

# Laboratorium Monitoring aplikacji w Azure

---

Damian Janas

<b>Założenia .....</b>	<b>2</b>
<b>Przygotowanie .....</b>	<b>2</b>
<b>Laboratorium .....</b>	<b>3</b>
I. Aktywacja Application Insights dla Web App.....	3
II. Instalacja paczki Application Insights do projektu ASP.NET Core .....	4
III. Konfiguracja server-side monitoringu w projekcie .....	5
IV. Konfiguracja client-side monitoringu w projekcie.....	5
V. Rozszerzenie aplikacji webowej o dodatkowe funkcjonalności .....	6
VI. Publikowanie aplikacji webowej jako Web App do chmury Azure .....	7
VII. Monitorowanie aplikacji za pomocą Live Metrics w Application Insights .....	9
VIII. Monitorowanie aplikacji przy pomocy Application Map w Application Insights .....	11
IX. Dodanie Function App i monitorowanie go w Application Insights .....	12
X. Analiza błędów przy wykorzystaniu monitoringu Application Insights .....	18
XI. Analiza logów przy użyciu Log Analytics (aka Azure Logs).....	24
XII. Definiowanie alertów przy użyciu Application Insights .....	25

## Założenia

- Student posiada aktywną subskrypcję Azure
- Student posiada zainstalowane Visual Studio 2022 na swojej maszynie
- Student posiada zainstalowany .NET Core min. 3.1 (może być .NET 7)

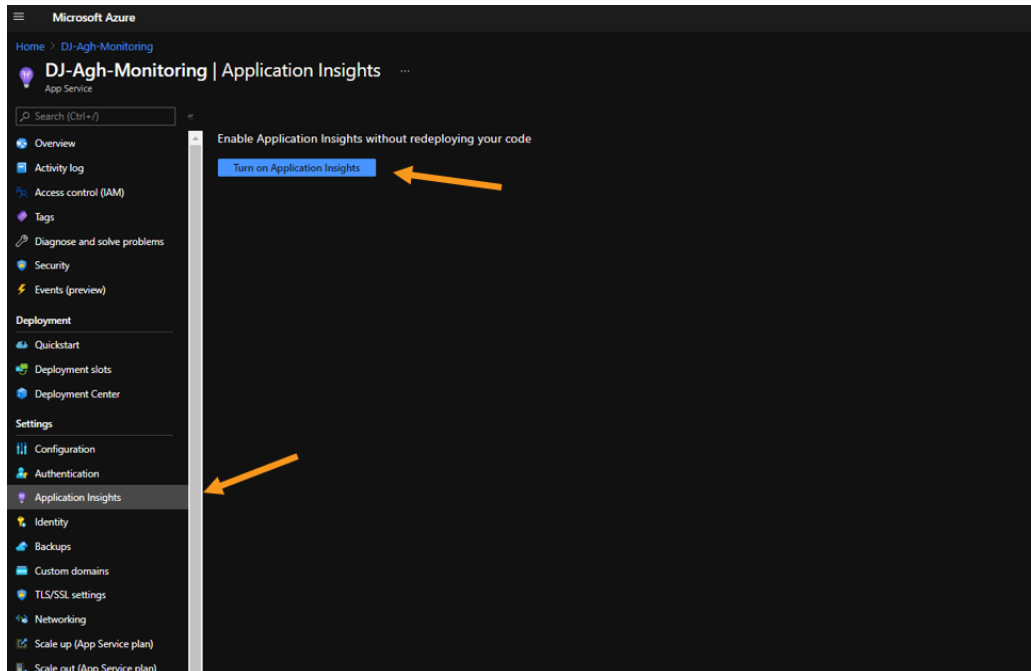
## Przygotowanie

1. Utworzenie nowego *Web App* w Azure Portal (laboratoria nr 2)
2. Utworzenie nowego projektu *ASP.NET Core Web Application (MVC)* w Visual Studio  
*Visual Studio -> File -> New -> Project*
3. Zweryfikowanie kompatybilności wersji .NET projektu oraz Web App:  
*Azure Portal -> Web App -> Configuration -> General Settings -> .NET version*
4. Opublikowanie utworzonego projektu jako Web App do chmury Azure (instrukcja w ćw. VI):  
PPM na projekcie w *Visual Studio -> Publish -> Azure -> Azure App Service*  
lub opcjonalnie publish profile
5. Więcej szczegółów w dokumencie z [Laboratorium nr 2 WebApp](#)

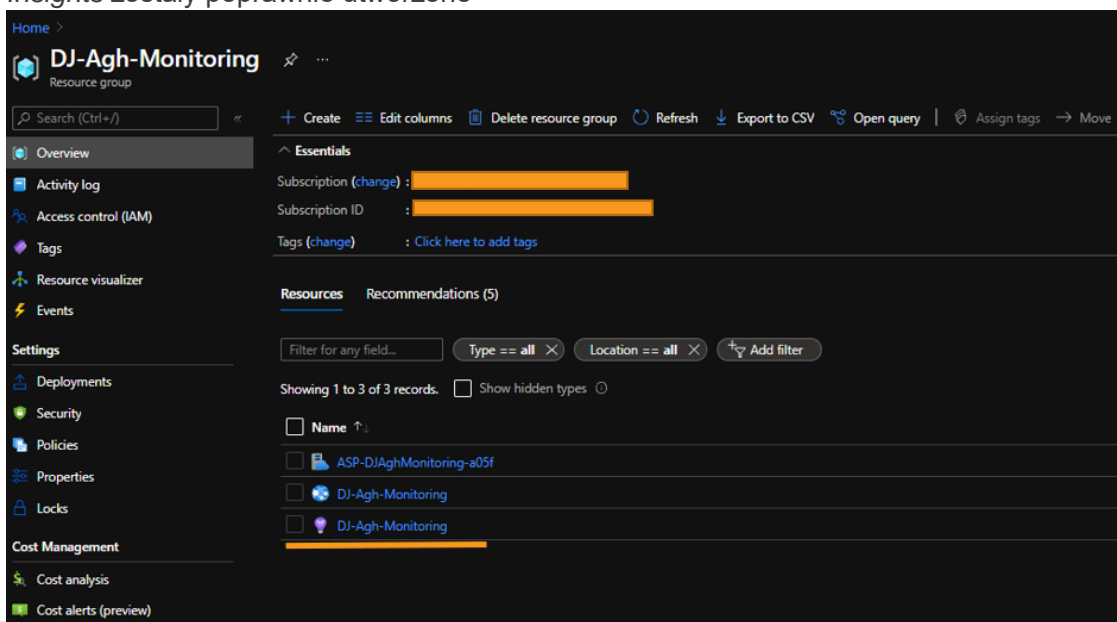
# Laboratorium

## I. Aktywacja Application Insights dla Web App

1. W Azure Portal, przejdź do nowo utworzonego Web App (*Przygotowanie p. 1*)
2. W lewym panelu, kliknij w Application Insights
3. Aktywuj Application Insights dla Web App poprzez kliknięcie w *Turn on Application Insights*

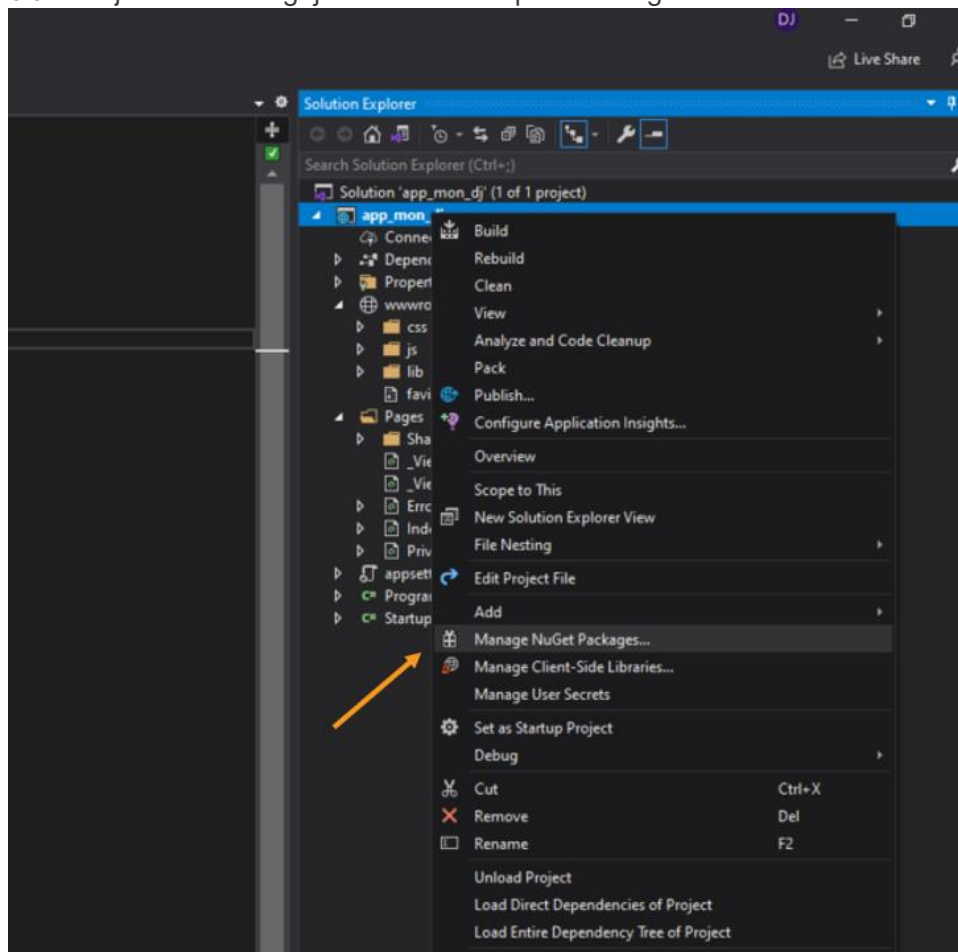


4. Zatwierdź zmiany przez kliknięcie w *Apply*
5. Zaczekaj na zaaplikowanie zmian
6. Wyszukaj wcześniej utworzoną resource grupę (*Przygotowanie p. 1*) i sprawdź czy *Application Insights* zostały poprawnie utworzone



## II. Instalacja paczki Application Insights do projektu ASP.NET Core

1. Otwórz utworzony project ASP.NET Core (*Przygotowanie p. 2*) w Visual Studio 2019/2022
2. Na projekcie kliknij PPM i znawiguj do menadżera paczek Nuget



3. W zakładce *Browse* wyszukaj [\*Microsoft.ApplicationInsights.AspNetCore\*](#)
4. Kliknij w przycisk *Install*, aby zainstalować go w projekcie

### III. Konfiguracja server-side monitoringu w projekcie

1. W Visual Studio, znawiguj do pliku *Startup.cs* w głównym katalogu projektu
2. W metodzie *ConfigureServices* dodaj następujący kod źródłowy, dzięki któremu nasza aplikacja będzie automatycznie wysyłała metryki do Application Insights:

```
23 // This method gets called by the runtime. Use this method to add services to the container.
24 // References
25 public void ConfigureServices(IServiceCollection services)
26 {
27     services.AddApplicationInsightsTelemetry();
28     services.AddControllersWithViews();
29 }
```

[Code Snippet](#)

### IV. Konfiguracja client-side monitoringu w projekcie

1. W Visual Studio, znawiguj do pliku *\_ViewImports.cshtml* w folderze *Views*
2. Dodaj następującą intrukcję *@inject*, która zdefiniuje alias (snippet) na kod JavaScriptowy z Application Insights (dzięki niemu nasza warstwa kliencka będzie automatycznie raportować metryki do Application Insights):

```
1 @using agh_monitoring_mvc
2 @using agh_monitoring_mvc.Models
3 @addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
4 @inject Microsoft.ApplicationInsights.AspNetCore.JavaScriptSnippet AppInsightsJsSnippet
5
```

[Code Snippet](#)

3. Przejdź do pliku *\_Layout.cshtml* w folderze *Views/Shared*
4. Wstrzyknij następującą intrukcję do head-a strony, która odpowiada za finalne wyrenderowanie wcześniej stworzonego aliasa:

```
1 @using Microsoft.ApplicationInsights.AspNetCore
2 <!DOCTYPE html>
3 <html lang="en">
4 <head>
5     <meta charset="utf-8" />
6     <meta name="viewport" content="width=device-width, initial-scale=1.0" />
7     <title>@ViewData["Title"] - agh_monitoring_mvc</title>
8     <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
9     <link rel="stylesheet" href="~/css/site.css" />
10    @Html.Raw(AppInsightsJsSnippet.FullScript)
11 </head>
12 <body>
13     <header>
14         <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light bg-white b
15             <div class="container">
16                 <a class="navbar-brand" asp-area="" asp-controller="Home" asp-action="In
17                 <button class="navbar-toggler" type="button" data-toggle="collapse" data
18                 aria-expanded="false" aria-label="Toggle navigation">
```

[Code Snippet](#)

## V. Rozszerzenie aplikacji webowej o dodatkowe funkcjonalności

W tym ćwiczeniu rozszerzymy naszą aplikację webową o dodatkowe funkcjonalności, aby ukazać szeroką gamę możliwości monitoringu w Application Insights.

