



+5, Insightful: Azure Application Insights to improve your code



Tim Jarzombek
@tim_jarz

Who are *you*?

Developers

- Front-end *and* back-end can benefit from App Insights
- *Probably* already using Azure
- **Definitely** not anti-M\$FT
- Care about bugs, performance in production



Who are *you*?

Ops

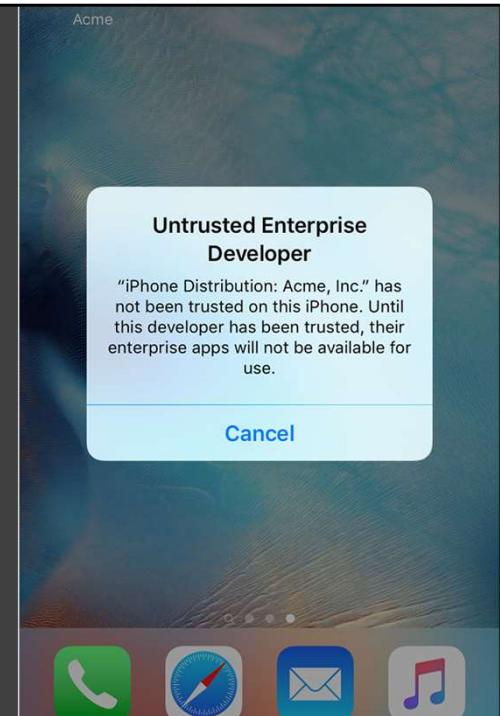
- *Probably* using Azure
- Want a tool to diagnose errors
- Want to know performance



App Insights can be used by more than devs, ops
- Business Analyst

Who am I?

- *Reluctant* “full-stack” developer
 - Mainly focus on back-end C#
 - Almost dangerous with Angular
- *Occasionally* masquerades as ops
 - CI & CD pipelines
 - DevTest environment management
- Like finding ways to do job *easily efficiently*
- Think that App Insights is under-appreciated



8 years of “enterprise-y” web-focused development

Currently developer @ Allegion – Schlage locks

Work on an e-commerce project. Bugs or issues that prevent people buying stuff are bad.

A long, long time ago...

BC (Before Cloud)

- Monoliths
- A couple servers
- Log files
 - Event Viewer
 - IIS / Apache
 - Plain text



