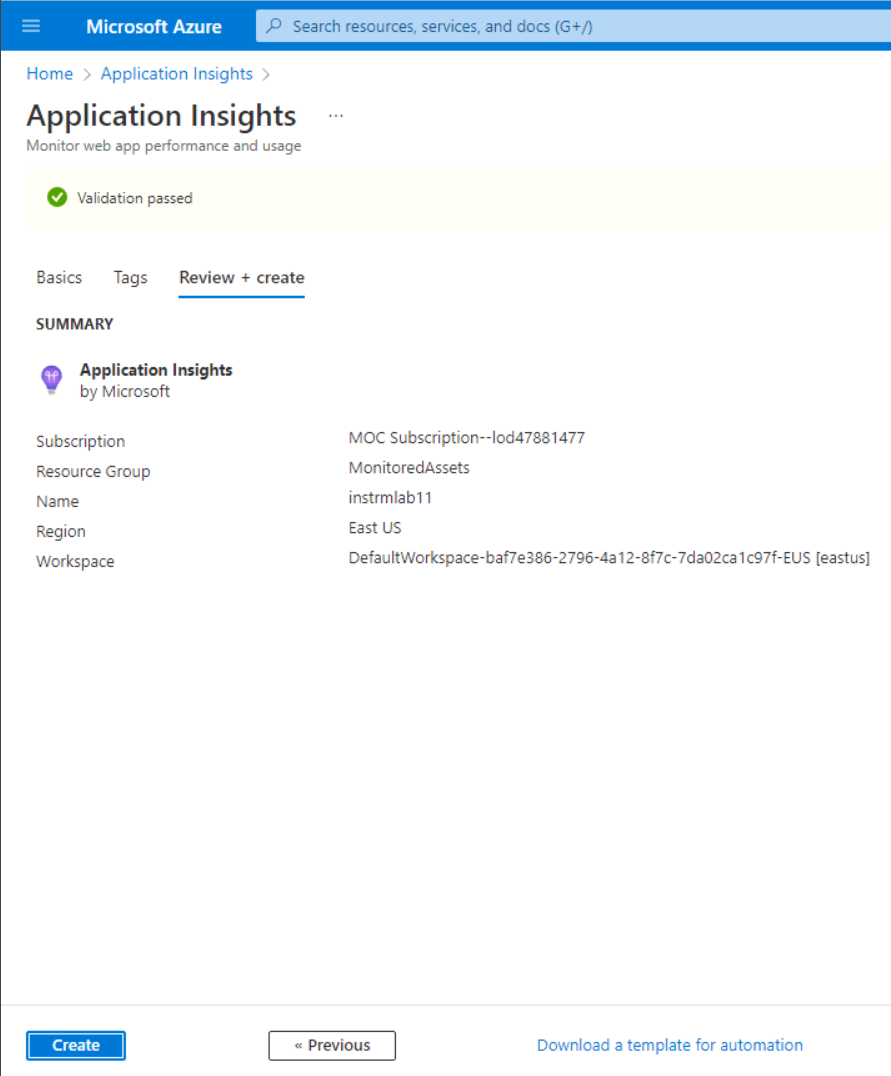


Рисунок 1 – Процесс создания ресурса для аналитики приложения

Далее нужно скопировать значение instrumentation key (рисунок 2). Данный ключ используется клиентским приложением для подключения к ресурсу для аналитики.

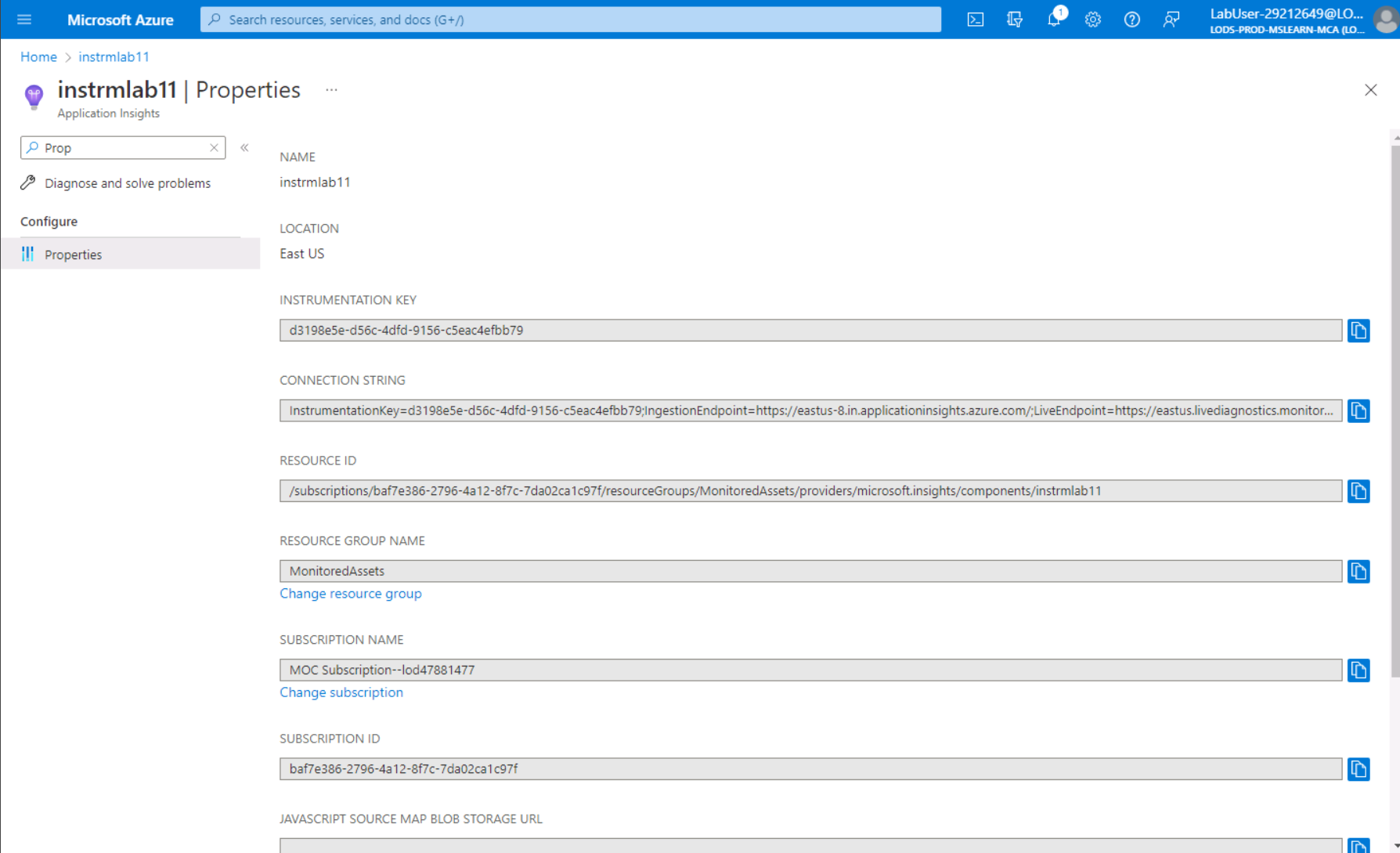


Рисунок 2 – Instrumentation key

Далее создаём ресурс приложения, которое мы будем мониторить (рисунок 3).