1. W Visual Studio, znaviguj do pliku *HomeController.cs* w folderze *Controllers*
2. W powyższym pliku dodaj następującą metodę, dzięki której nasza aplikacja webowa udostępni dodatkowy endpoint:

```
39 [HttpGet]
40 0 references
41 public ActionResult<string> GetText()
42 {
43     return new OkObjectResult("Success!");
44 }
```

Code Snippet

3. W Visual Studio, znaviguj do pliku *Index.cshtml* w folderze *Views/Home*
4. We wspomnianym pliku dodaj następujący kod źródłowy, który wyrenderuje na stronie aplikacji przycisk. Naciśnięcie przycisku będzie powodowało wywołanie wcześniej dodanego endpointu (p. 2):

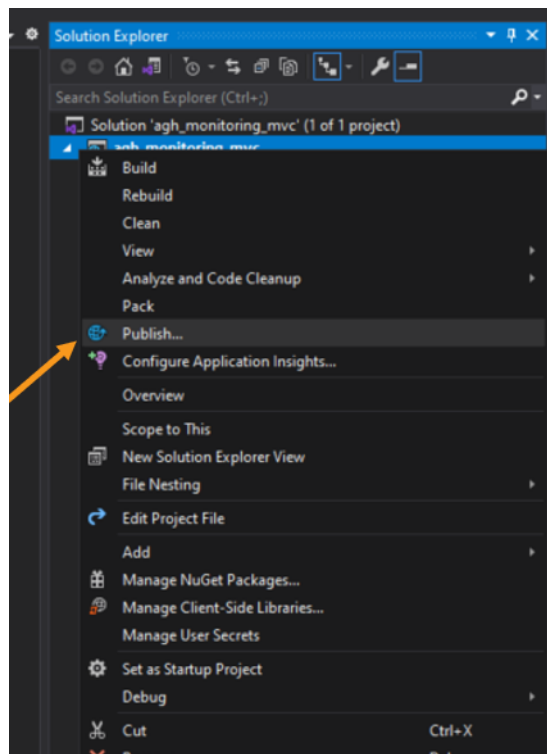
```
9 <div style="margin-bottom: 20px;">
10     <button id="button" class="btn btn-primary">Call API</button>
11 </div>
12 <div class="alert alert-primary" role="alert" id="message" style="display: none;"></div>
13 <div class="alert alert-danger" role="alert" id="error" style="display: none;"></div>
14 <script src="https://ajax.aspnetcdn.com/ajax/jquery/jquery-3.4.1.min.js"></script>
15 <script>
16     var url = "/Home/GetText";
17     $( "#button" ).click(function() {
18         $( "#message" ).text("Loading...");
19         $.get(url)
20             .done(function(data) {
21                 $( "#message" ).text(data).show();
22             })
23             .fail(function(error) {
24                 $( "#error" ).text(JSON.stringify(error)).show();
25             });
26     });
27 </script>
```

Code Snippet

## VI. Publikowanie aplikacji webowej jako Web App do chmury Azure

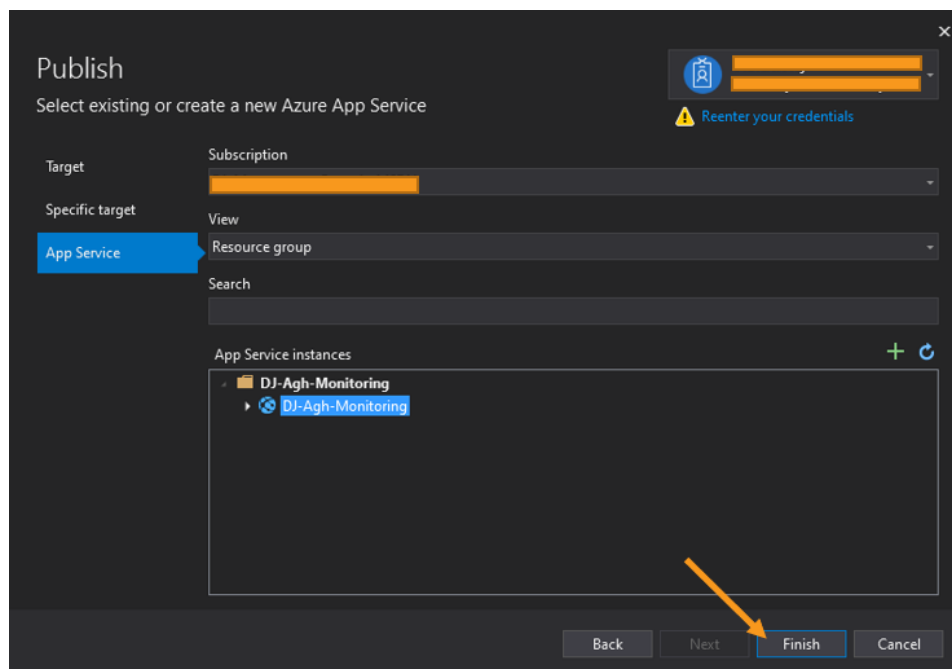
Na tym etapie nasz projekt jest skonfigurowany, aby raportować metryki zarówno z poziomu klienta (*ćw. IV*) jak i back-endu (*ćw. III*). Z kolei, Web App (*Przygotowanie*) w Azure Portal jest gotowy na opublikowanie naszej aplikacji.

1. W Visual Studio, w celu opublikowania aplikacji, klikamy PPM na nasz projekt i wybieramy *Publish*.

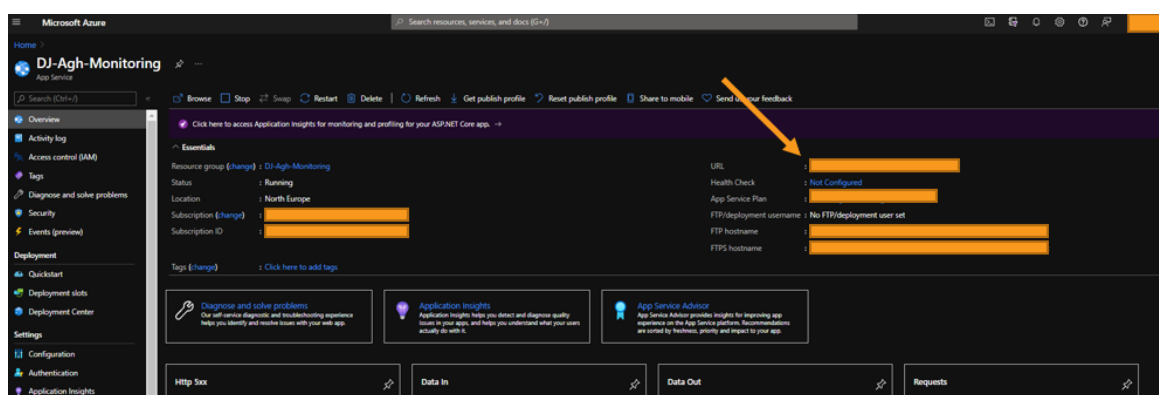


2. Następnie klikamy w Start -> Azure -> Azure App Service (Windows)
3. Logujemy się do Azure (jeśli wcześniej się nie logowaliśmy)

4. Wybieramy naszą subskrypcję oraz Web App, do którego będziemy publikować



5. Klikamy przycisk *Finish*
6. Po zamknięciu okna, wybieramy *Publish*
7. Po chwili oczekiwania aplikacja zostanie opublikowana i dostępna pod adresem URL, który można znaleźć w Web App (*Przygotowanie*) w Azure Portal

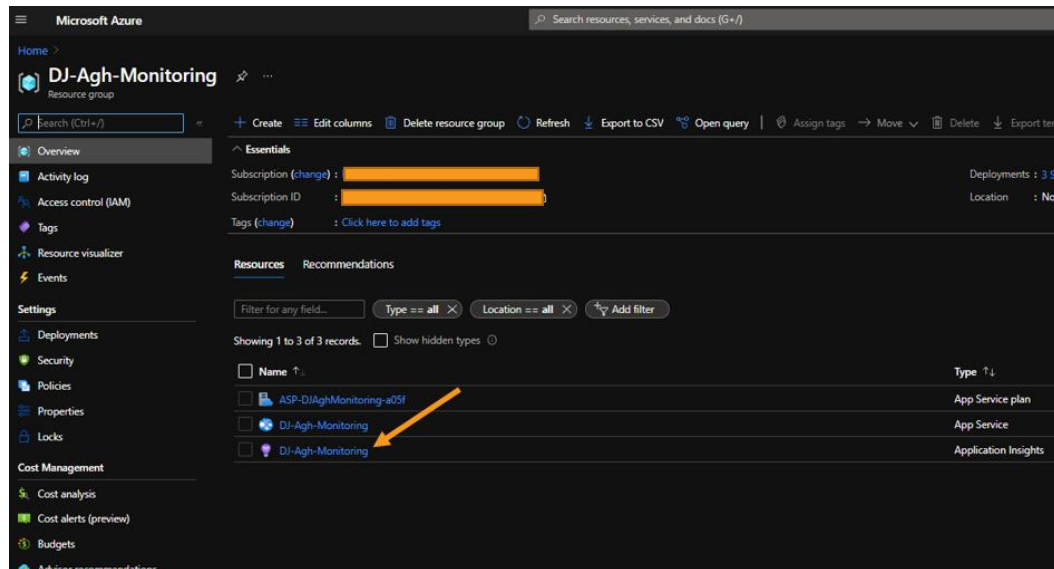


8. Sprawdź czy aplikacja poprawnie się ładuje w przeglądarce (**pamiętaj o wyłączeniu wszystkich wtyczek typu AdBlock**)

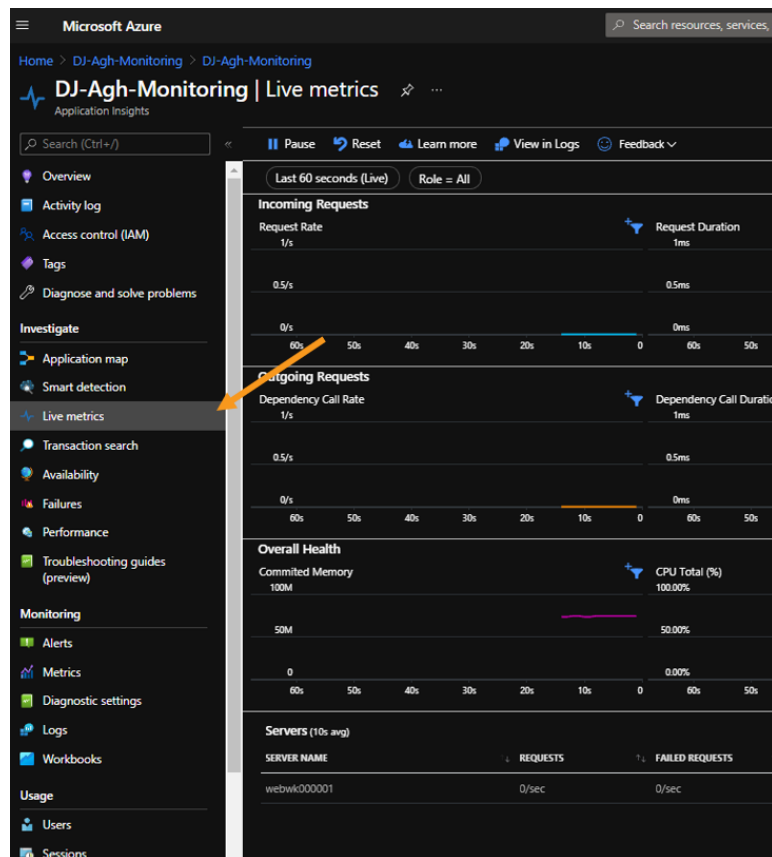


## VII. Monitorowanie aplikacji za pomocą Live Metrics w Application Insights

1. Znajdź resource group, w której znajduje się stworzony Web App (*Przygotowanie*) i wybierz wcześniej skonfigurowaną instancję Application Insights (*ćw. I*)



2. W lewym panelu kliknij w *Live Metrics*



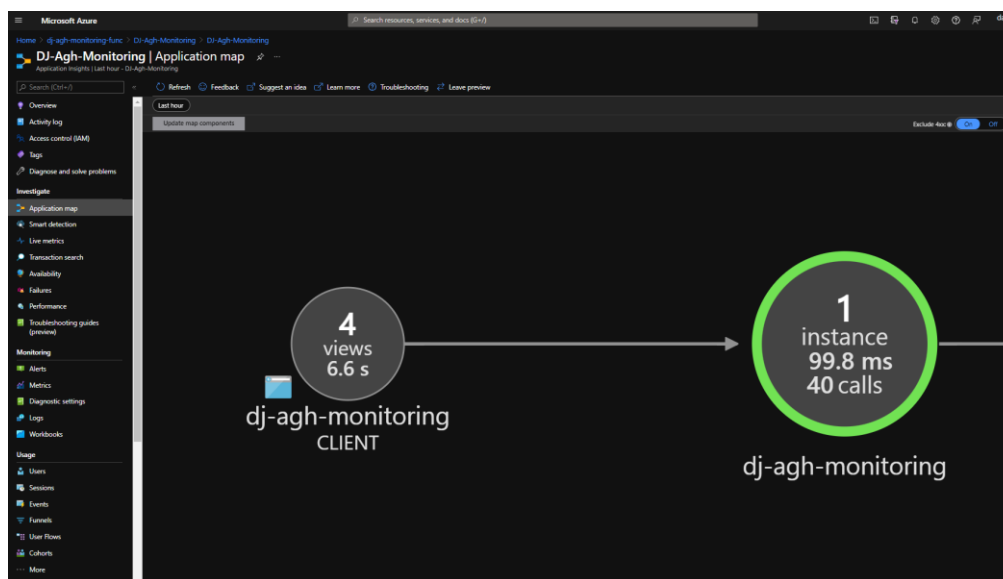
3. Następnie wykonaj kilka operacji na opublikowanej aplikacji (zmiana zakładek itp.) i prześledź w jaki sposób w czasie rzeczywistym metryki z twojej aplikacji są raportowane do Application Insights.