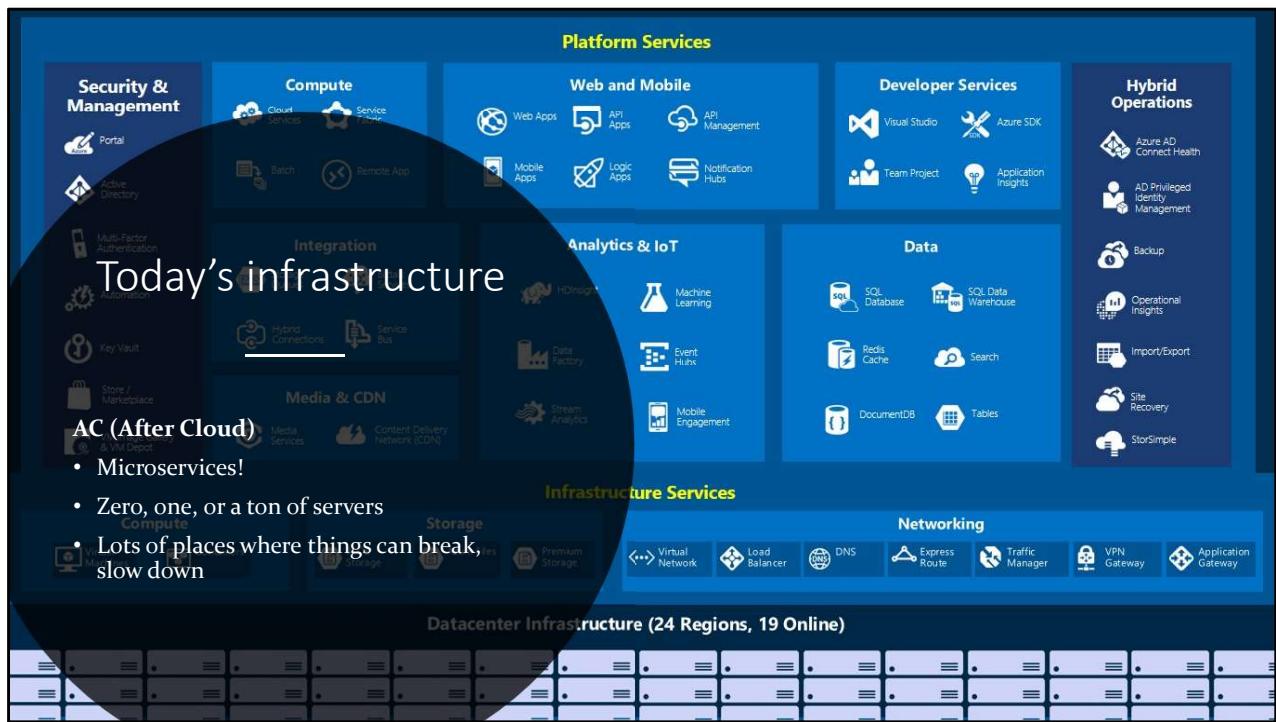
Monolithic code-bases

Software _installed_ on a couple servers

- Maybe even wrote the installer

Log files

- No real standards
- Mostly added as needed
- Usually didn't realize you didn't have something when you needed it the most



- Eighteen different “compute” services offered on Azure
- Building the right solution probably requires multiple services
 - API running on App Service
 - VM for compute-heavy processing (images)
 - Functions to handle event-based architecture