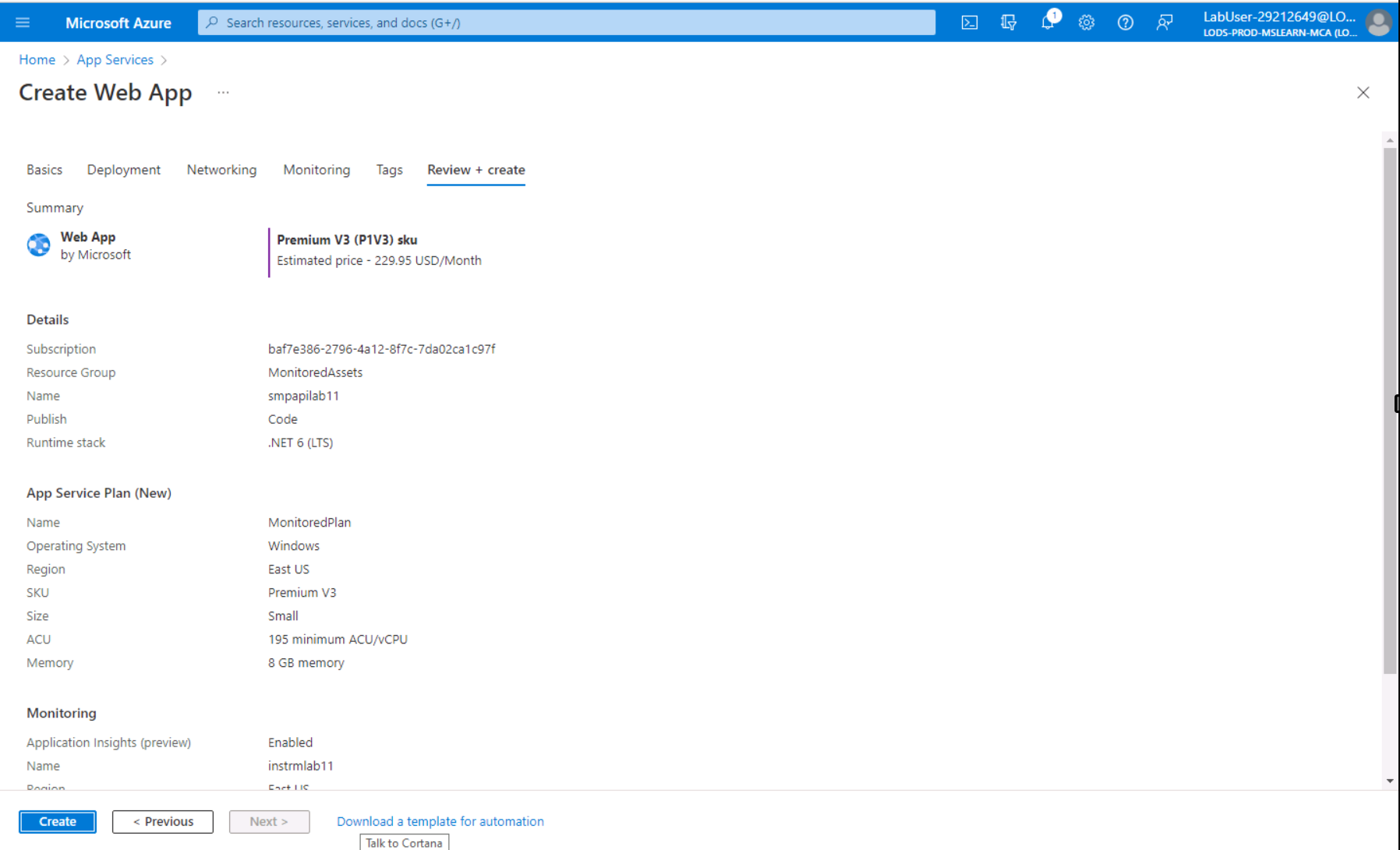


Рисунок 3 – Создание ресурса приложения, которое будет анализироваться

Далее необходимо скопировать значение URL (рисунок 4), которое будет использоваться для подтверждения запросов к Web API.

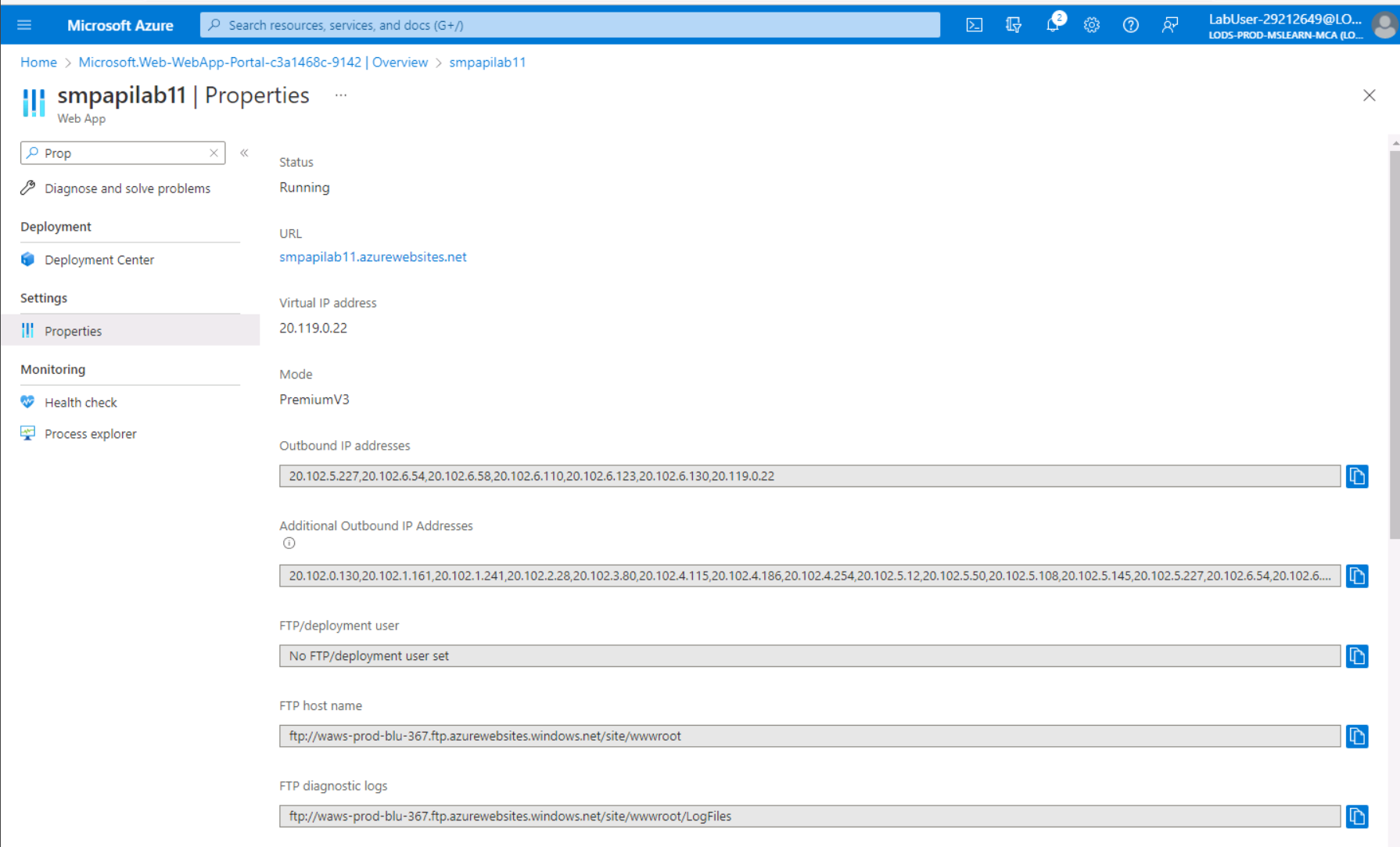


Рисунок 4 – Web API URL

На рисунке 5 продемонстрирована конфигурация параметров автомасштабирования.