Zwróć szczególną uwagę na następujące metryki:

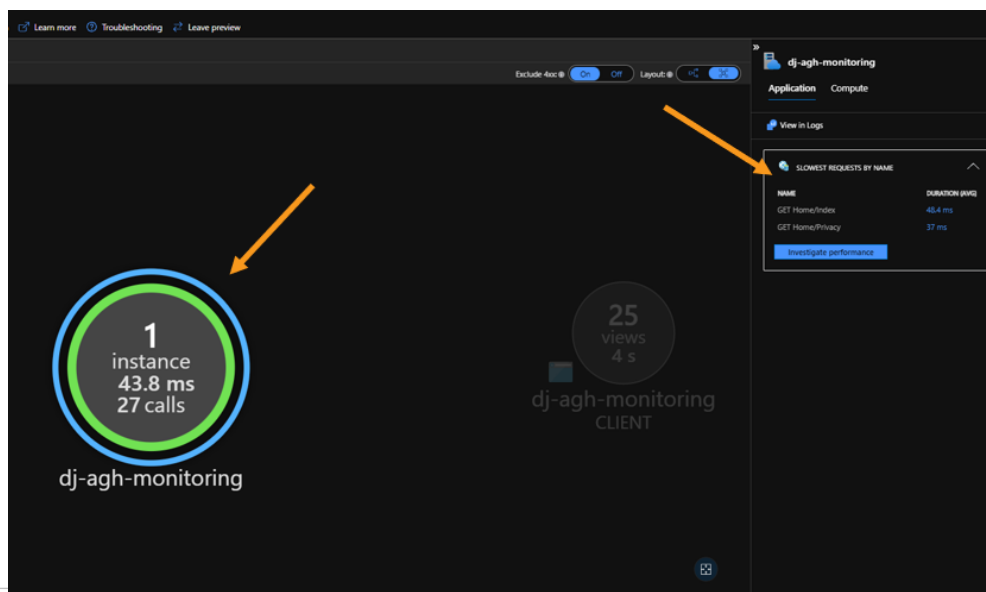
- a. Request Rate (ilość żądań na sekundę)
- b. Request Duration (średnia czasu trwania żądania)
- c. Committed Memory (zajętość pamięci RAM)
- d. CPU Total (procentowe obciążenie procesora)

## VIII. Monitorowanie aplikacji przy pomocy Application Map w Application Insights

1. Będąc w Application Insight, kliknij w zakładkę Application Map znajdującą się w lewym panelu Azure Portal
2. Jeśli Application Map się nie ładuje, sprawdź czy twoja aplikacja webowa jest włączona oraz wykonaj na niej kilka operacji (np. kliknięcie w zakładkę)
3. Przeanalizuj w jaki sposób twoja aplikacja została zmapowana do diagramu wyświetlonego w Application Map. Na tym etapie, powinieneś uzyskać dwa komponenty, które odpowiadają odpowiednio warstwie klienckiej i warstwie back-endu:



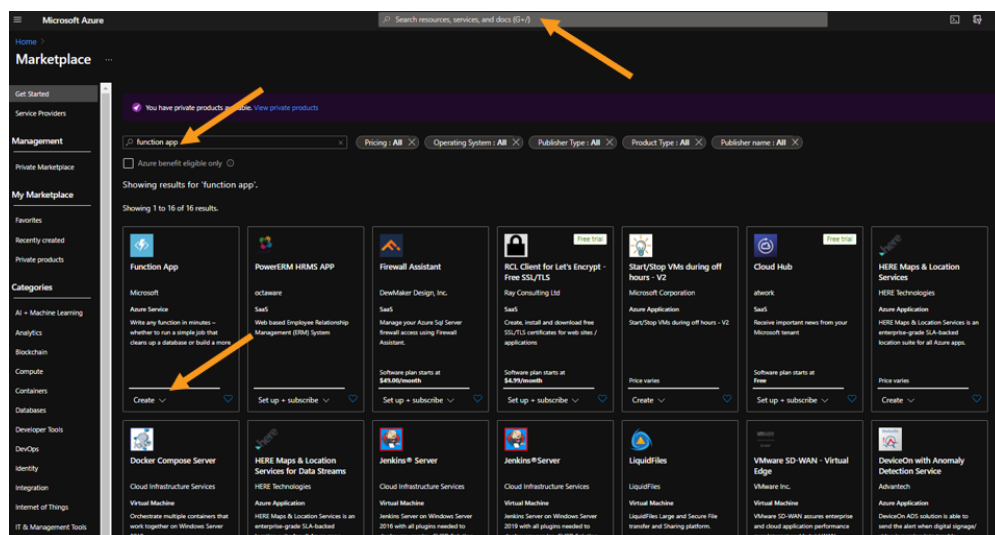
4. Zauważ jakie informacje są dostępne na diagramie:
  - a. Ilość dostępnych instancji aplikacji
  - b. Średni czas odpowiedzi
  - c. Ilość zapytań
5. Klikając w poszczególne komponenty sprawdź w jaki sposób Application Map dostarcza nam szczegółowych danych nt. wykonanych zapytań



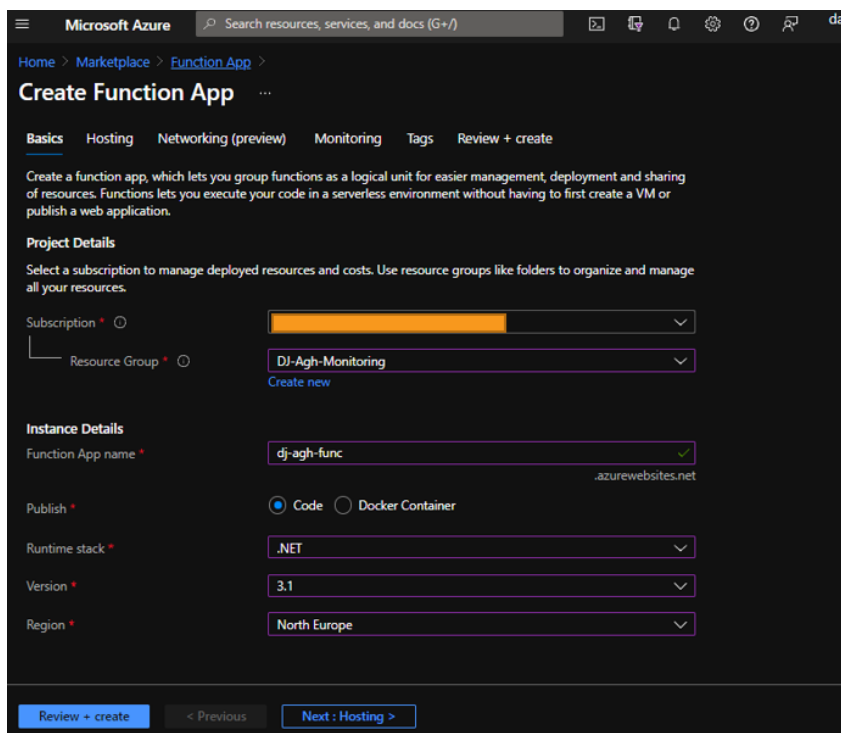
## IX. Dodanie Function App i monitorowanie go w Application Insights

Function App to nic innego jak usługa oferująca płatną moc obliczeniową w chmurze. W tym ćwiczeniu utworzymy Function App, skonfigurujemy go tak aby wysyłał metryki do Application Insights i integrujemy go z wcześniej utworzoną aplikacją webową.

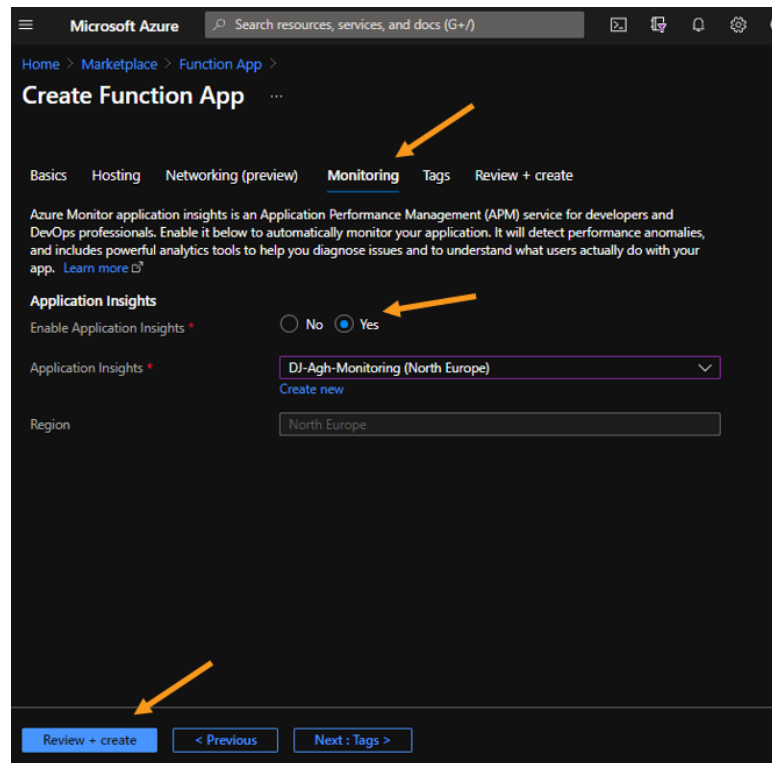
1. W Azure Portal, w głównym polu wyszukiwania, wyszukaj *Marketplace*
2. W *Marketplace*, wyszukaj usługi *Function App*



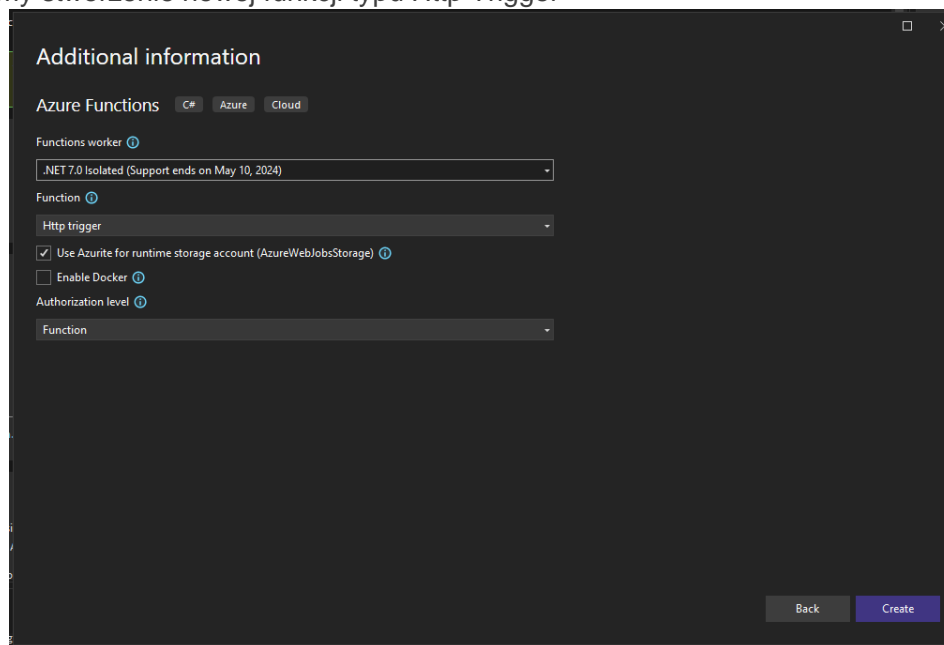
3. Klikamy w *Function App* i wybieramy przycisk *Create*
4. Konfigurujemy *Function App* w następujący sposób (nie tworzymy nowej resource grupy, tylko korzystamy z tej, w której znajduje się nasz Web App):



5. W zakładce *Monitoring* aktywujemy Application Insights dla Function App oraz wybieramy wcześniej utworzoną instancję Application Insights (ta sama, która była użyta dla Web App):