Note: still no time machine to add logging you wish you had



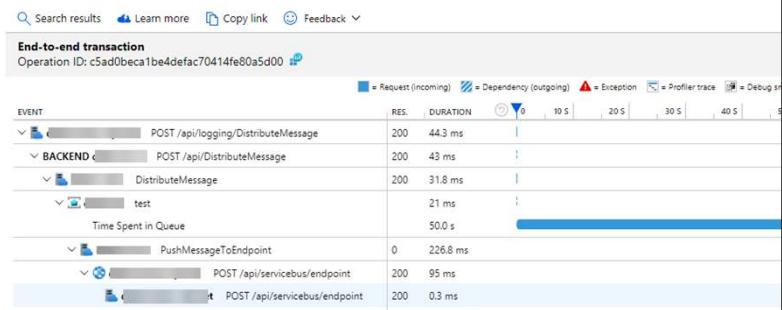
First solution: logging & analytics

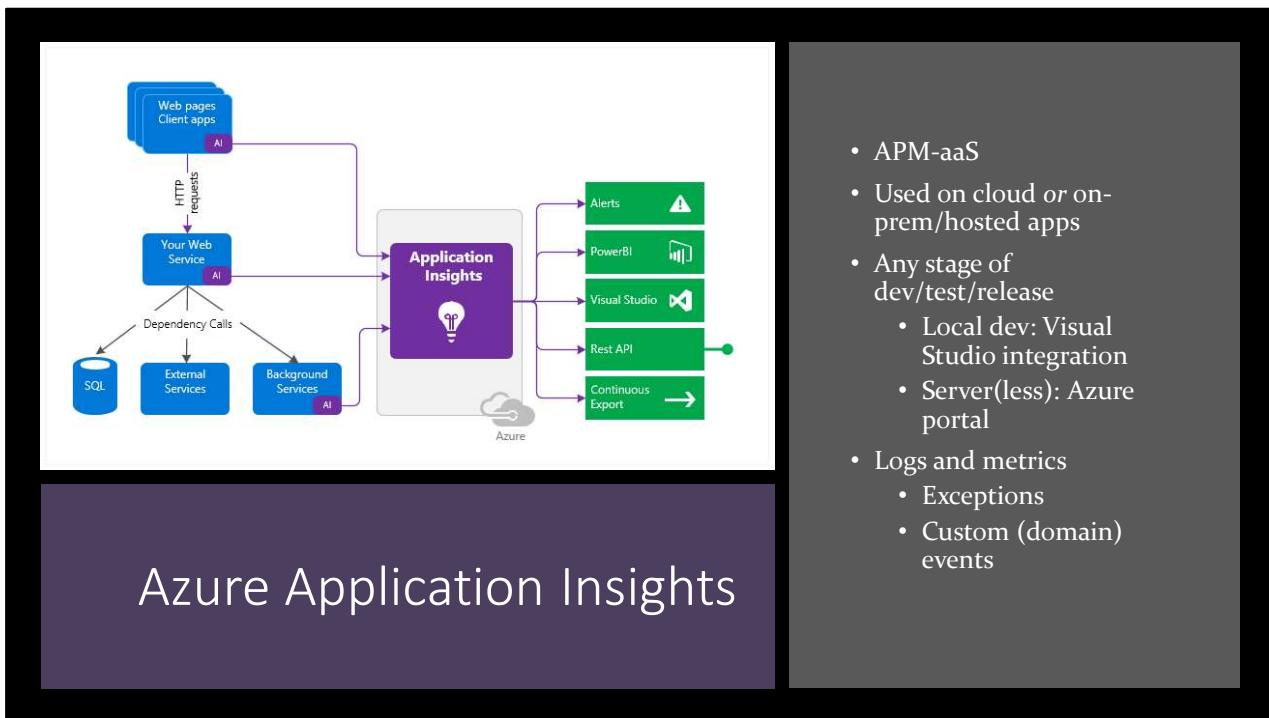
- Log everything!
- Send it one place
- Dig through when stuff breaks
 - Which microservices did that request hit?
 - Why did or didn't it do *that*?
- Enter analytics: a *bit* smarter
 - Alerting, actions based on logs

- Tools like Splunk or ELK stack
- Eventually, some structured logging
- We still can't agree on the one true way
 - Log4net, Serilog, NLog

Today: Application Performance Management

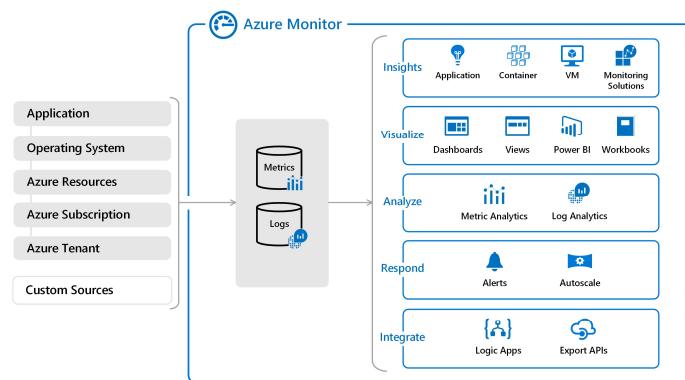
- Automatic log correlation
- Identifies performance issues
- Availability monitoring
- End-to-end
- Quickly identify the root cause
 - Line of code
 - Bad SQL query
 - Poor network connection