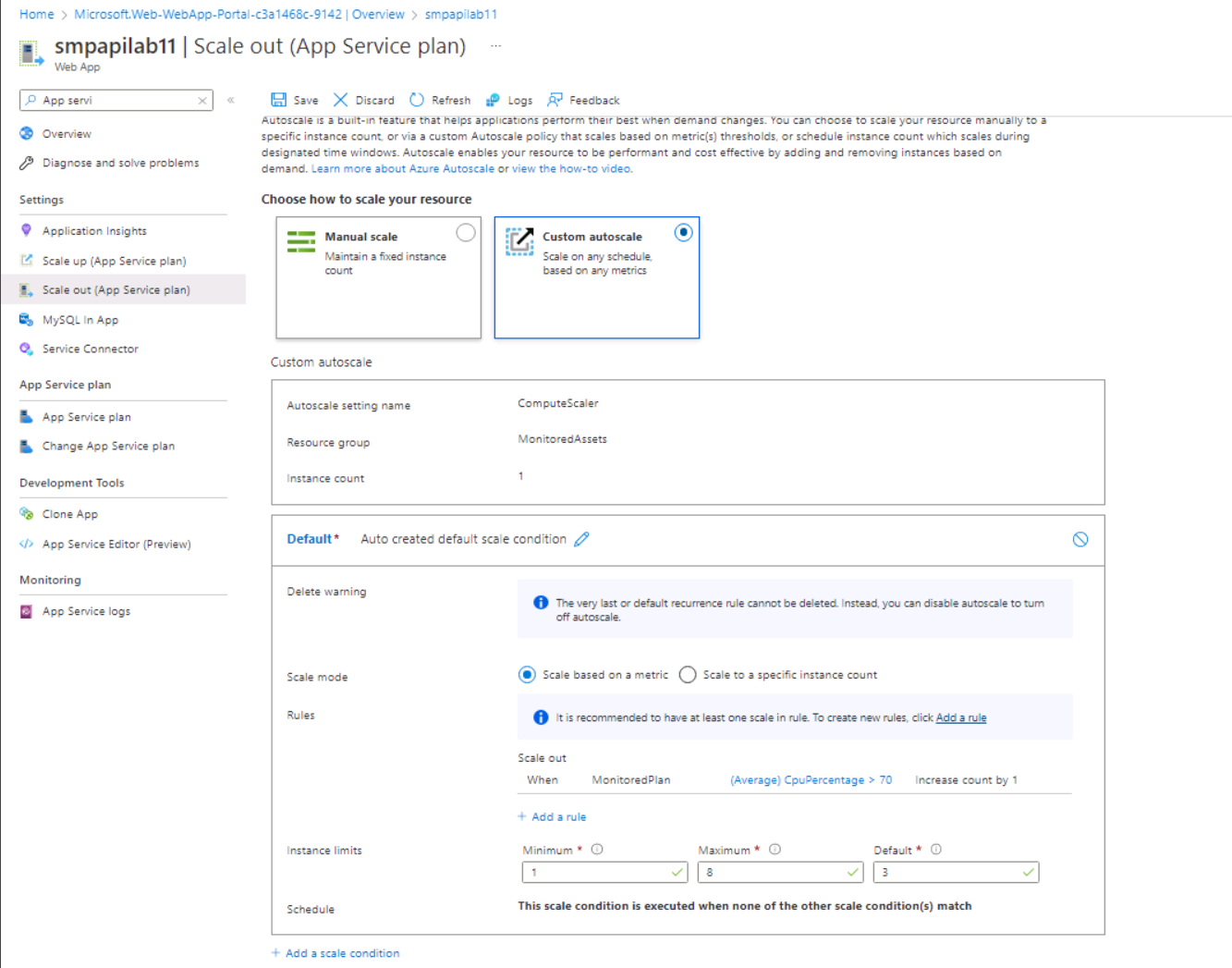


Рисунок 5 – Конфигурация параметров автомасштабирования

**2 Мониторинг локального Web API с использованием Application Insights**

Создание Web API приложения (рисунок 6).

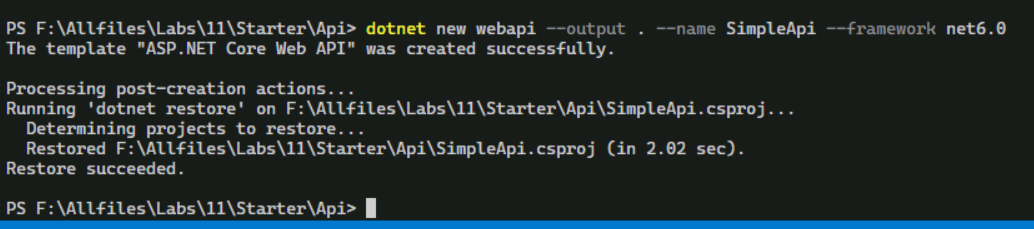


Рисунок 6 – Создание Web API приложения

Далее необходимо отключить переадресацию HTTPS в Program.cs, удалив строчки как в листинге ниже.

|  |
| --- |
| builder.Services.AddServiceProfiler();  app.UseHttpsRedirection(); |

Task 2: Update app code to disable HTTPS and use Application Insights

И добавить две строчки, чтобы включить профилировщик, зарегистрировав его вместе с Application Insights.

|  |
| --- |
| builder.Services.AddApplicationInsightsTelemetry();  builder.Services.AddServiceProfiler(); |

Далее в appsettings.json необходимо добавить секцию ApplicationInsights с InstrumentationKey, скопированным ранее.

|  |
| --- |
| "ApplicationInsights":  {  "InstrumentationKey": "instrumentation-key"  }, |

Далее запустим созданное приложение (рисунок 7) и протестируем API локально (рисунок 8).

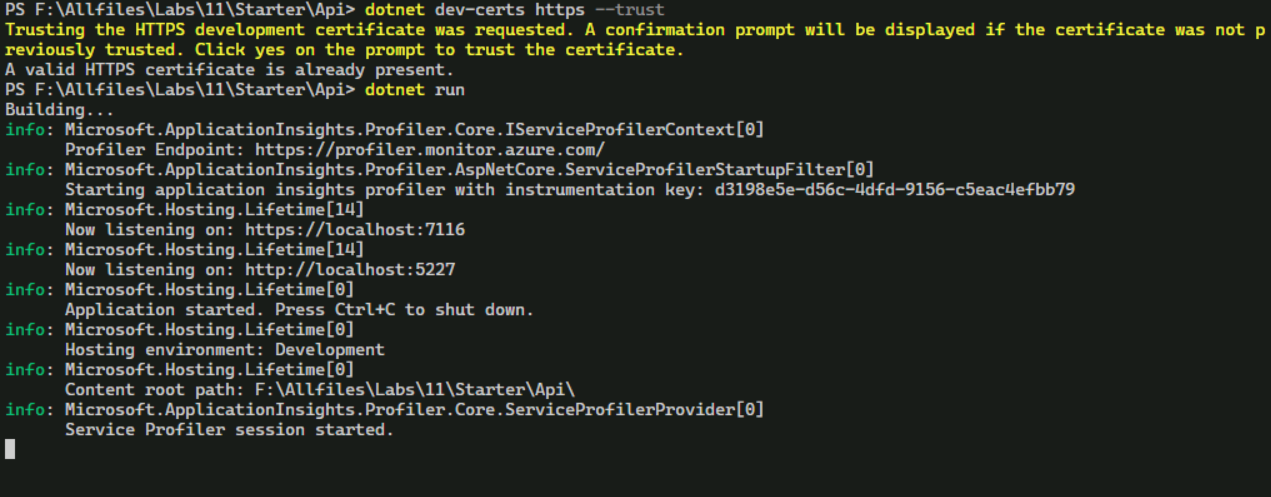


Рисунок 7 – Запуск Web API

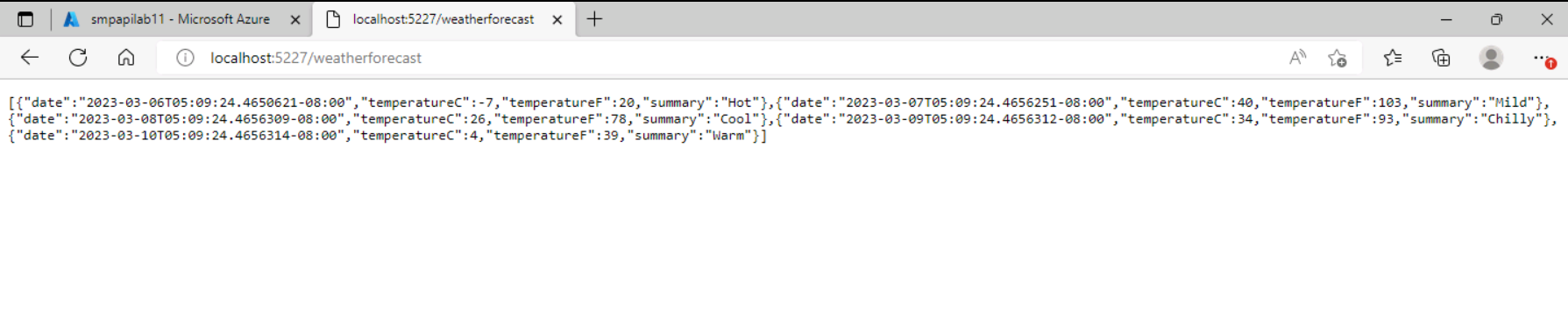


Рисунок 8 – Тестирование Web API локально

В Application Insights можно найти отображаемые метрики. В частности, количество запросов к серверу и среднее время ответа сервера.