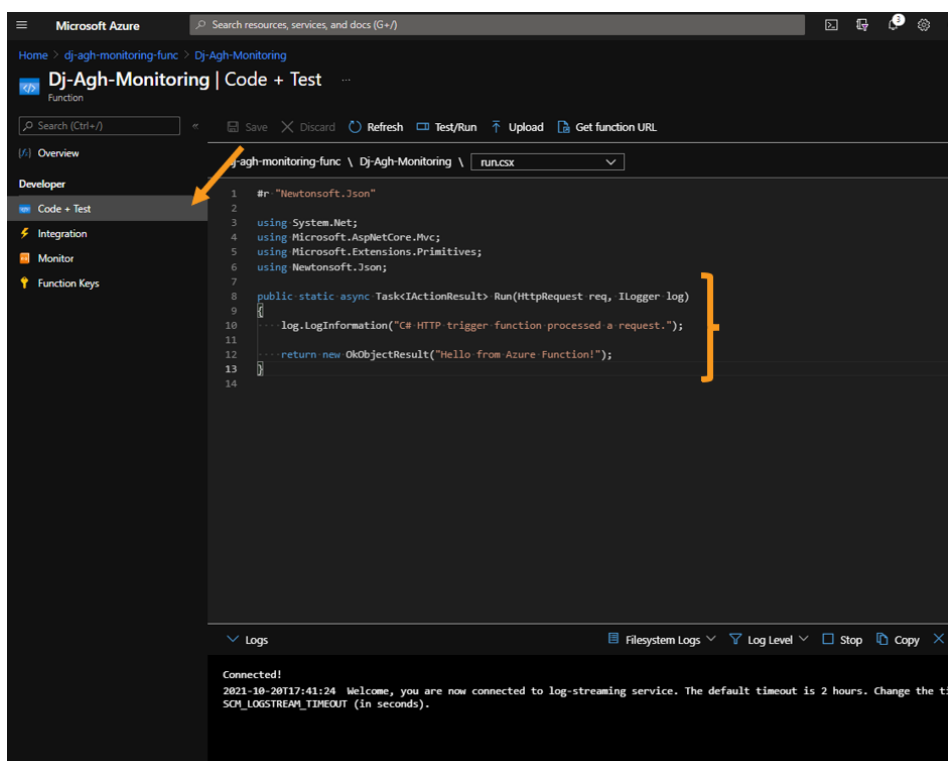
6. Zatwierdzamy zmiany przyciskiem *Review + create*
7. Klikamy przycisk *Create*
8. Czekamy na utworzenie nowych zasobów (Function App)
9. Jeśli zasoby już są gotowe, klikamy w przycisk *Go to Resource*
10. W Visual Studio, tworzymy nowy projekt z szablonu *Azure Functions*
11. Zatwierdzamy stworzenie nowej funkcji typu Http Trigger



12. Klikamy PPM na stworzony projekt w Visual Studio > Publish
13. Logujemy się do Azure (jeśli tego nie zrobiliśmy wcześniej)
14. Tworzymy domyślny profil deploymentu (wybieramy funkcję, którą wcześniej stworzyliśmy w portalu Azure)
15. Publikujemy funkcję z poziomu Visual Studio
16. Czekamy na pomyślne zakończenie publikowania funkcji
17. Przenosimy się do portalu Azure do stworzonej wcześniej funkcji (p.3)
18. W głównym panelu funkcji powinna znaleźć się opublikowana przez nas funkcja – klikamy w nią



19. Nawigujemy do zakładki *Code + Test*
20. Jeśli w Azure Portal nie ma możliwości aktualizacji kodu źródłowego z kolejnego punktu – zmian dokonujemy z poziomu Visual Studio i następnie publikujemy funkcję na chmurę (tak jak w p. 12).
21. Modyfikujemy kod źródłowy funkcji w następujący sposób:

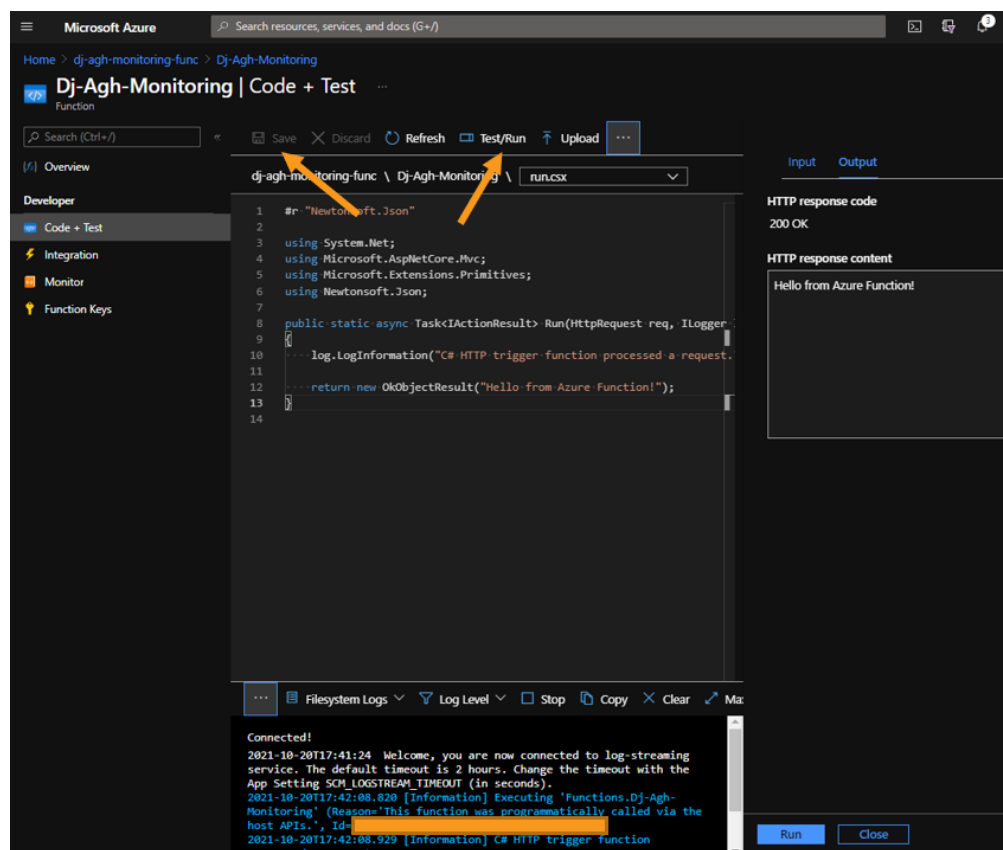


```
1 #r "Newtonsoft.Json"
2
3 using System.Net;
4 using Microsoft.AspNetCore.Mvc;
5 using Microsoft.Extensions.Primitives;
6 using Newtonsoft.Json;
7
8 public static async Task<IActionResult> Run(HttpRequest req, ILogger log)
9 {
10     log.LogInformation("C# HTTP trigger function processed a request.");
11
12     return new OkObjectResult("Hello from Azure Function!");
13 }
14
```

Code Snippet

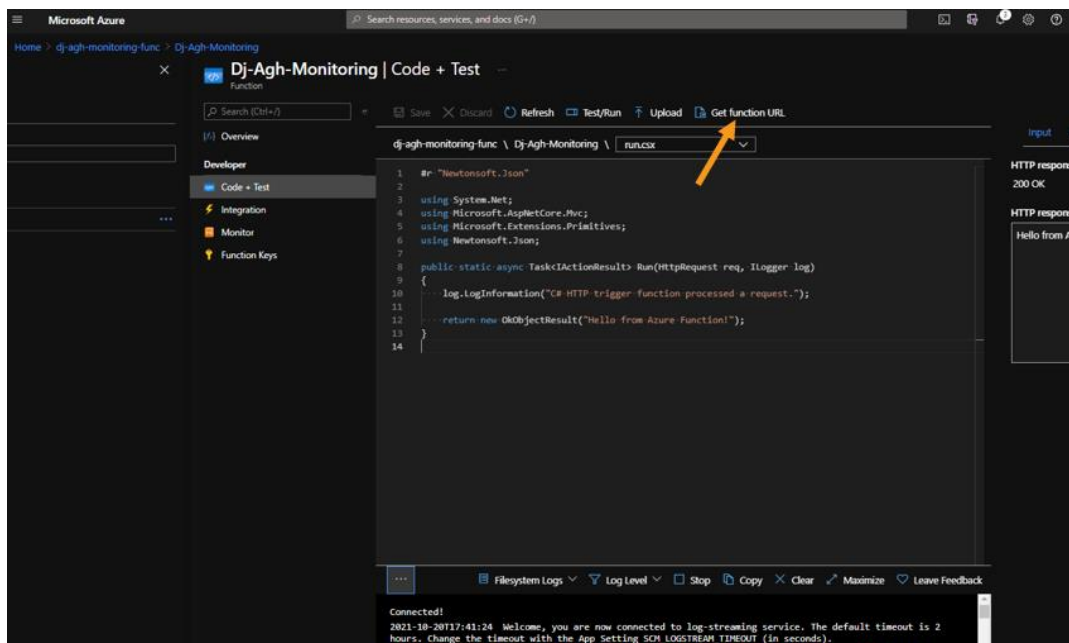
22. Zatwierdzamy zmiany przyciskiem Save

23. Klikając w przycisk *Test/Run* możemy zweryfikować czy funkcja zwraca poprawny rezultat:



24. W tym samym czasie możemy znawigować do wcześniej utworzonej instancji Application Insight i w zakładce *Live Metrics* zweryfikować czy nasza funkcja poprawnie raportuje metryki

25. Następnie klikamy w *Get function URL* w celu pobrania adresu funkcji Azure



26. W Visual Studio, nawigujemy do pliku *HomeController.cs* w folderze *Controllers*

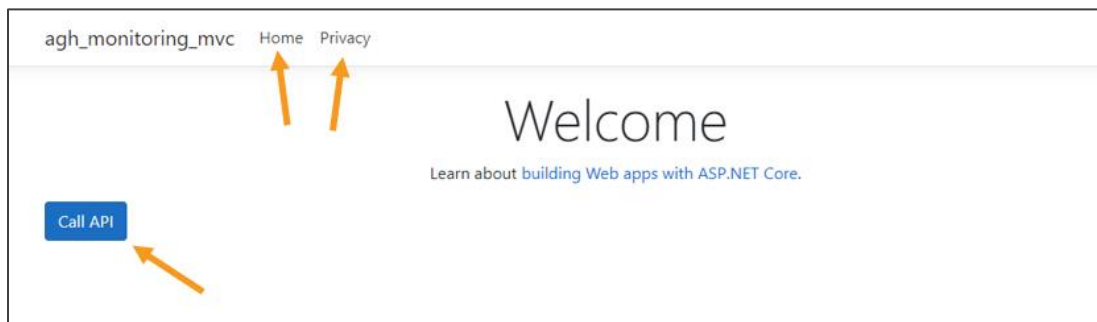
27. Modyfikujemy metodę *GetText* następującym kodem źródłowym (w miejsce *<func-url>* wklejamy wcześniej pobrany adres funkcji Azure (p. 20))

```
[HttpGet]
0 references
public async Task<ActionResult<string>> GetText()
{
    var functionUrl = "<func-url>";
    var client = new HttpClient();
    var response = await client.GetAsync(functionUrl);
    var text:string = await response.Content.ReadAsStringAsync();
    return new OkObjectResult(text);
}
```

Code Snippet

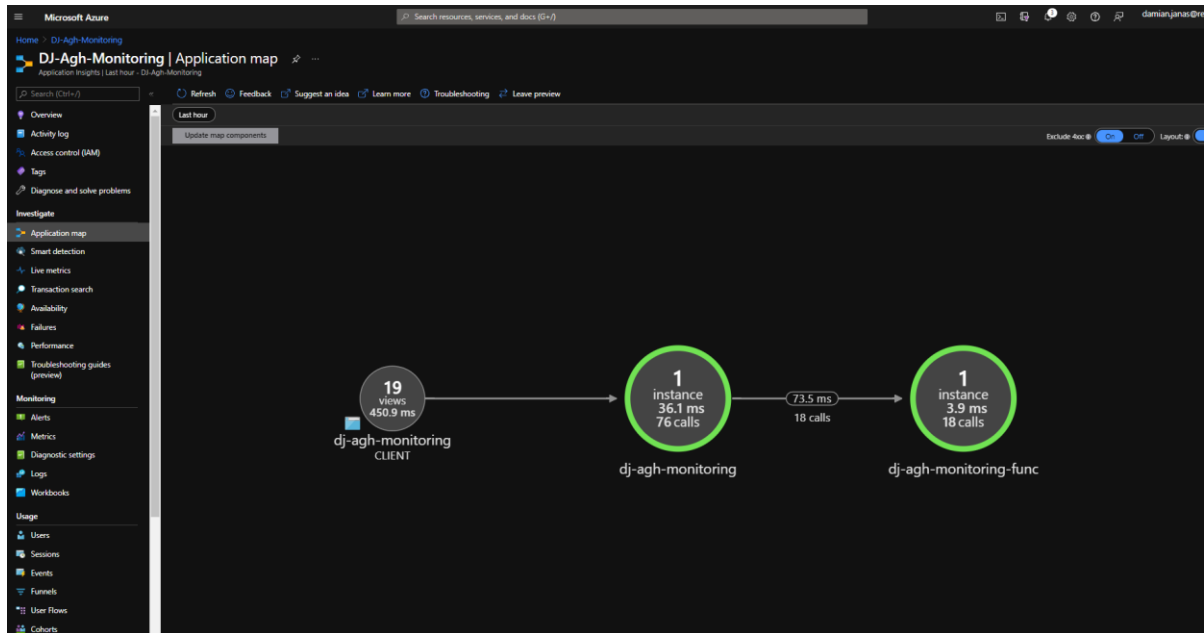
28. Zapisujemy zmiany i publikujemy naszą aplikację tak jak w *cw. VI*