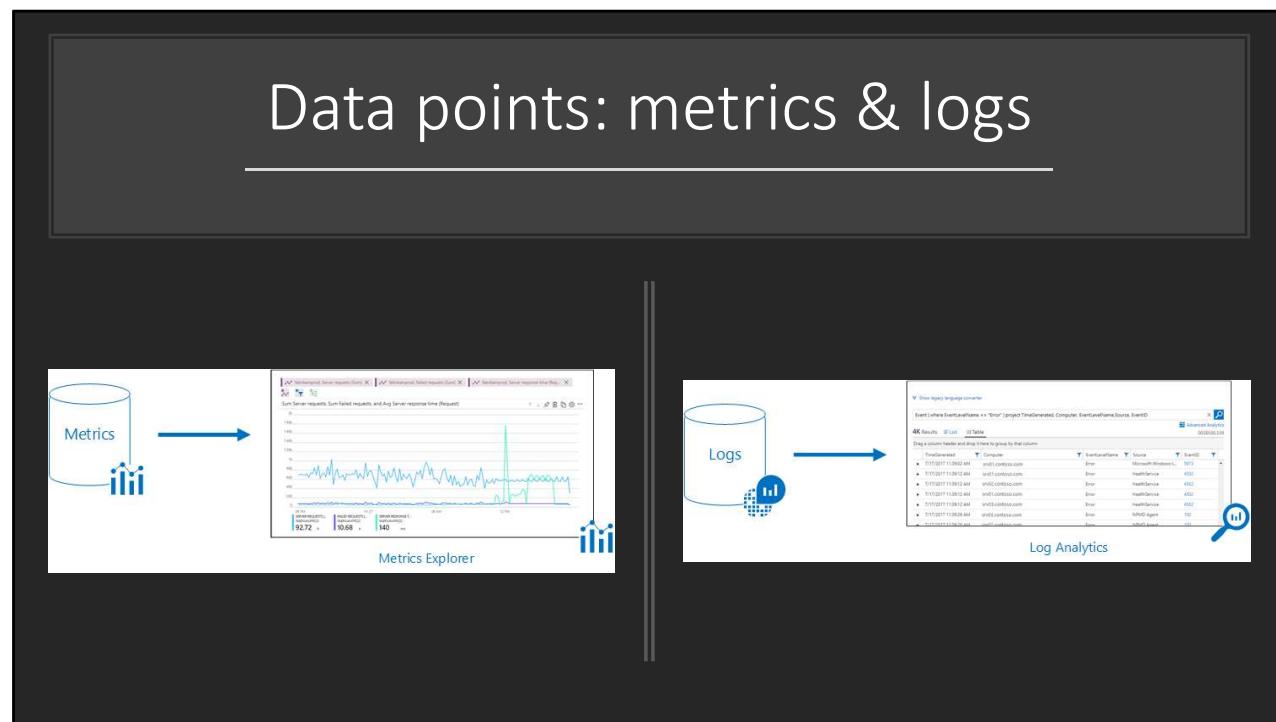


Bigger picture: Azure Monitor

- Suite of services focused on application telemetry
- Collect
- Analyze
- See
- Act



Data points: metrics & logs



Visualize metrics in Metrics Explorer

- Numeric data: CPU utilization, requests
- Data that can be aggregated

Review logs in Log Analytics

- Event traces, exceptions
- Structured (or not)

Lots of data, but we want information

Actionable information

Application Insights

The screenshot displays the Application Insights dashboard with the following sections:

- Failed requests:** A chart showing failed requests over time, with a total count of 430.
- Server response time:** A chart showing server response time in milliseconds, with a mean of 59.01 ms.
- Server requests:** A chart showing server requests per second, with a mean of 3.56 s.
- Availability:** A chart showing availability percentage, with a mean of 31.15 %.
- App Service Home Page test summary:** A dependency graph showing calls between services. Nodes include fabrikamaccount (Azure Blob), fabrikamprod, and fabrikamxyz (SQL). Call details are listed below:

Dependency	Latency (ms)	Throughput (calls)
fabricamaccount to fabrikamprod	16.2 ms	851 calls
fabricamprod to fabricamaccount	25.8 ms	25.8 ms
fabricamprod to fabrikamxyz	20 ms	111 calls
fabricamxyz to fabrikamprod	169.1 ms	169.254 calls
fabricamxyz to fabricamaccount	158 ms	158 calls

Below the dependency graph is a heatmap showing call distribution over time.