На рисунке 9 показаны метрики Application Insights для локального веб-приложения.

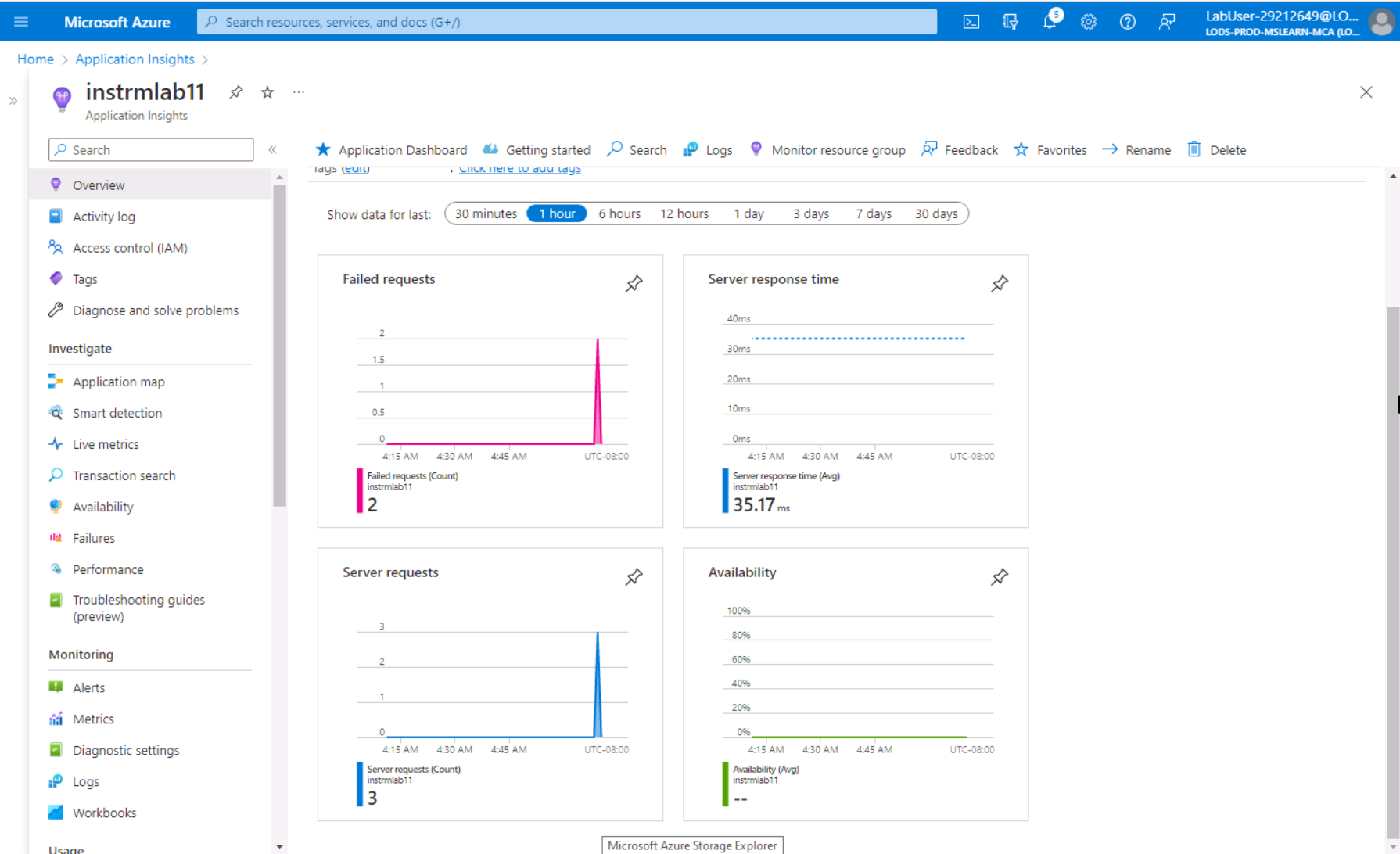


Рисунок 9 – Метрики Application Insights для локального веб-приложения

В результате было создано Web API и настроено для потоковой передачи метрик приложения в Application Insights. Затем была использована панель мониторинга Application Insights для просмотра сведений о производительности API.

**3 Мониторинг web API с использование Application Insights**

Далее опубликуем приложение. Для этого создадим файл web.config. Содержимое web.config представлено в листинге ниже.

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <configuration>  <location path="." inheritInChildApplications="false">  <system.webServer>  <handlers>  <add name="aspNetCore" path="\*" verb="\*" modules="AspNetCoreModuleV2" resourceType="Unspecified" />  </handlers>  <aspNetCore processPath="dotnet" arguments=".\SimpleApi.dll" stdoutLogEnabled="false" stdoutLogFile=".\logs\stdout" hostingModel="inprocess" />  </system.webServer>  </location>  </configuration> |

Далее создадим файл ZIP (рисунок 10), содержащий стартовый проект, который будет опубликован в Azure web API.