29. Po opublikowaniu nasz aplikacja powinna wyglądać następująco:





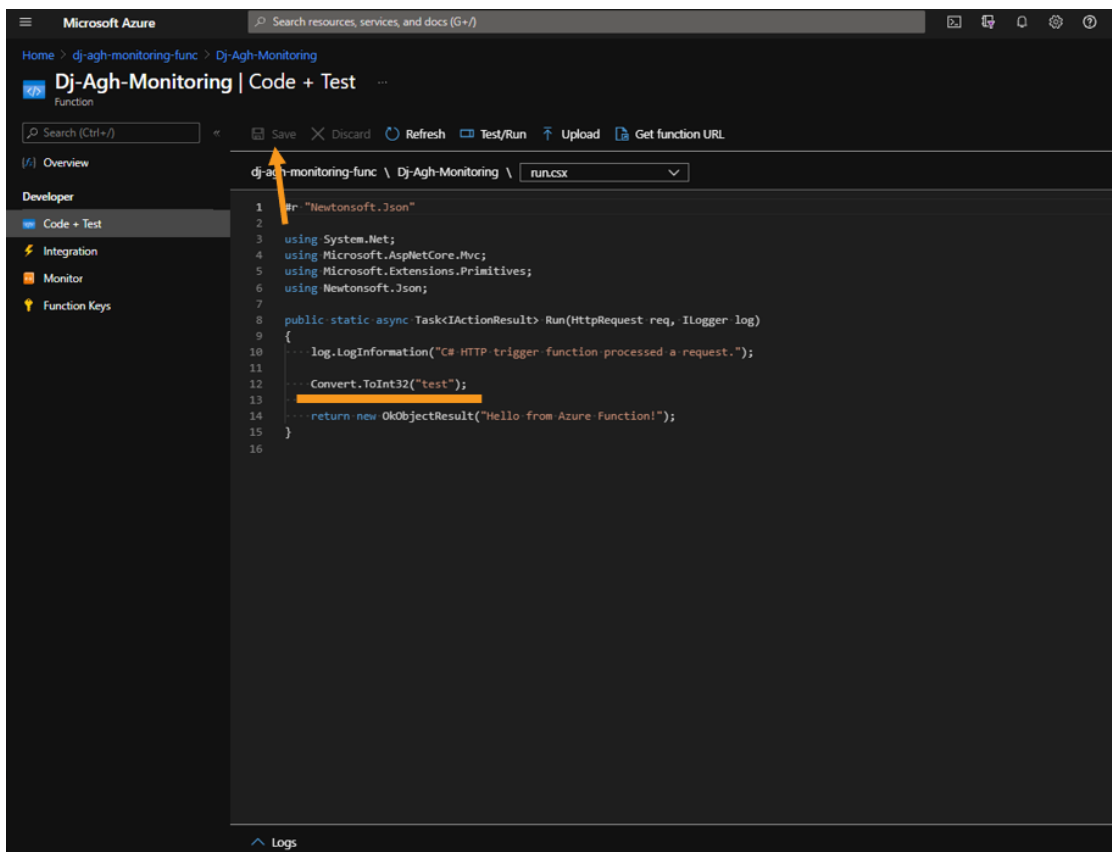
30. Wykonaj kilka operacji na aplikacji, tj. parokrotne kliknięcie w przycisk *Call API* oraz zmiana zakładek.
31. W Azure Portal, udaj się do instancji Application Insights -> *Application Map* i sprawdź w jaki sposób została zwizualizowana twoja aplikacja:



32. Zwróć uwagę w jaki sposób została zwizualizowana komunikacja między naszą aplikacją webową a funkcją Azure oraz jakie informacje możemy uzyskać dzięki monitoringowi, m.in. ilość wykonanych zapytań pod dany adres i średni czas odpowiedzi.

## X. Analiza błędów przy wykorzystaniu monitoringu Application Insights

1. Z nawiguj do funkcji Azure, która została stworzona w **ów. IX**.
2. Zmodyfikuj kod źródłowy funkcji tak, aby wyrzucał błąd. Na przykład w następujący sposób:

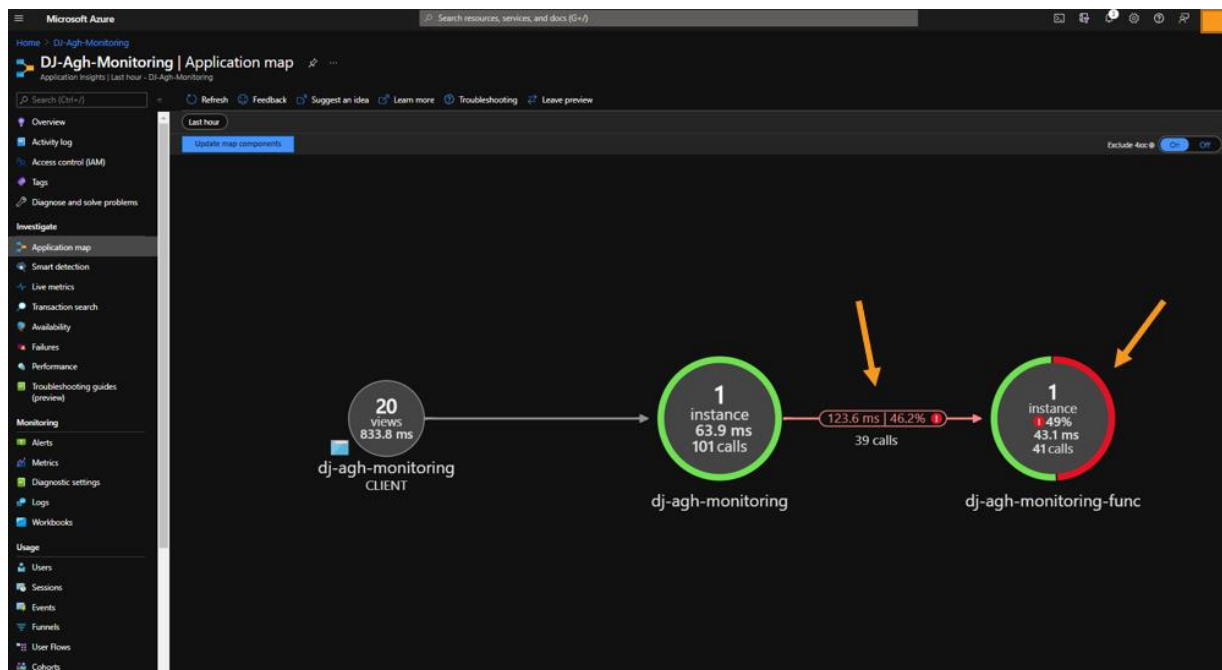


```
1 #r "Newtonsoft.Json"
2
3 using System.Net;
4 using Microsoft.AspNetCore.Mvc;
5 using Microsoft.Extensions.Primitives;
6 using Newtonsoft.Json;
7
8 public static async Task
```

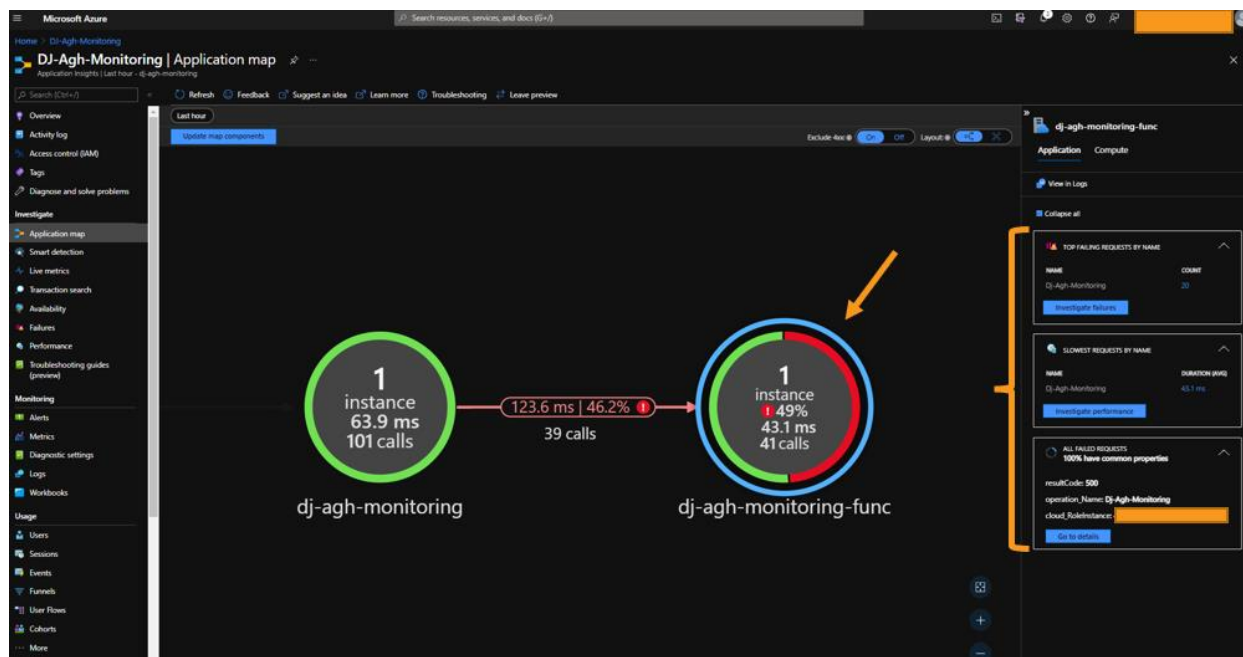
Code Snippet

3. Zapisz zmiany, klikając w przycisk **Save**
4. Otwórz opublikowaną aplikację webową i naciśnij kilkakrotnie w przycisk **Call API**
5. Ponownie znawiguj do Application Insights -> *Application Map* (generowanie/odświeżanie grafu może trwać od kilkunastu do kilkudziesięciu sekund)

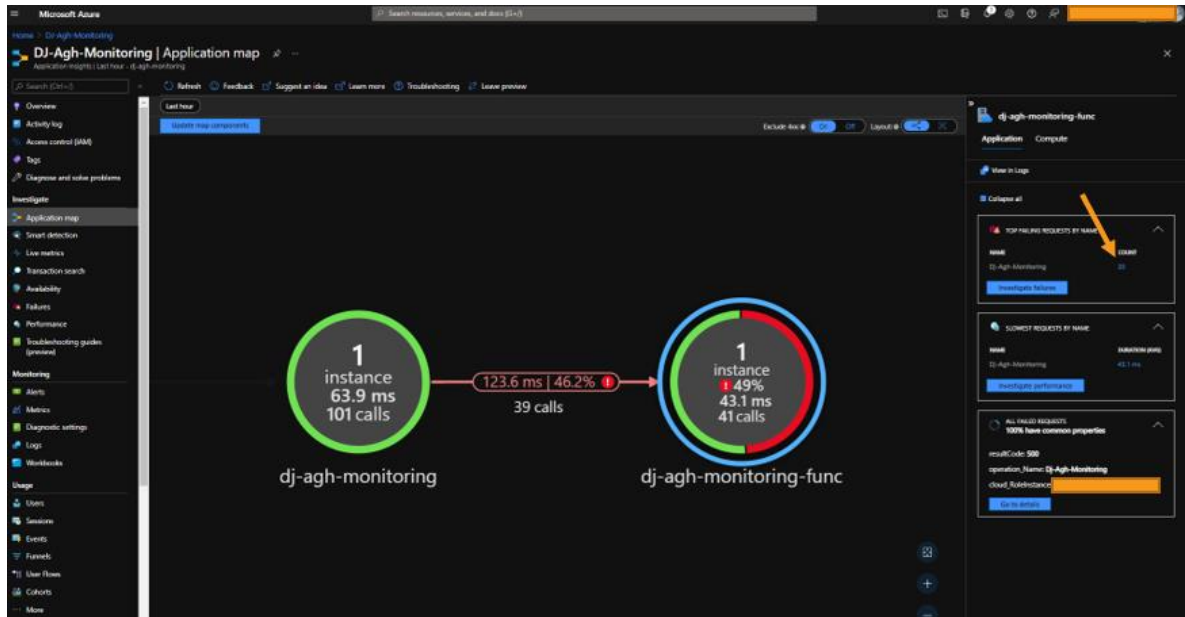
6. Przeanalizuj w jaki sposób błędy aplikacji zostały zawarte na diagramie oraz jakie informacje o błędach dostarcza nam Application Insights:



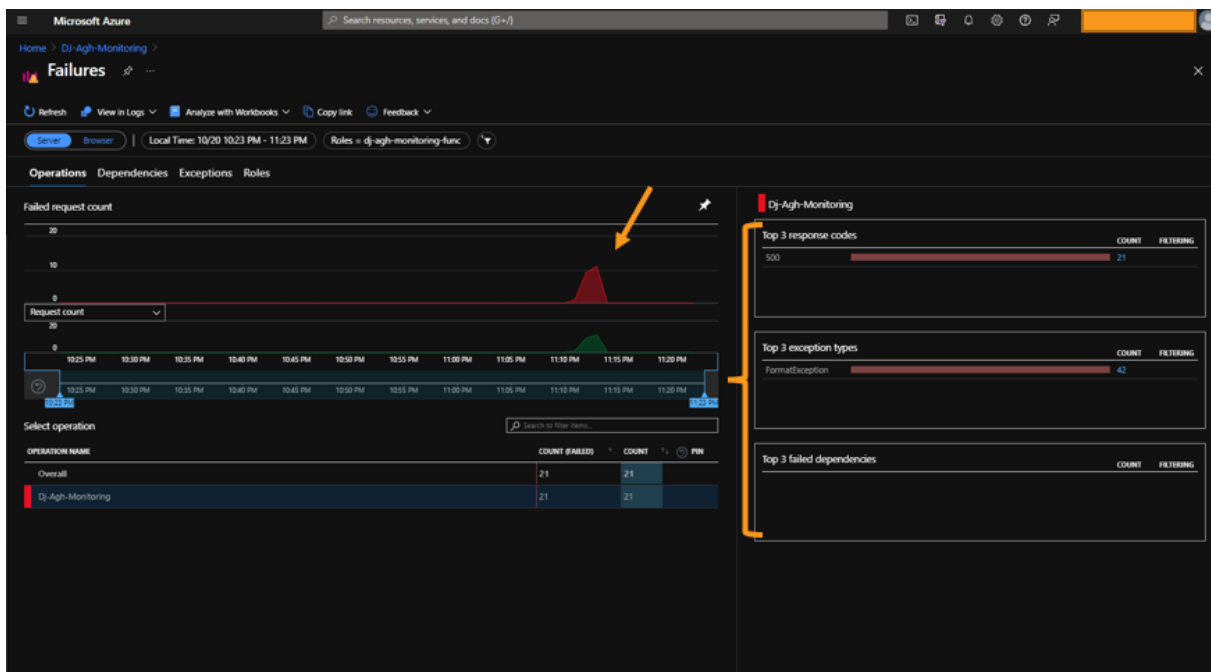
7. Klikając w usługę, która zwróciła błąd rozwiniesz panel z dodatkowymi informacjami:



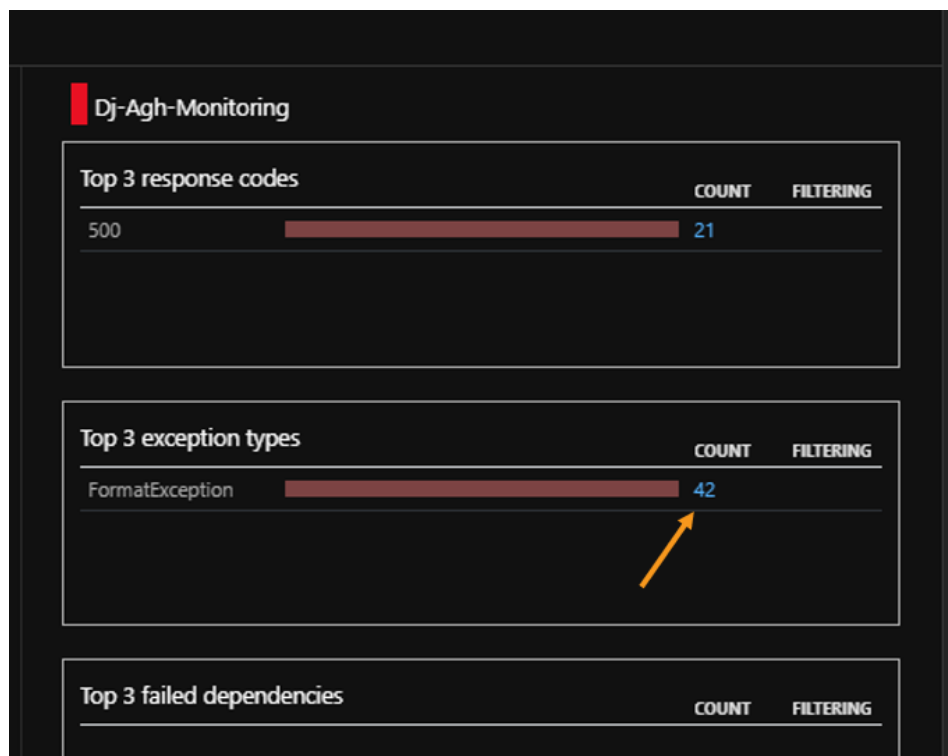
8. Kliknij ilość błędów przy usłudze, aby zostać przekierowanym do szczegółów błędu i jego źródła:



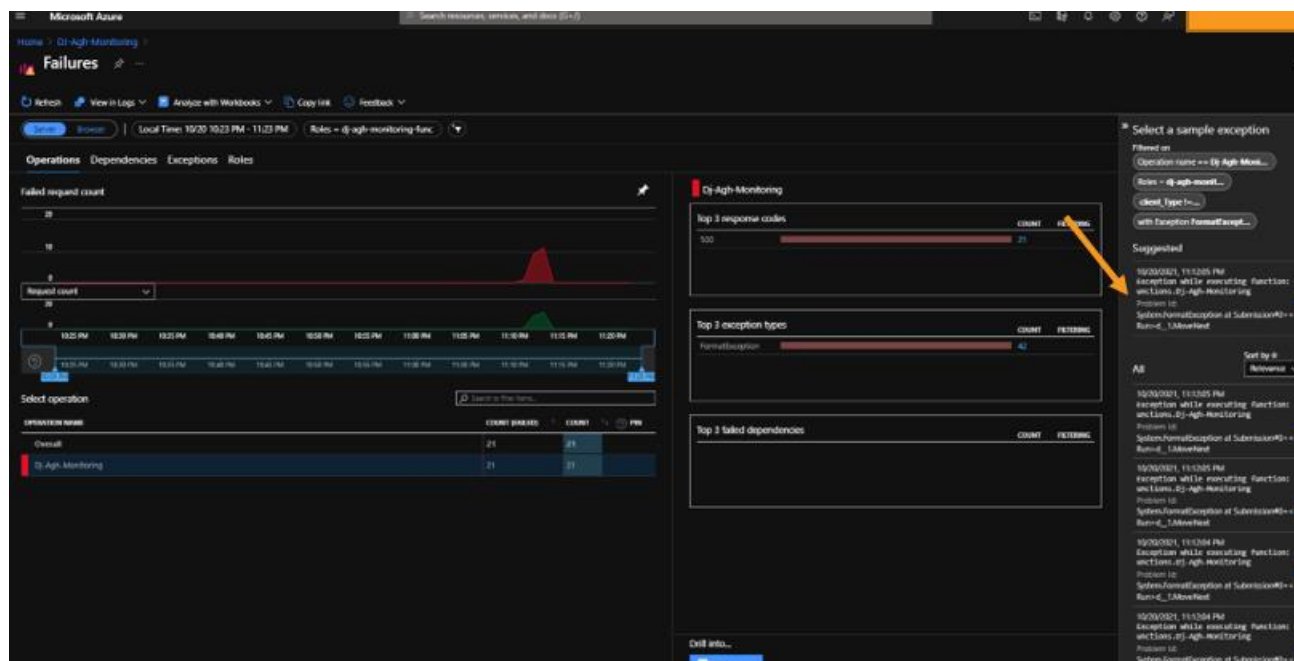
9. Zostałeś przekierowany na zakładkę *Failures*, która raportuje dokładne informacje na temat ilości błędów, kodu błędów oraz rodzajów wyjątków, które zwróciły twoje usługi na przestrzeni czasu.



10. Kliknij w ilość wystąpień danego wyjątku



11. Rozwinięty panel w prawej strony przedstawia wszystkie wystąpienia danego wyjątku na przestrzeni czasu. Kliknij w jeden z nich:



12. Zostałeś przekierowany na widok, który przedstawia stos wywołań poszczególnych zapytań w ramach twoich usług. Klikając na **EXCEPTION**, uzyskasz dokładne informacje na temat wystąpienia błędu.

The screenshot displays the 'End-to-end transaction details' page in the Azure portal. The left sidebar contains search filters and suggested results. The main area shows a table of events for the operation ID 'c6fc3d1ecbe482fa2a7a7a226eb71'. An orange arrow points to a row with the event name 'EXCEPTION System.FormatException'. The right-hand panel shows the 'Dependency Properties' and 'Custom Properties' for the selected event, including details like 'Event time', 'Type', 'Result code', 'Call status', 'Duration', 'Name', and 'HTTPMethod'.

13. Odnaczenie pola wyboru *Just my code* w prawym panelu, doprowadzi Cię do dokładnej linii w kodzie źródłowym, która jest odpowiedzialna na wyrzucony wyjątek:

The screenshot displays the 'End-to-end transaction details' page in the Azure portal. The left sidebar contains search filters and suggested results. The main area shows a table of events for the operation ID 'c6fc3d1ecbe482fa2a7a7a226eb71'. An orange arrow points to a row with the event name 'EXCEPTION System.FormatException'. The right-hand panel shows the 'Call Stack' for the selected event. The 'Just my code' checkbox is checked, and the call stack shows the exception being thrown in the 'Run' method of the 'dj-agh-monitoring' function.

request (incoming)

Create work item

**EXCEPTION**  
System.FormatException



ProcessId 3788

prop\_invocationId 3107e742-6136-446b-8b62-76b3c9dbf52a

prop\_status Failed

InvocationId 3107e742-6136-446b-8b62-76b3c9dbf52a


**Call Stack**

☒ Just my code  Copy  Expand

Method	File	Line
Microsoft.Azure.WebJobs.Host.FunctionInvocationException		
System.Runtime.ExceptionServices.ExceptionDispatchInfo		
[internal code]		
Microsoft.Azure.WebJobs.Host.Executors.FunctionExecu	FunctionExecu	105
Inner exception System.FormatException handled at System.Runtime.ExceptionServices.ExceptionDispatchInfo.Throw		
System.Number.ThrowOverflowOrFormatException		
[internal code]		
Submission#0+<Run>d__1.MoveNext	FUNCTION	12
[internal code]		
Microsoft.Azure.WebJobs.Host.Executors.FunctionExecu	FunctionExecu	296

(Don't see snapshot? Troubleshoot)