Quickly see issues

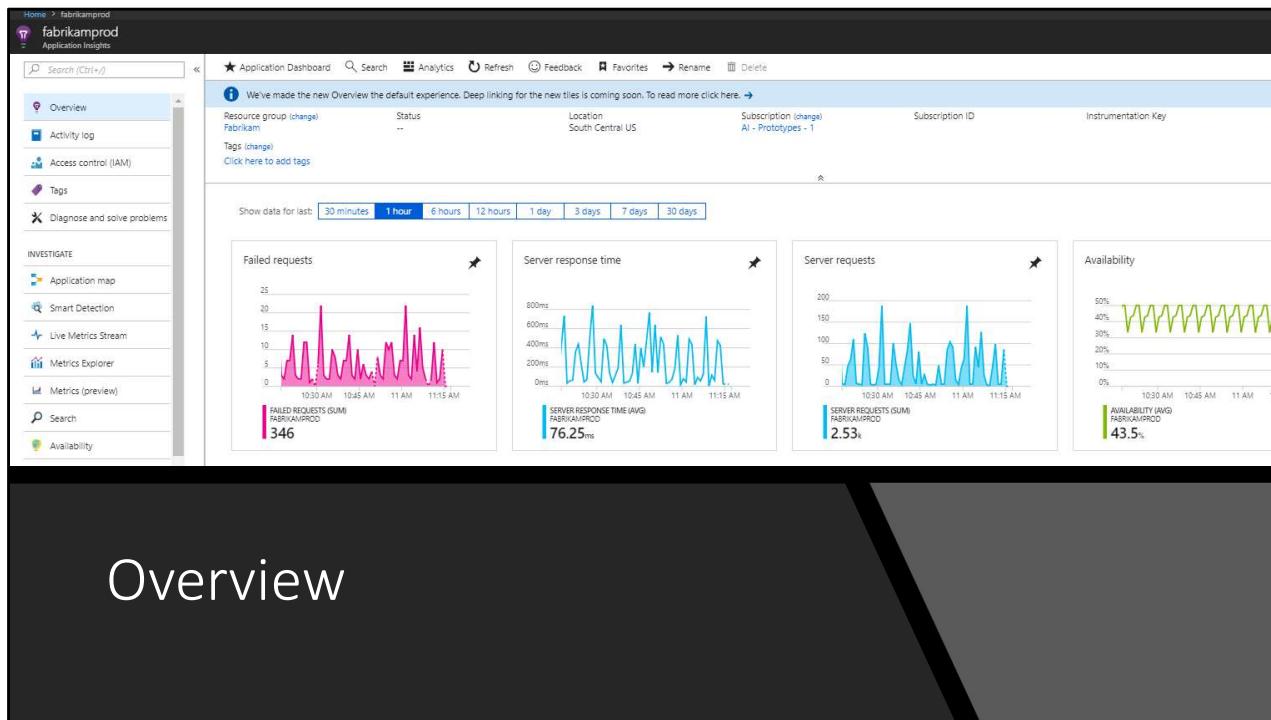
- Failed requests
- Dependencies to external systems
- Availability

What can Application Insights do?

- Requests
 - Rates, response times, failures
- Dependencies
 - Rates, response times, failures
- Exceptions
 - Aggregated statistics
 - Detailed stack traces, correlated
- Client-side metrics
 - Page views and performance
 - Exception tracking
 - Includes AJAX calls
- Trace log integration
- Your own events and metrics
 - New user, Order placed
 - Whatever you need



