DevOps Kata

**Monitor and Analyze**

Last updated: 12/18/2016



1. **TABLE OF CONTENT**

[Overview 3](#_Toc469899322)

[Prerequisites 3](#_Toc469899323)

[Exercises 3](#_Toc469899324)

[Exercise 1: Use application insights locally 4](#_Toc469899325)

[Task 1: Add app insights to a new project 4](#_Toc469899326)

[Task 2: Use App insights locally 4](#_Toc469899327)

[Exercise 2: Add custom insight telemetry 5](#_Toc469899328)

[Note: You must have installed the package management extension from the VSTS markeplace to get this started. 5](#_Toc469899329)

[Task 1: Use the SDK to create add instrumentation and custom events 5](#_Toc469899330)

[Task 2: Publish web app to azure 6](#_Toc469899331)

[Exercise 3: Add a web test 6](#_Toc469899332)

[Task 1: Add a web test 6](#_Toc469899333)

[Exercise 4: Use app analytics 7](#_Toc469899334)

[Task 1: Use analytics 7](#_Toc469899335)

[Query your telemetry 7](#_Toc469899336)

[Write a query 8](#_Toc469899337)

[Run a query 9](#_Toc469899338)

## Overview

Application Insights is an extensible Application Performance Management (APM) service for web developers. Use it to monitor your live web application. It will automatically detect performance anomalies. It includes powerful analytics tools to help you diagnose issues and to understand what users actually do with your app. It's designed to help you continuously improve performance and usability. It works for apps on a wide variety of platforms including .NET, Node.js and J2EE, hosted on-premises or in the cloud. It also integrates with your devOps process, and has connection points to a wide variety of other tools.

### Prerequisites

1. In order to complete the lab
2. 1. Log on to your Visual Studio / MSDN subscription and create or use your own VSTS instance. Alternatively, you can use a team sandbox VSTS instance if you have one.
3. 2. Have an azure subscription available that you can deploy resource to.
4. 3. Visual Studio Enterprise
5. 4. Install package management for your VSTS instance.

### Exercises

* 1. This hands-on lab includes the following exercises:
  2. Use application insights locally
  3. Add custom insight telemetry
  4. Add a web test
  5. Analyze a variable
  6. Estimated time to complete this exersize: **15 minutes**.

Exercise 1: Use application insights locally

#### Task 1: Add app insights to a new project

1. 1. Launch an instance of Visual Studio 2015 from the taskbar.
2. 2. Select File | New | Project and then select the Visual C# | Web node.
3. 3. Select the ASP.NET Web Application template.
4. 4. Ensure that the “Add Application Insights to Project” option is selected.
5. 5. At this point, you should be prompted to sign in to Azure using your credentials. Sign in to add your account to Visual Studio.
6. 6. If you have multiple subscriptions associated with your account, make sure to select the one that you would like to use for this lab.
7. 7. Note that telemetry will be sent to a new Application Insights resource by default. Accept the default by clicking OK to continue.
8. 8. In the ASP.NET project wizard, note that both ASP.NET 4.6 and ASP.NET 5 Preview templates are shown. Select the ASP.NET 4.6 MVC template, select the option to “Host in the cloud” and then click OK.
9. 9. In the Configure Microsoft Azure Web App window, use the default suggestion for Web App name.
10. 10. Select the Central US region.
11. 12. Click OK to continue. Note that it will take a few moments for needed NuGet packages to be downloaded and for the Azure resources to be created.
12. 13. After all of the configured resources are created, the Azure web application should be ready (but not deployed yet).
13. 14. Note the PublishScripts folder that was added to the solution. This folder contains PowerShell scripts and configuration that automate the publication of the web application to Azure.
14. 15. In Solution Explorer, double-click on ApplicationInsights.config to open it. This configuration file specifies the various telemetry modules and initializers to use, as well as the Application Insights telemetry key.

#### Task 2: Use App insights locally

Now that we have configured Application Insights for the application, we can test it out by running locally. Press F5 to launch the MVC application.

2.       Exercise the application by clicking on Home, About, Contact pages.

3.       Close the browser window and return to Visual Studio. You should see a popup stating that “Your first Application Insights events have been sent!”

4.       Return to the Application Insights blade in the Azure portal and click the Refresh button. Note that some application performance and usage data is now available (if not, wait a minute and refresh again). Click on the first chart shown on the Overview Timeline that corresponds to server response time.

5.       All of the data that you see here was sent during the debugging session that you just performed.

Note: It is possible to configure Application Insights to disable telemetry for debugging purposes.

6.       We’ll examine the Application Insights blade in more detail later.

Exercise 2: Add custom insight telemetry

#### Note: You must have installed the package management extension from the VSTS markeplace to get this started.

#### Task 1: Use the SDK to create add instrumentation and custom events

1. Open \_Layout.cshtml once again and take a look at the IntelliSense for the appInsights object (temporarily remove the period after ‘appInsights’ and then re-add it). Note that the JavaScript SDK includes methods to track events, exceptions, metrics, and traces.

2.       Press Escape to prevent modification of the JavaScript. Although we could add in additional instrumentation in the client, we’ll opt to just track a custom event on the server for now.