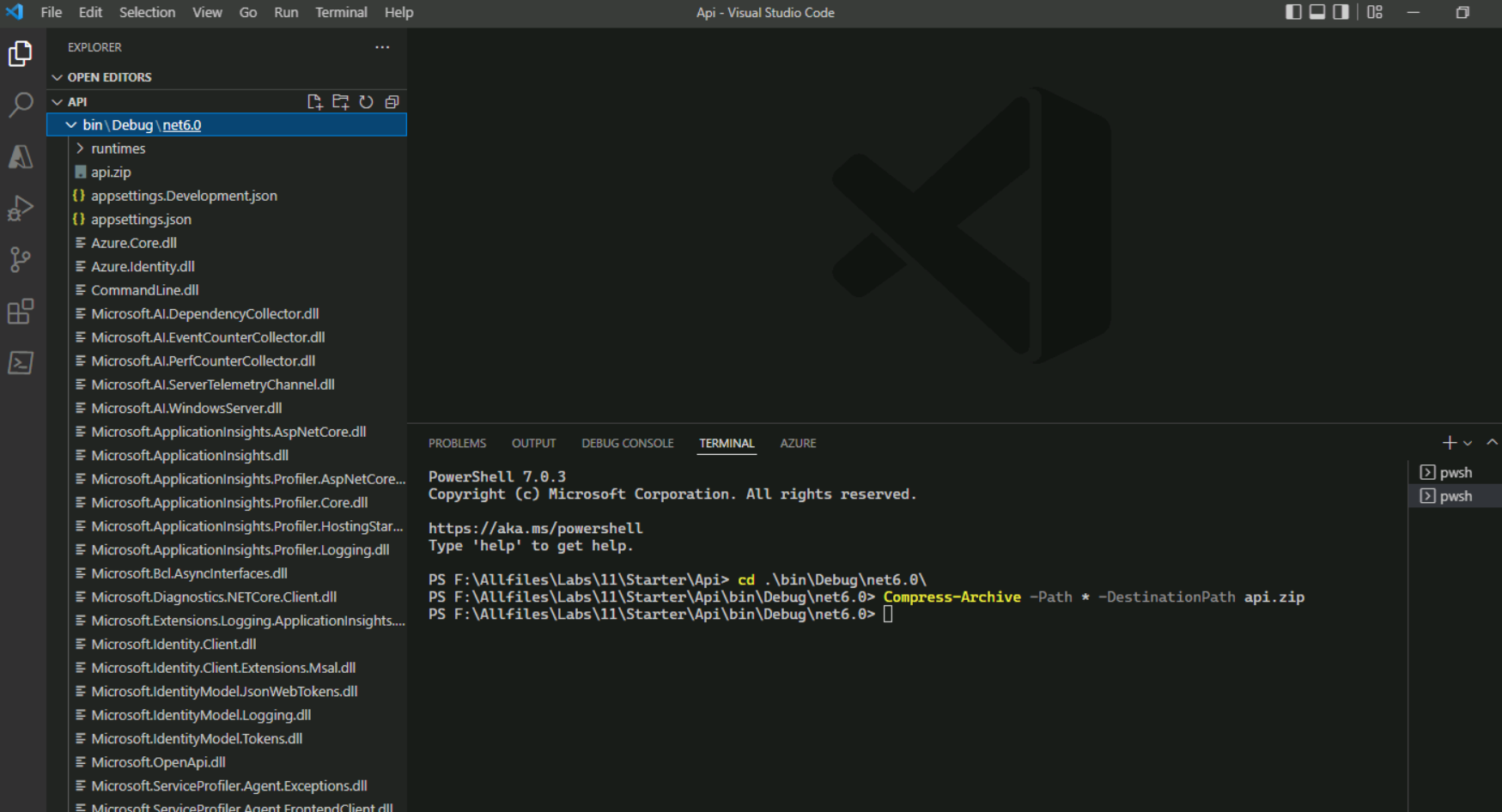
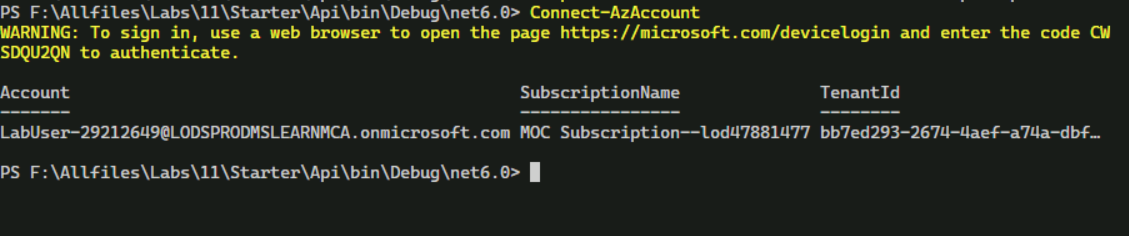


Рисунок 10 – Архивация проекта для дальнейшей публикации в Azure web API



Далее отобразим список все web apps в группе MonitoredAssets (рисунок 12).

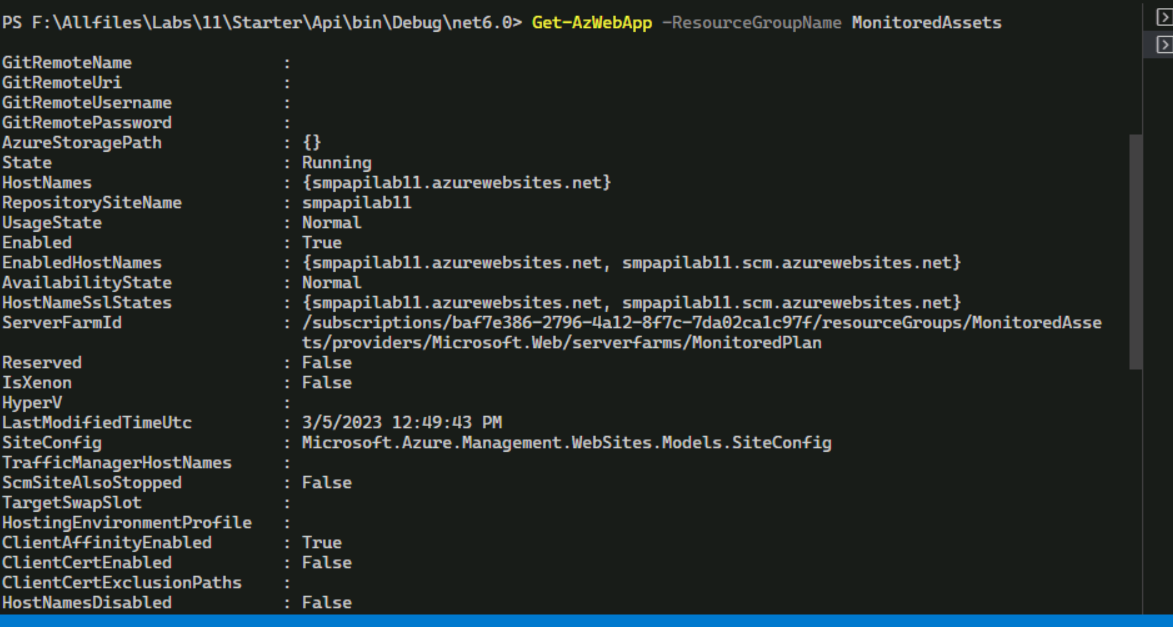


Рисунок 12 – Список всех Web Apps в группе MonitoredAssets

Выполним команды на рисунке 13, чтобы вывести список Web App в группе ресурсов MonitoredAssets, которые совпадают с smpapilab11.

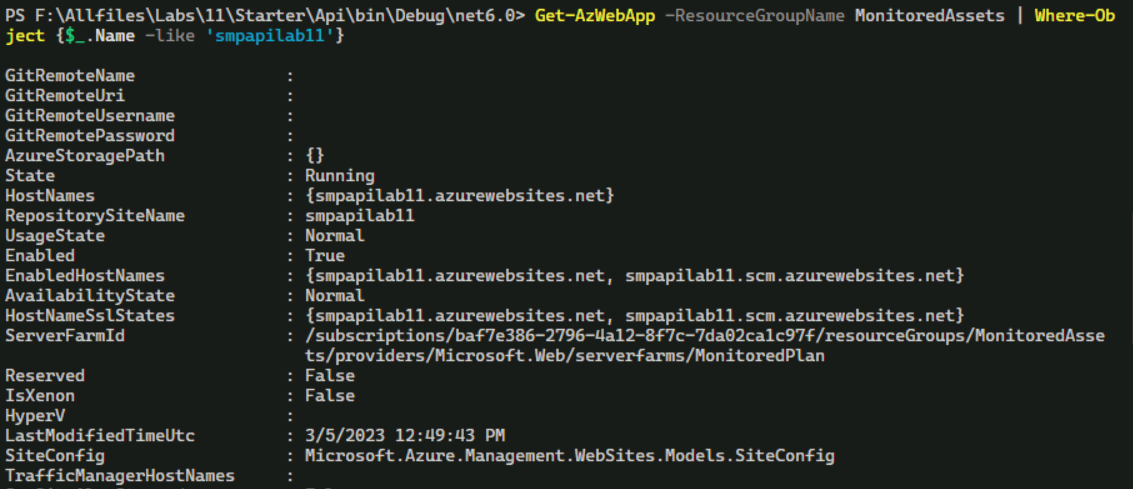


Рисунок 13 – Вывод списка Web Apps в группе MonitoredAssets по совпадению имени

Далее выполним команду (рисунок 14), которая отобразит название первого приложения из списка и сохраним его в переменную $webAppName.

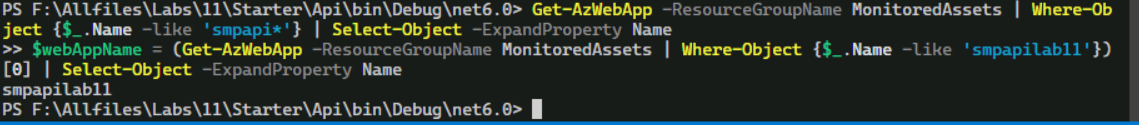


Рисунок 14 – Сохранение названия Web App в переменную

Далее опубликуем созданный архив в web API (рисунок 15), имя которой мы сохранили в переменную.