Some of these options are deprecated/redirect

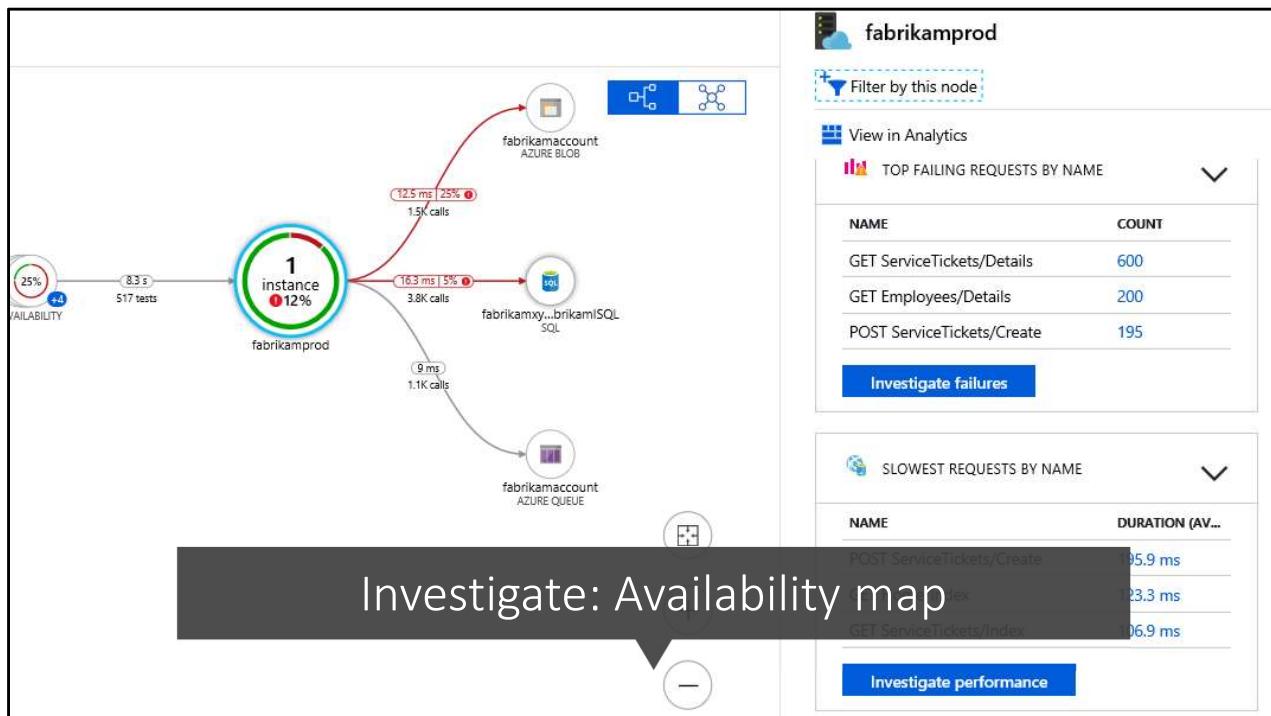


Overview

Landing page in Application Insights resource

Quickly see some of the most important metrics for web applications

- Failures
- Response times
- Inbound traffic
- Availability



Quickly investigate issues in and outside our control:

- Issues in our app like
 - Top failing requests
 - Slowest requests
- And how they correlate to external services
 - Databases
 - Azure storage
 - External APIs

The screenshot shows the Azure Application Insights interface for the 'fabrikamprod' application under the 'Smart Detection' tab. The left sidebar includes options like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Investigate (with Smart Detection highlighted), Live Metrics Stream, Search, Availability, Failures, Performance, Servers, Browser, Troubleshooting guides, Monitoring (Alerts, Metrics, Logs (Analytics), Workbooks), and Application map.

The main area displays four detected issues:

- Abnormal rise in exception volume (preview)**: When: 7/28 5:30 AM - 7/29 5:29 AM; What: 450% increase in 'System.Web.HttpException' volume compared to the previous 7 days.
- Insecure form data transmission detected (preview)**: When: 7/9 5:30 AM - 7/10 5:29 AM; What: 2 operations or forms in your application submit data to insecure (non-HTTPS) URLs. Note: Data provided by 1 user was potentially compromised due to this insecure data transmission.
- Potentially insecure URL access detected (preview)**: When: 7/9 5:30 AM - 7/10 5:29 AM; What: 2 URLs were accessed by both HTTP and HTTPS protocols. Note: 4 users accessed multiple URLs using HTTP instead of HTTPS.
- Abnormal rise in exception volume (preview)**: When: 6/29 5:30 AM - 6/30 5:29 AM; What: 575% increase in 'System.Web.HttpException' volume compared to the previous 7 days.