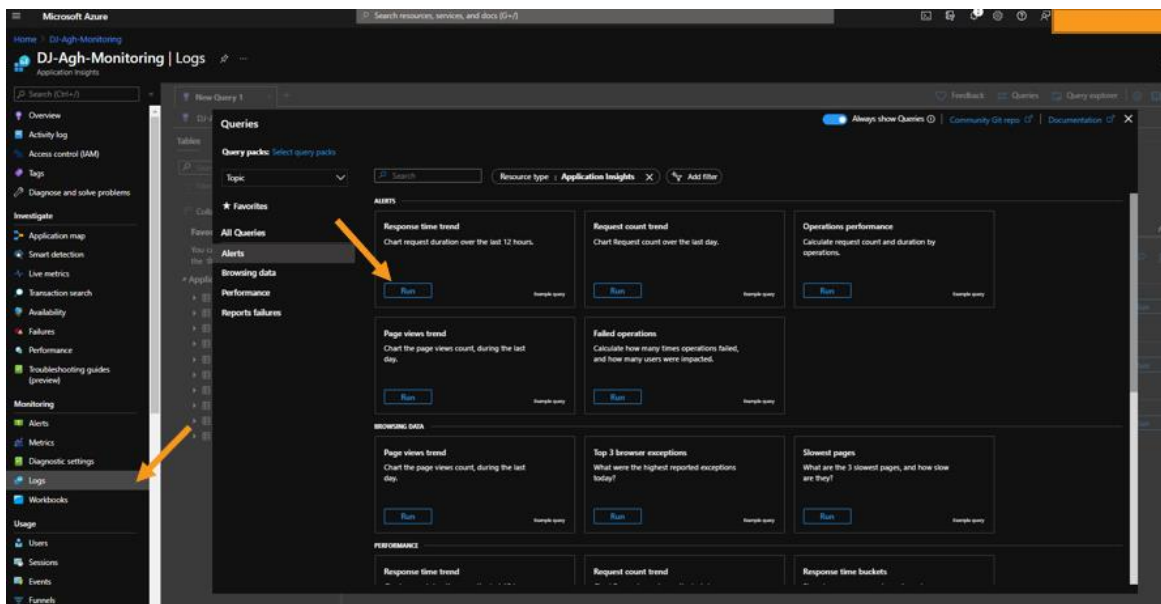
**Related Items**

 Show what happened before and after this exception in User Flows

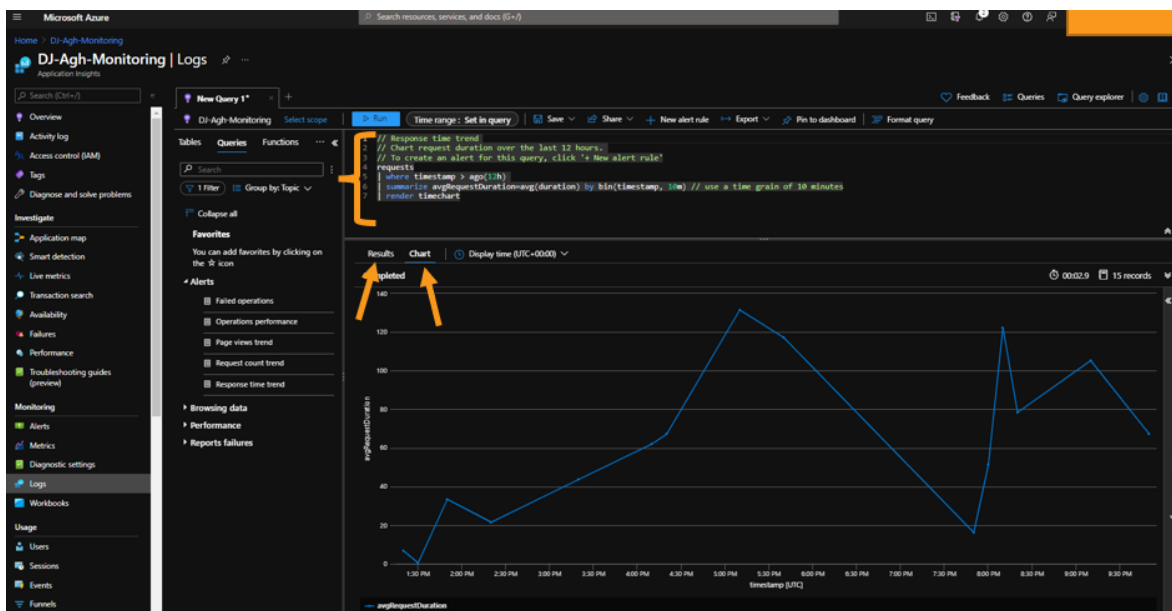
## XI. Analiza logów przy użyciu Log Analytics (aka Azure Logs)

Oprócz informacji o metrykach, w chmurze Azure jesteśmy w stanie przechowywać logi aplikacji. Możemy je przeglądać i analizować za pomocą narzędzia *Log Analytics*.

1. W Application Insights, kliknij w zakładkę Logs
2. Wybierz interesujące Cię zapytanie i kliknij przycisk *Run*



3. Na głównym ekranie pojawi się zapytanie napisane w języku Kusto

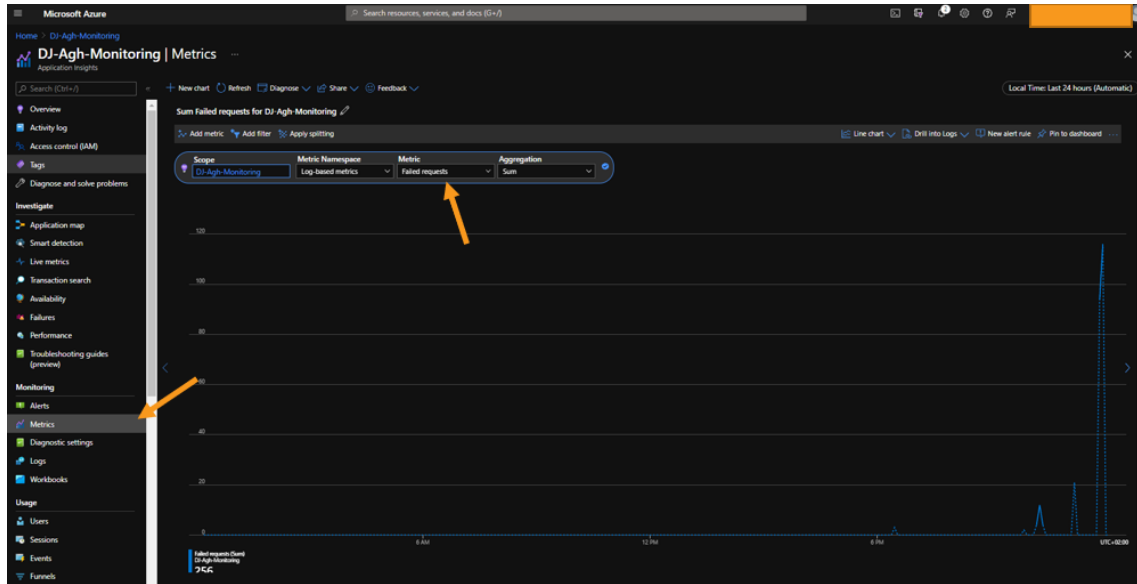


4. Logi możemy przeglądać zarówno w formie surowej (*Results*) jak i wizualizowanej graficznie (*Chart*)
5. Wykonaj kilka różnych zapytań i przeanalizuj możliwości *Log Analytics*

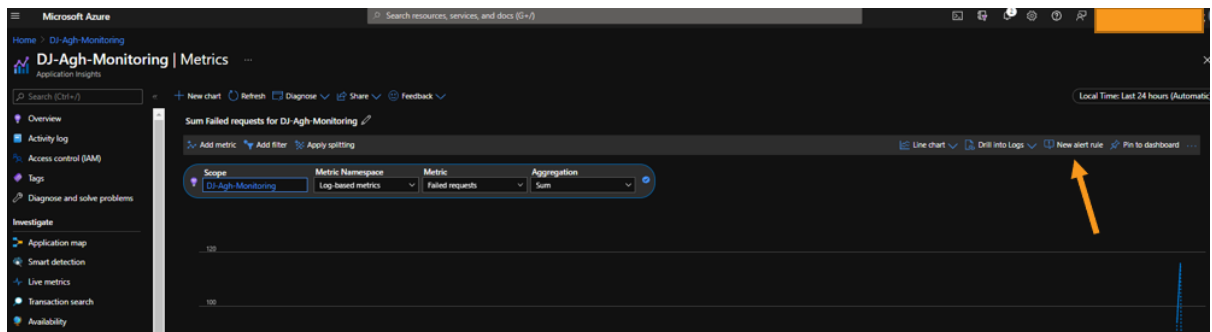


## XII. Definiowanie alertów przy użyciu Application Insights

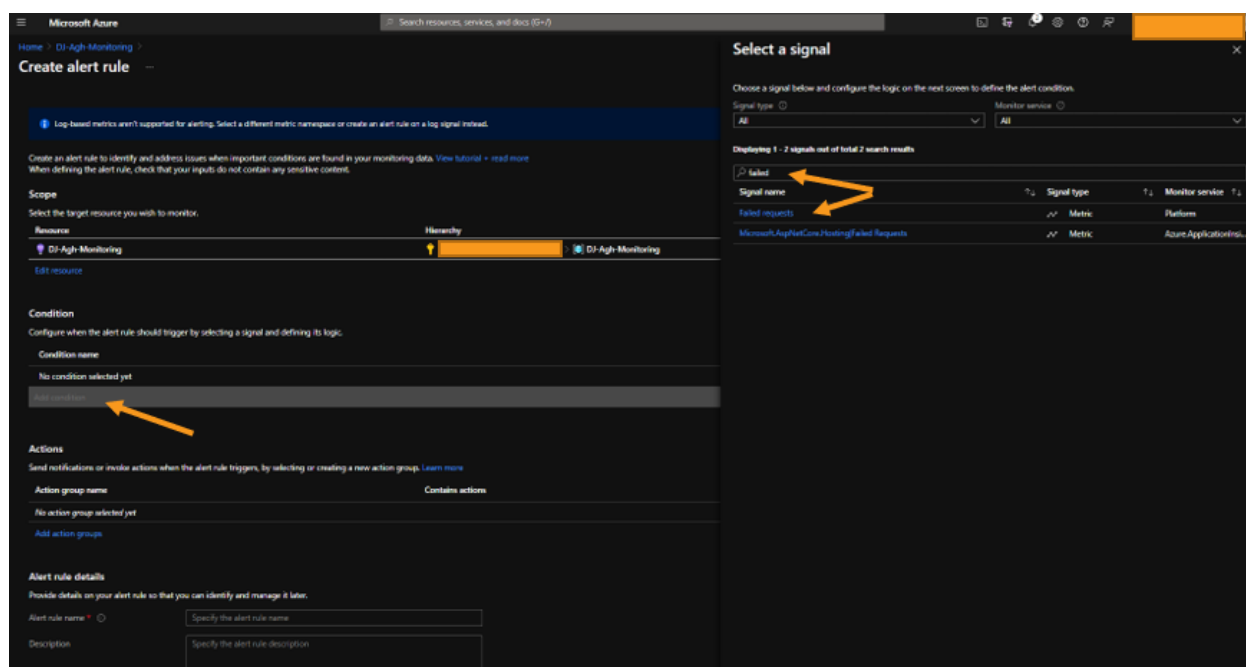
1. W Application Insights, przejdź na zakładkę *Metrics*, w której mamy możliwość graficznego przedstawienia danych metryk naszej aplikacji
2. W polu *Metrics* wybierz interesującą metrykę aplikacji i zatwierdź



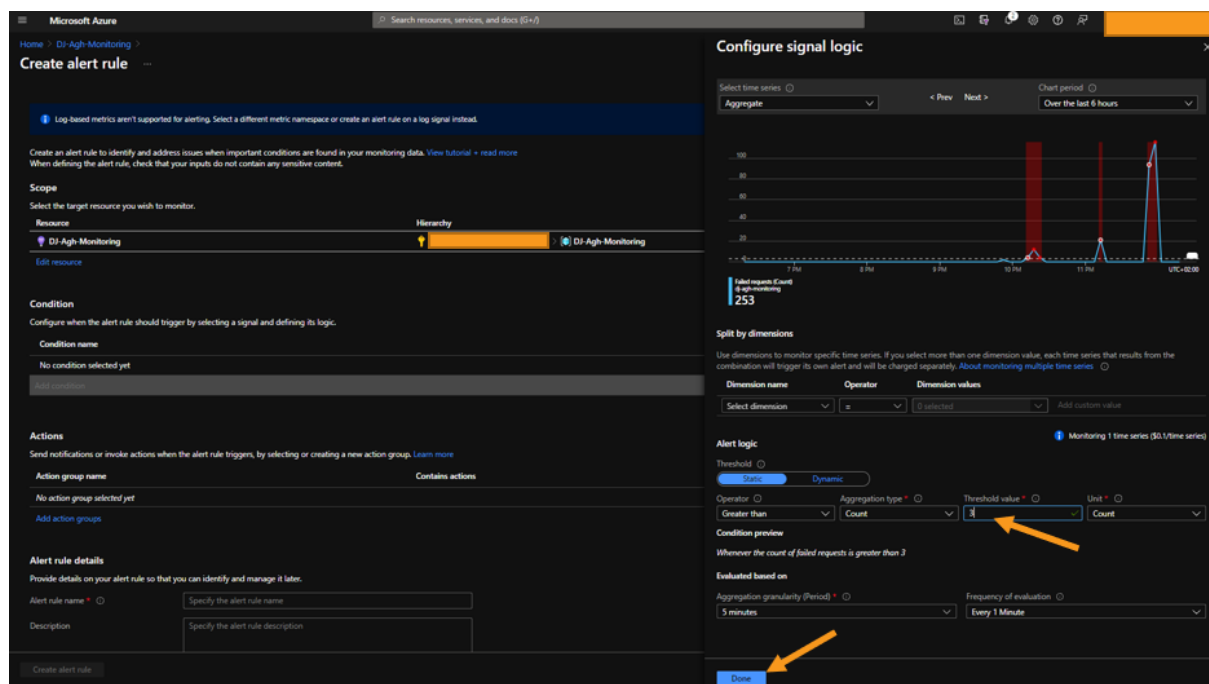
3. W tym przypadku, chcemy stworzyć alert, który powiadomi nas, że liczba nieudanych zapytań do naszej aplikacji przekroczyła dany poziom. W tym celu, klikamy w przycisk *New alert rule* w prawym górnym rogu:



4. Na kolejnym widoku, klikamy w *Add condition*, w celu zdefiniowania warunków naszego alertu i wybieramy nasz sygnał, którym jest metryka ilości nieudanych zapytań (*Failed requests*):