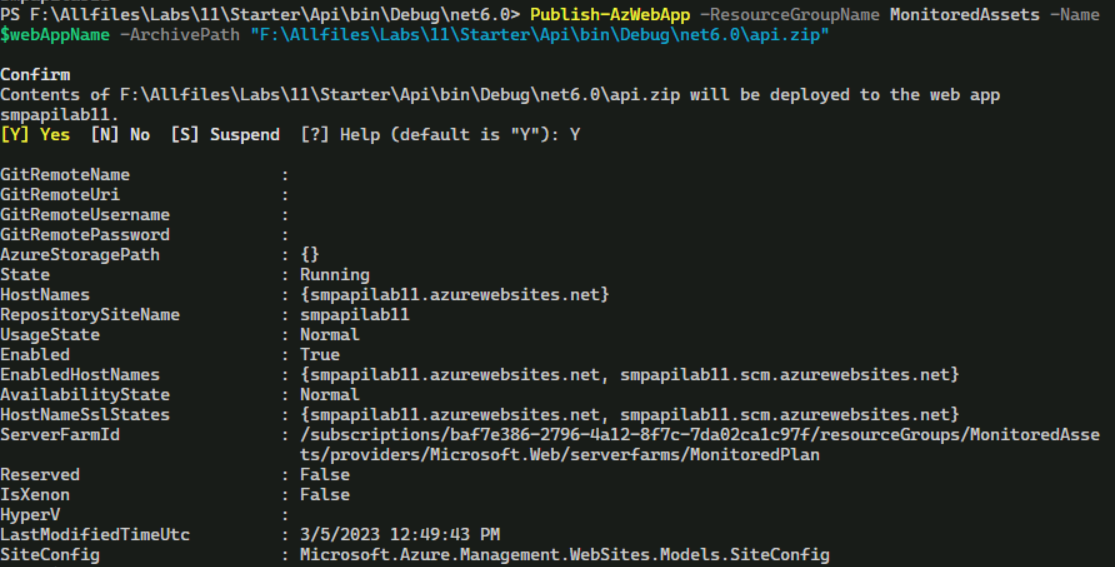


Рисунок 15 – Публикация архива

Далее проверим опубликованное приложение в браузере по скопированному ранее URL.



Рисунок 16 – Запрос к опубликованному Web API

Далее настроим коллекцию углубленных метрик для Web Apps (рисунок 17).

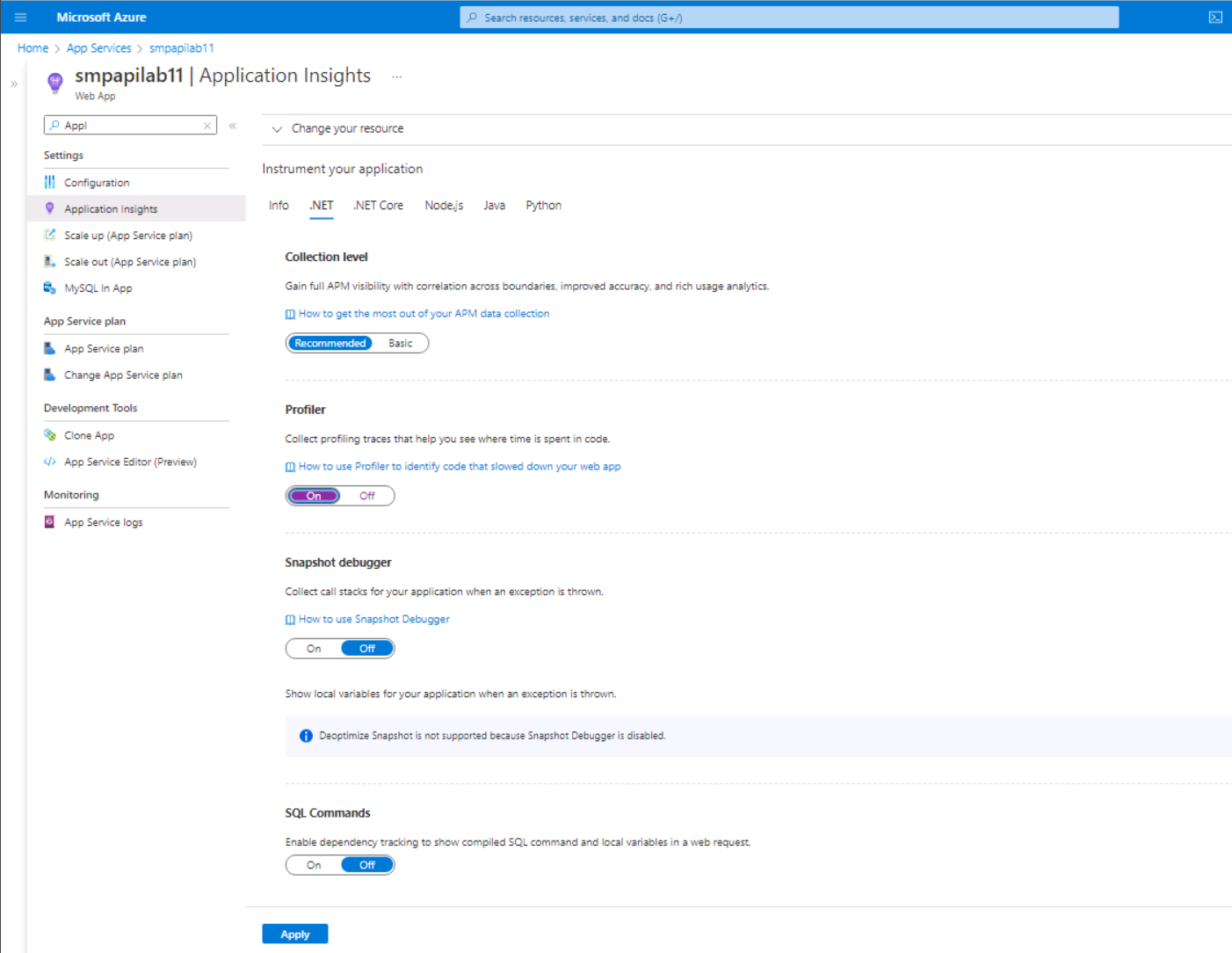


Рисунок 17 – Конфигурация метрик

После чего, сделаем повторно несколько запросов и посмотрим на результат.

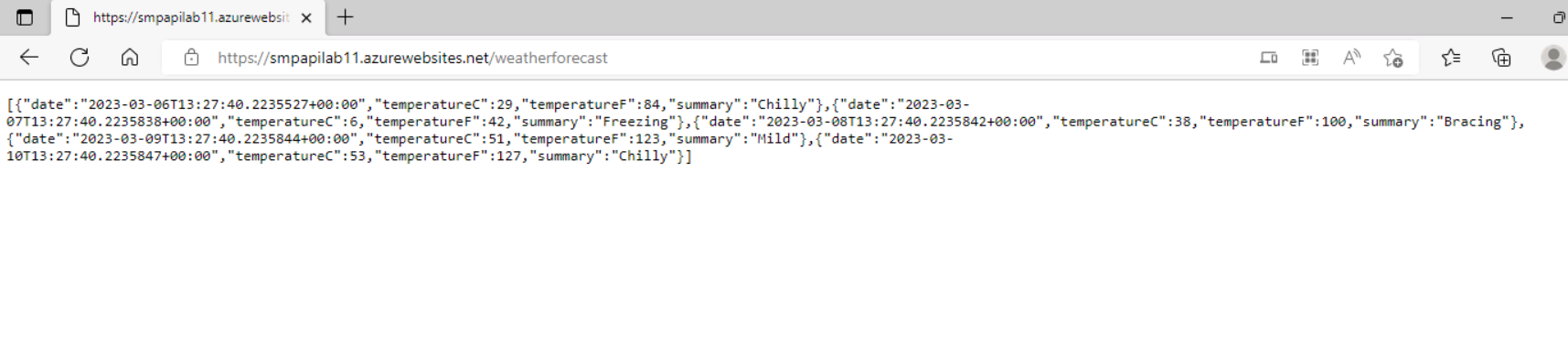


Рисунок 18 – Результат запроса после конфигурации метрик

Далее вернёмся в Application Insights и посмотрим метрики выполненных ранее запросов.