3.       Open HomeController.cs in the code editor and add a using statement for Application Insights to the top of the file:

using Microsoft.ApplicationInsights;

4.       Locate the Contact method and then add the following snippet to the beginning (**Using Control-C, Control-V)**:

TelemetryClient client = new TelemetryClient();

client.TrackEvent("Contact Page",

   new Dictionary<string, string>()

   {

      ["IsAuthenticated"] = User.Identity.IsAuthenticated.ToString(),

      ["DayOfWeek"] = DateTime.UtcNow.DayOfWeek.ToString()

   });

#### Task 2: Publish web app to azure

1. Right-click on the project node in Solution Explorer and select Publish.

2.       The connection should already be setup with Web Deploy, so click the Publish button to initiate the publication process. This process will take a few moments.

3.       The published web application should load in a browser window. Exercise the application as you did during the local debug session earlier. Make sure to visit the Contact page and refresh it a few times.

4.       Click on the Register link in the top-right and go through the process of creating a new account (you can use a fake email address here). Then exercise the application as before, ensuring that you visit the Contact page.

5.       Navigate to a page that does not exist by modifying the address to be “/Demo”. This will result in a 404 - Not Found response from the server.

6.       Take note of the base address to the web application running in Azure. We will use it in the next task.

Exercise 3: Add a web test

#### Task 1: Add a web test

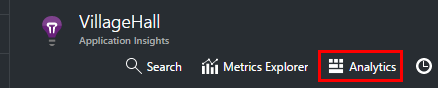
* 1. 1. Open the Azure portal and find your AI instance
  2. 2. In the Web Tests blade, click the Add Web Test button.
  3. 3. Provide a Test Name of “Landing Page” and provide the URL to the Azure Website that you previously created. Go ahead and select a couple more test locations while you are here. Click Create when you are done defining the test.
  4. 4. It will take a few minutes for the availability tests to start recording data, so it is suggested that you refresh the Application Insights blade later on to see the results.

5. Close the Web Tests blade.

Exercise 4: Use app analytics

#### Task 1: Use analytics

From your app's home resource in Application Insights, click Analytics.+

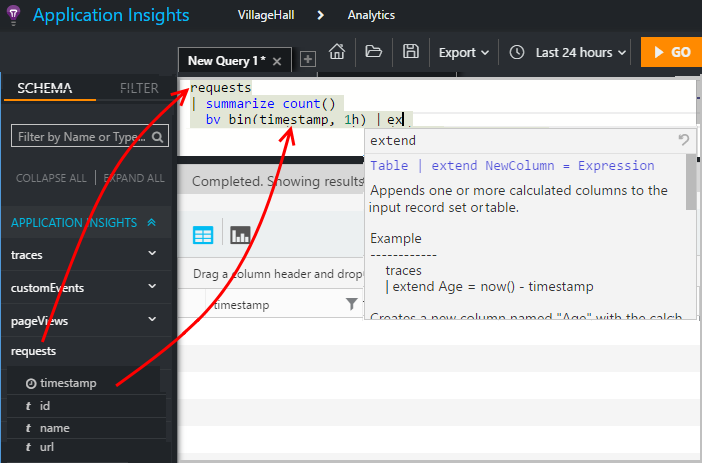
+

The inline tutorial gives you some ideas about what you can do.+

There's a [more extensive tour here](https://docs.microsoft.com/en-us/azure/application-insights/app-insights-analytics-tour).+

## Query your telemetry

### Write a query

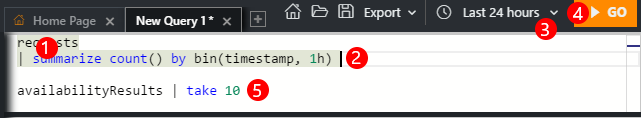
+

Begin with the names of any of the tables listed on the left (or the [range](https://docs.microsoft.com/en-us/azure/application-insights/app-insights-analytics-reference#range-operator) or [union](https://docs.microsoft.com/en-us/azure/application-insights/app-insights-analytics-reference#union-operator) operators). Use | to create a pipeline of [operators](https://docs.microsoft.com/en-us/azure/application-insights/app-insights-analytics-reference#queries-and-operators). +

IntelliSense prompts you with the operators and the expression elements that you can use. Click the information icon (or press CTRL+Space) to get a longer description and examples of how to use each element.+

See the [Analytics language tour](https://docs.microsoft.com/en-us/azure/application-insights/app-insights-analytics-tour) and [language reference](https://docs.microsoft.com/en-us/azure/application-insights/app-insights-analytics-reference).+

### Run a query

+

1. You can use single line breaks in a query.
2. Put the cursor inside or at the end of the query you want to run.
3. Check the time range of your query. (You can change it, or override it by including your own [where...timestamp...](https://docs.microsoft.com/en-us/azure/application-insights/app-insights-analytics-tour#time-range) clause in your query.)
4. Click Go to run the query.
5. Don't put blank lines in your query. You can keep several separated queries in one query tab by separating them with blank lines. Only the query that has the cursor runs.