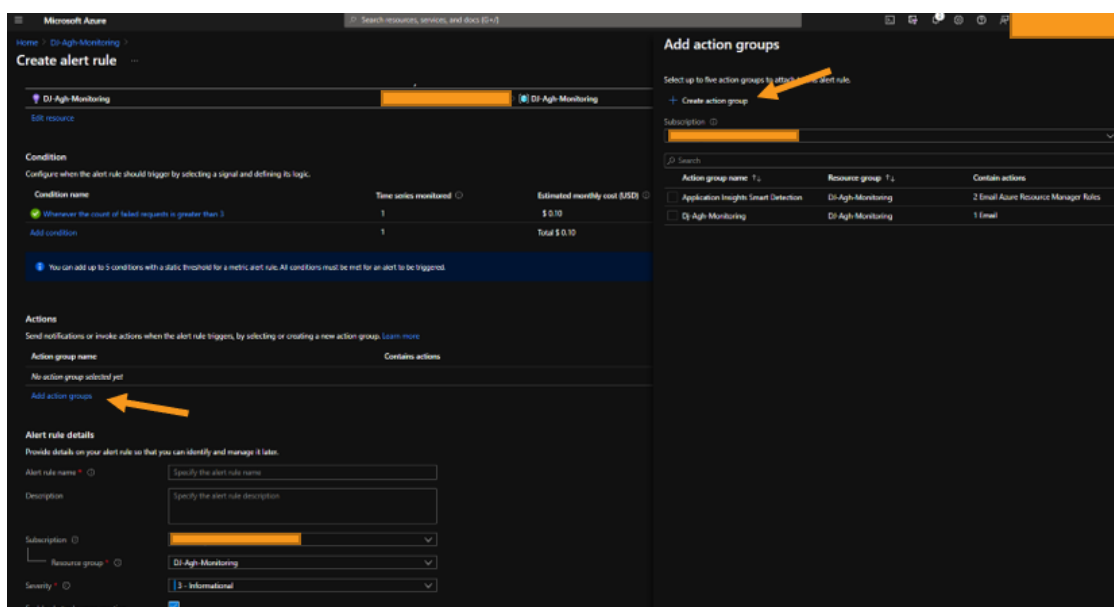


5. Ustawiamy naszą graniczną wartość (*Threshold value*), której przekroczenie będzie uruchamiało alert

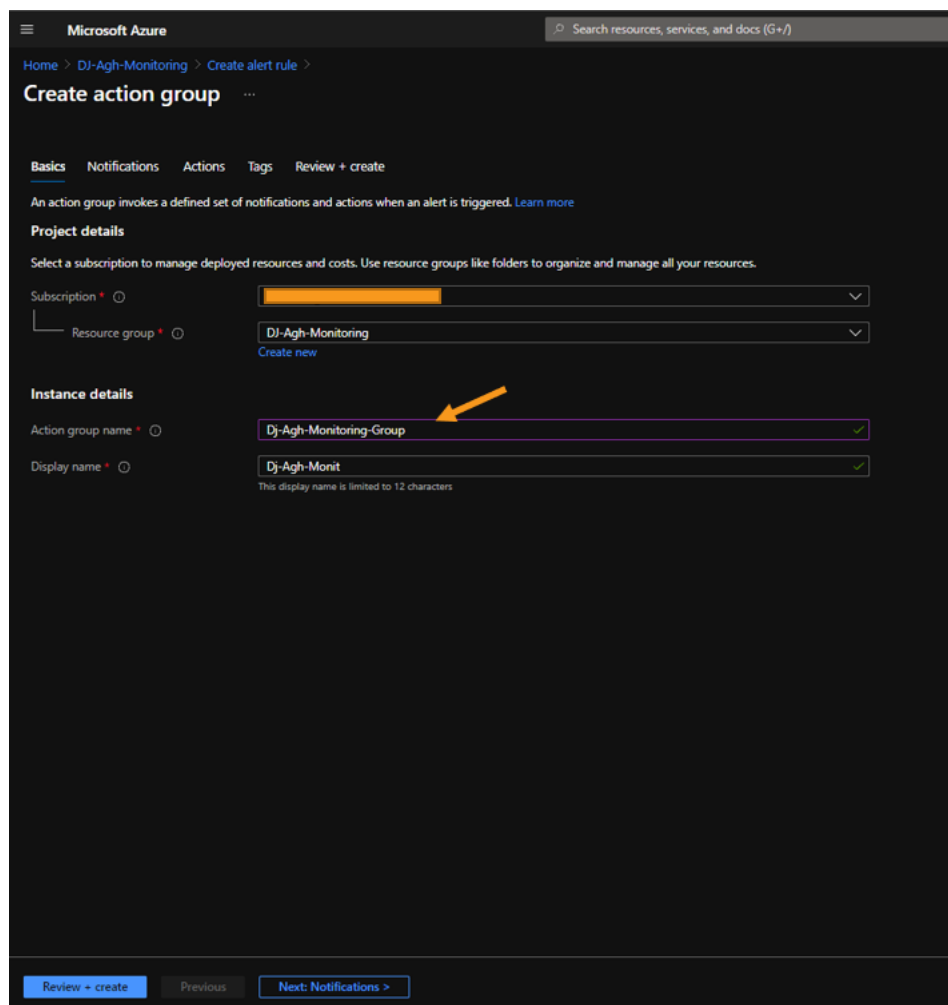


6. Zatwierdzamy zmiany przyciskiem *Done*

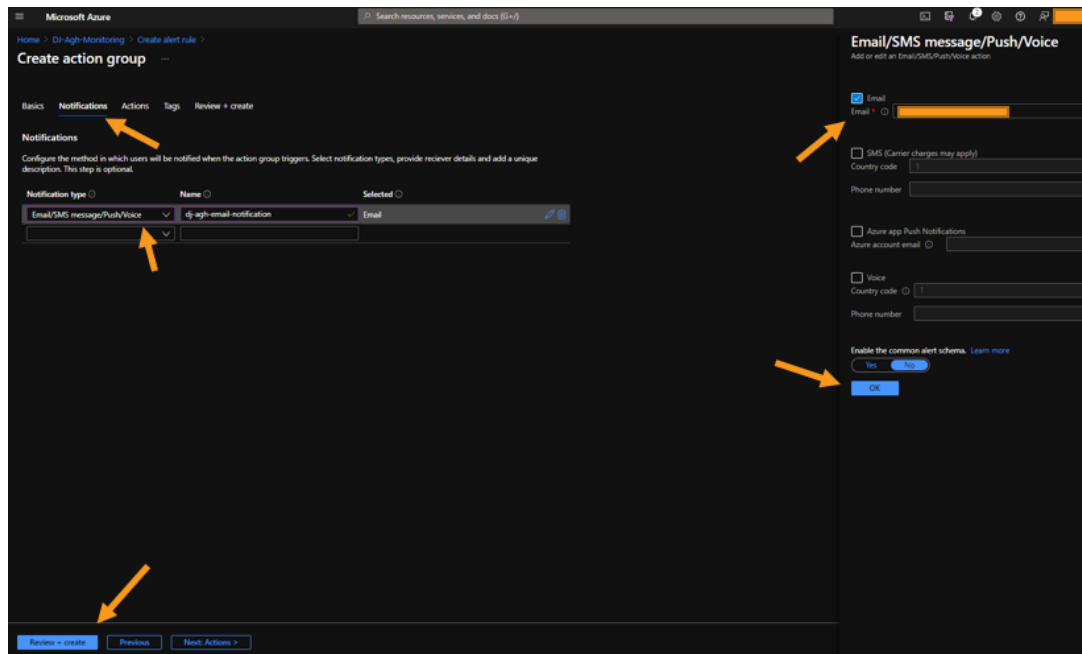
7. W kolejnym kroku musimy ustalić grupę odbiorców naszego alertu. W tym celu klikamy *Add action group*. Następnie, w rozwiniętym panelu wybieramy *Create action group*.



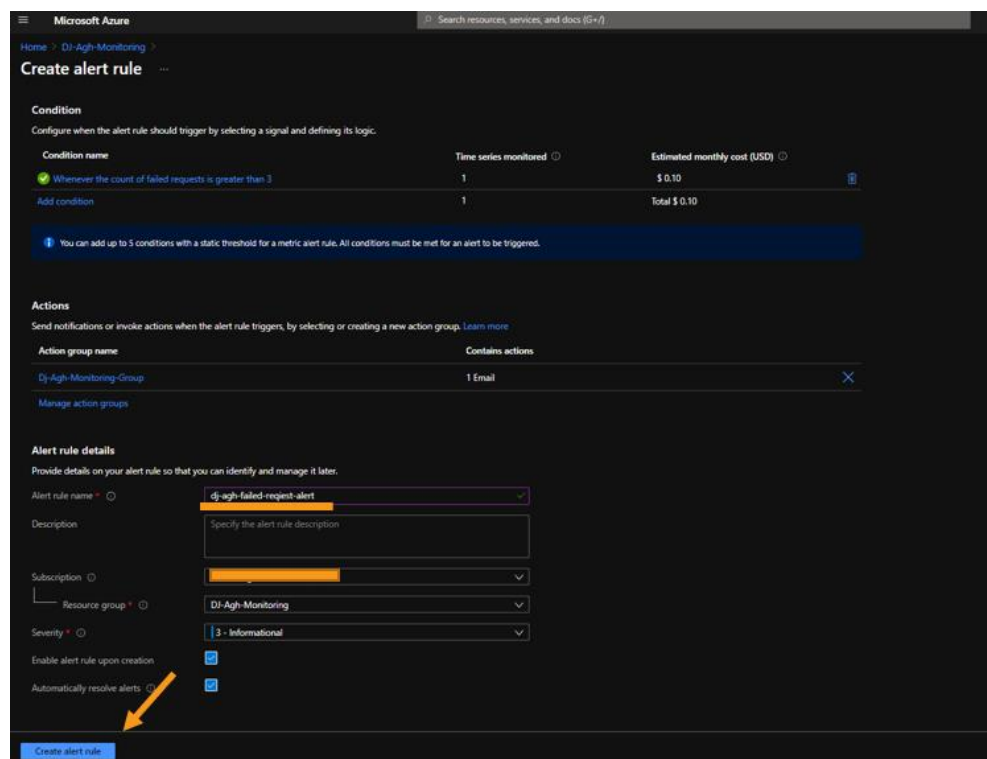
8. Na kolejnym widoku ustawiamy nazwę naszej grupy



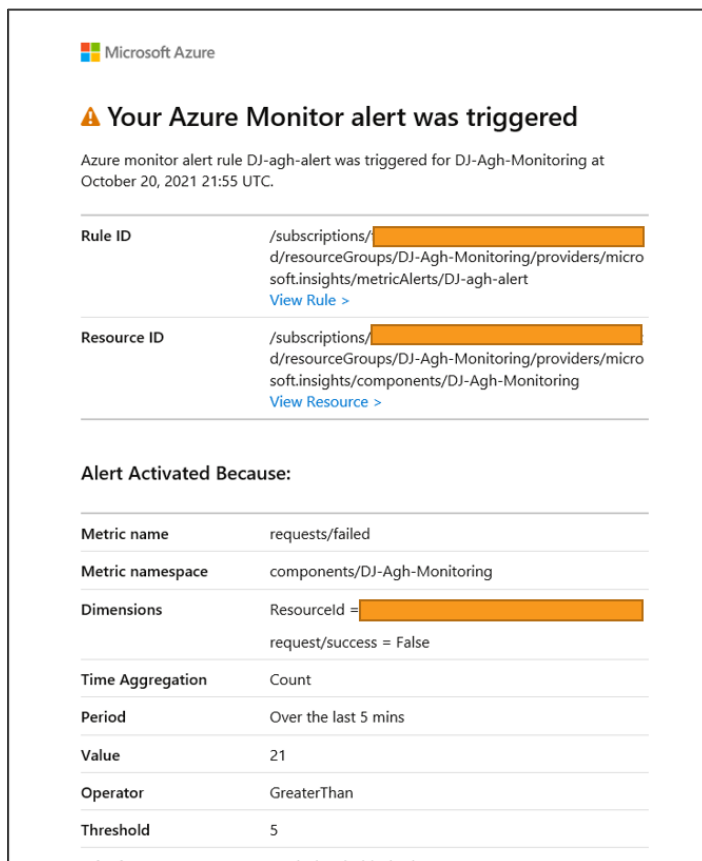
9. Na tym samym widoku, przechodzimy do zakładki *Notifications* i wybieramy rodzaj notyfikacji, które będą generowane przez alert. Na potrzeby tego ćwiczenia wybierzemy *Email/SMS Message/Push/Voice*. W rozwiniętym panelu wypełniamy dane dot. Interesujących nas notyfikacji i zatwierdzamy przyciskiem *OK*.



10. Zatwierdzamy przyciskiem *Review + create*  
11. Klikamy przycisk *Create*  
12. Na tym etapie mamy zdefiniowane warunki i grupę odbiorców alertu. Dodatkowo ustawiamy nazwę alertu i zatwierdzamy przyciskiem *Create alert rule*



13. Nasz alert jest zdefiniowany, znawigujmy do aplikacji webowej i ponownie wygenerujmy serię błędów przyciskając kilkakrotnie w guzik *Call API*.
14. Po chwili (do kilku minut) na naszą skrzynkę mailową powinniśmy dostać tego typu e-mail:



### Uwaga!

Po skończonych zajęciach warto usunąć/wyłączyć usługi, z których korzystaliśmy tak aby nie generowały one żadnych kosztów na subskrypcji.