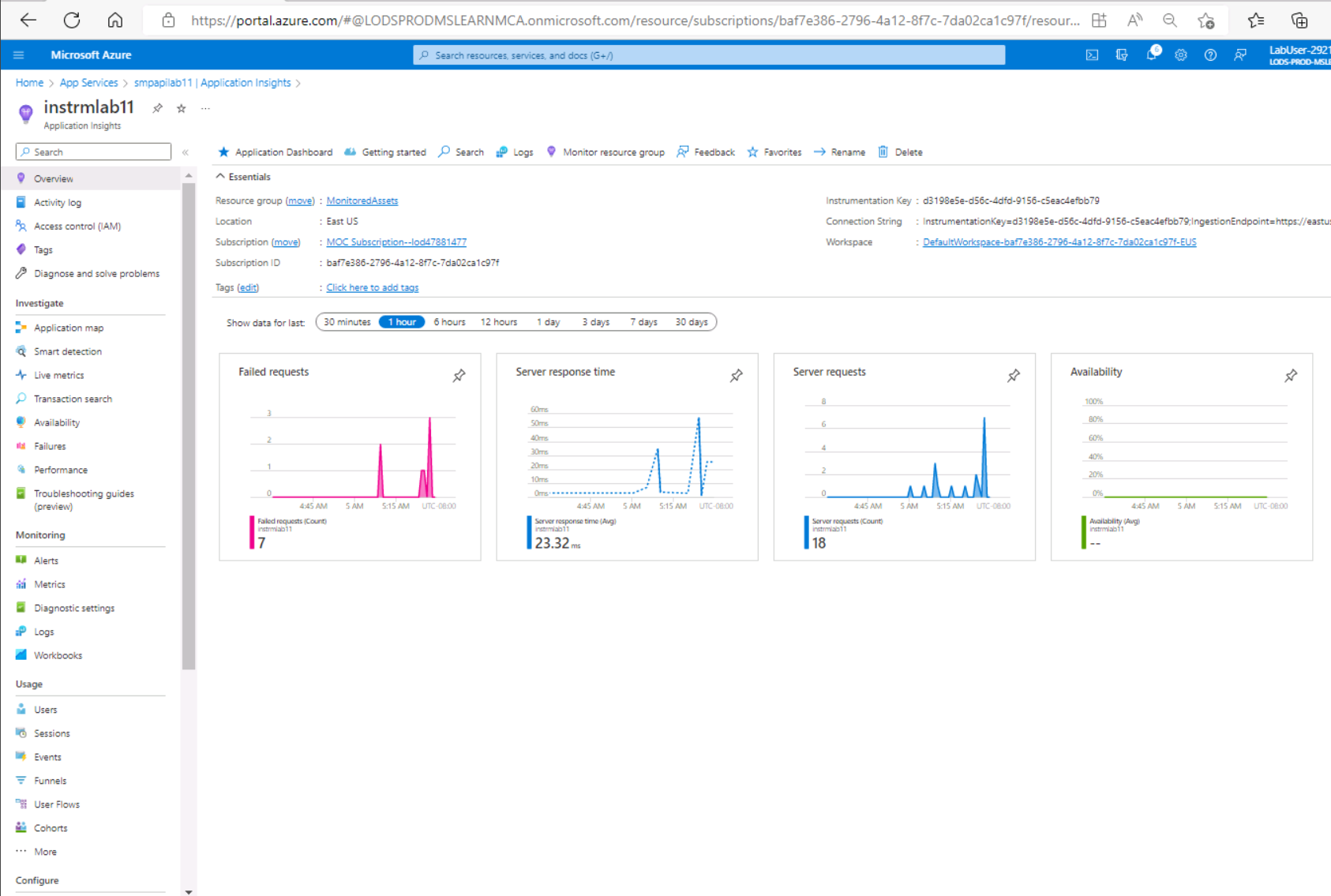


Рисунок 19 – Метрики выполненных запросов

Real-time метрики представлены на рисунке 20.

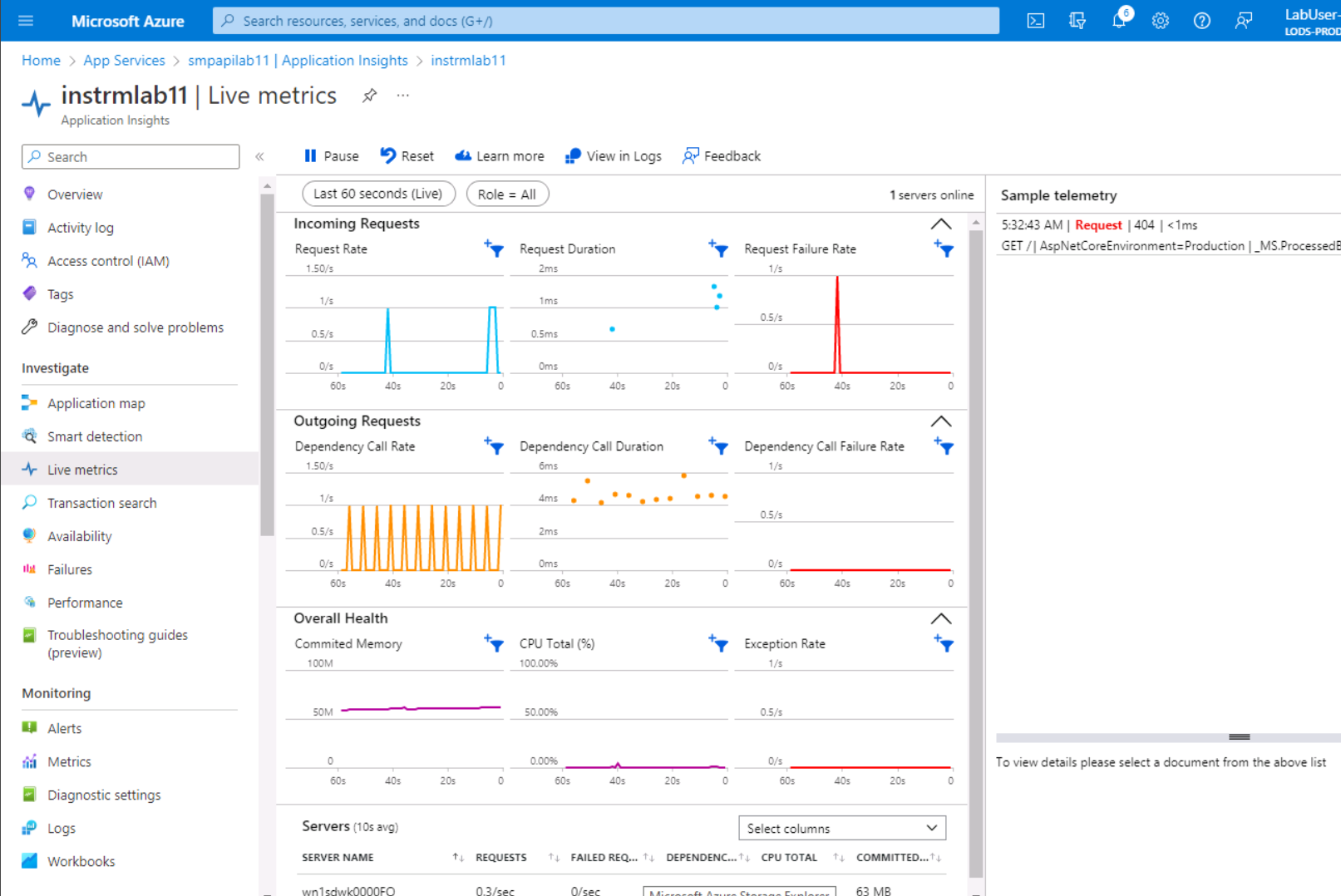


Рисунок 20 – Real-time метрики

**4 Application Insights логи**

Сконфигурируем логирование для .NET Web Api. Необходимо изменить Controller для отображения одного прогноза. Для каждого прогноза будет создаваться информационная запись, предупреждение или ошибка, указывающая тип погоды. После того как эти логи будут записаны Application Insights в соответствующей рабочей области Log Analytics, каждой будет автоматически назначен уровень серьёзности 1, 2 или 3.

Код метода Get контроллера WeatherForecastController представлен в листинге ниже.

|  |
| --- |
| {  var rng = new Random();  int temperatureC = rng.Next(-20, 55);  var summaryId = rng.Next(Summaries.Length);  switch (summaryId)  {  case 0: case 9:  \_logger.LogError("WeatherForecast: extreme weather");  break;  case 1: case 2: case 7: case 8:  \_logger.LogWarning("WeatherForecast: severe weather");  break;  default:  \_logger.LogInformation("WeatherForecast: mild weather");  break;  }  return Enumerable.Range(1, 1).Select(index => new WeatherForecast  {  Date = DateTime.Now.AddDays(index),  TemperatureC = temperatureC,  Summary = Summaries[summaryId]  })  .ToArray();  } |

Протестируем логирование локально (рисунок 21 и 22).

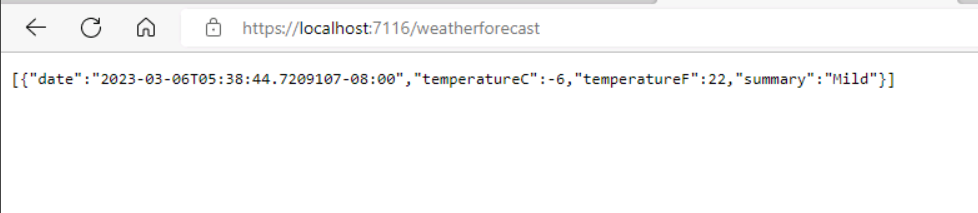


Рисунок 21 – Запрос к обновлённому endpoint weatherforecast

Каждое обновление страницы должно приводить к отображению информационных сообщений, предупреждений или сообщений об ошибках в терминале в следующем формате.

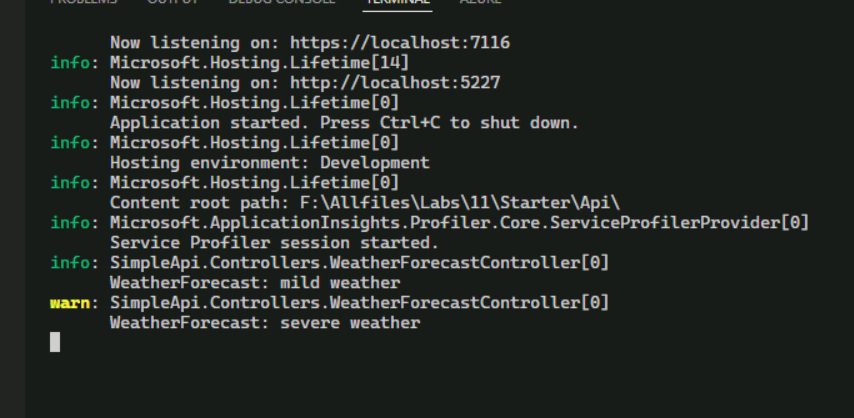


Рисунок 22 – Отображение логов в консосли

Далее посмотрим логи в Application Insights (рисунок 23).

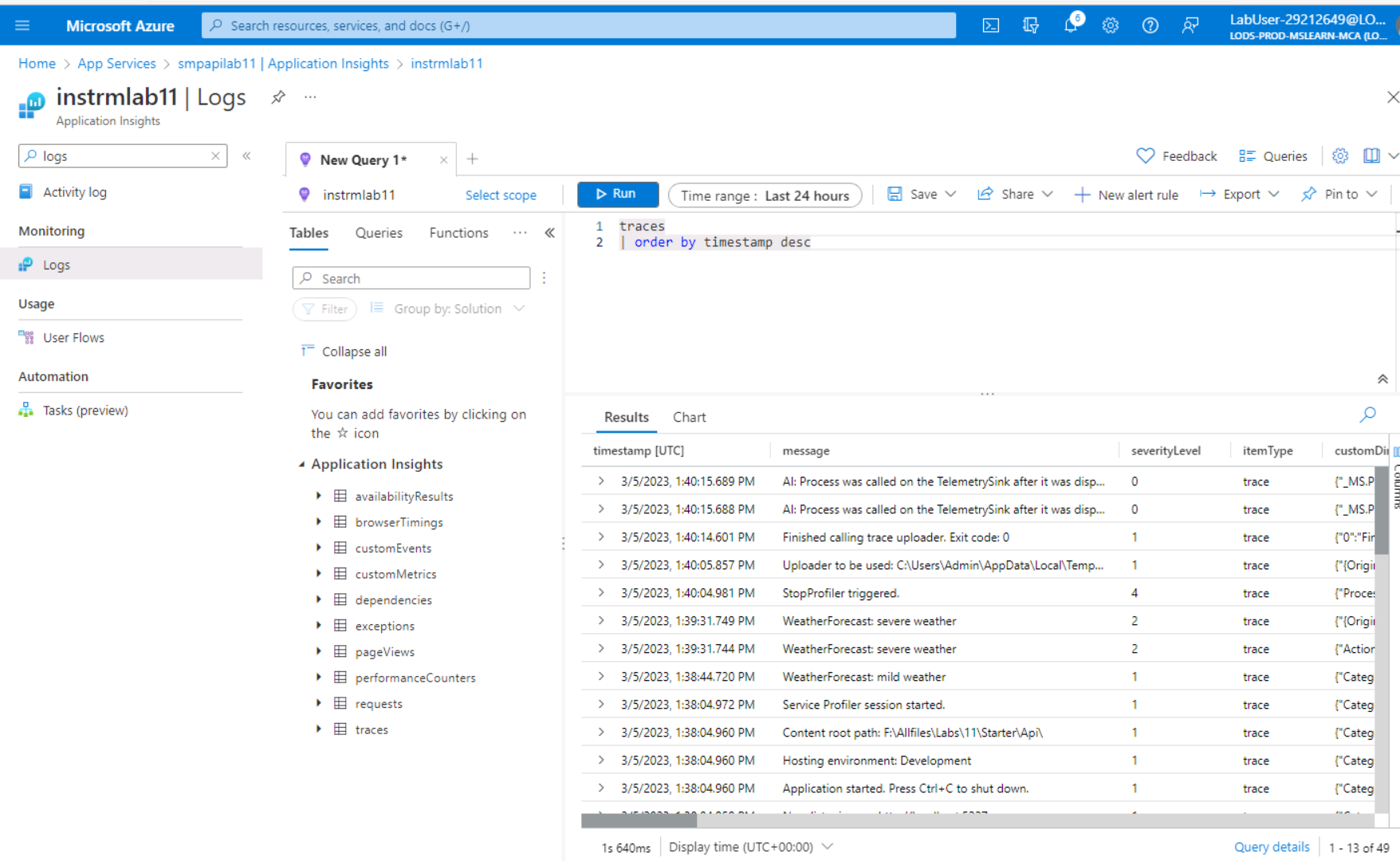


Рисунок 23 – Логи в Application Insights

Можно сделать более детальную настройку логов как в листинге ниже.

|  |
| --- |
| {  "Logging": {  "LogLevel": {  "Default": "Information",  "Microsoft": "Warning",  "Microsoft.Hosting.Lifetime": "Information"  },  "ApplicationInsights": {  "LogLevel": {  "Microsoft": "Information"  }  }  }  } |

**Вывод:** в результате выполнения лабораторной работы, я ознакомился с Azure Application Insights, настроил и протестировал Application Insights логирование.