A large dark overlay box covers the bottom right portion of the screen with the text "Investigate: Smart Detection".

Possible issues that Azure has detected based on your application's telemetry:

- An uptick in exceptions
- Data submitted over insecure endpoints... that'd be tough for me to recognize among all the log data

Incoming Requests

- Requests/Sec: 8, 6, 4, 2, 0
- Request Duration (ms): 200, 150, 100, 50, 0
- Requests Failed/Sec: 6, 4, 2, 0

Outgoing Requests

- Dependency Calls/Sec: 15, 10, 5, 0
- Dependency Call Duration (ms): 100, 50, 0
- Dependency Calls Failed/Sec: 6, 4, 2, 0

Server Health

- Committed Memory (MB): 4,000, 3,000, 2,000, 1,000, 0
- CPU Total (%): 100, 50, 0

Sample Telemetry

- 5:12:10 PM | **Exception** | Exception: Error reading request content <--> The client disconnected.
- 5:12:10 PM | **Exception** | Exception: Error reading request content <--> The client disconnected.
- 5:12:09 PM | **Request** | 500 | 2 ms POST /QuickPulseService.svc/ping
- 5:12:09 PM | **Exception** | InvalidOperationException: Error deserializing a Ping request. Requires...sing.QuickPulseService.MonitoringDataPoint.
- 5:12:09 PM | **Exception** | InvalidOperationException: Error deserializing a Ping request. Requires...sing.QuickPulseService.MonitoringDataPoint.
- 5:12:09 PM | **Exception** | JsonSerializationException: Error converting value "Instance" to type 'Microsoft.ManagementServices.RealTimeDataProcessing.QuickPulseService.MonitoringDataPoint'. Path "...line 1, position 10. <--> Could not cast or convert from System.String to Microsoft.ManagementServices.RealTimeDataProcessing.QuickPulseService.MonitoringDataPoint.'
- 5:12:09 PM | **Exception** | ArgumentException: Could not cast or convert from System.String to Microsoft.ManagementServices.RealTimeDataProcessing.QuickPulseService.MonitoringDataPoint.

Time: 5:12:09 PM

Exception message: Error converting value "Instance" to type 'Microsoft.ManagementServices.RealTimeDataProcessing.QuickPulseService.MonitoringDataPoint'. Path "...line 1, position 10. <--> Could not cast or convert from System.String to Microsoft.ManagementServices.RealTimeDataProcessing.QuickPulseService.MonitoringDataPoint.'

Instance name: I_9

_MS_ProcessedByMetricExtractors (Name: Exceptions, Ver: 1.0)

```

Newtonsoft.Json.JsonSerializationException: Error converting value "Instance" to type 'Microsoft.ManagementServices.RealTimeDataProcessing.QuickPulseService.MonitoringDataPoint'. Path "...line 1, position 10. <--> Could not cast or convert from System.String to Microsoft.ManagementServices.RealTimeDataProcessing.QuickPulseService.MonitoringDataPoint."
   at Newtonsoft.Json.Utilities.ConvertUtils.EnsureTypeIsAssignableFrom(Object value, Type type)
   at Newtonsoft.Json.Utilities.ConvertUtils.ConvertOrCast(Object initialValue, CultureInfo culture)
   at Newtonsoft.Json.Serialization.JsonSerializerInternalReader.Deserialize(JsonReader reader, Type type, Object existingValue)
   at Newtonsoft.Json.Serialization.JsonSerializerInternalReader.Deserialize(JsonReader reader, Type type, Object existingValue, Object& result)

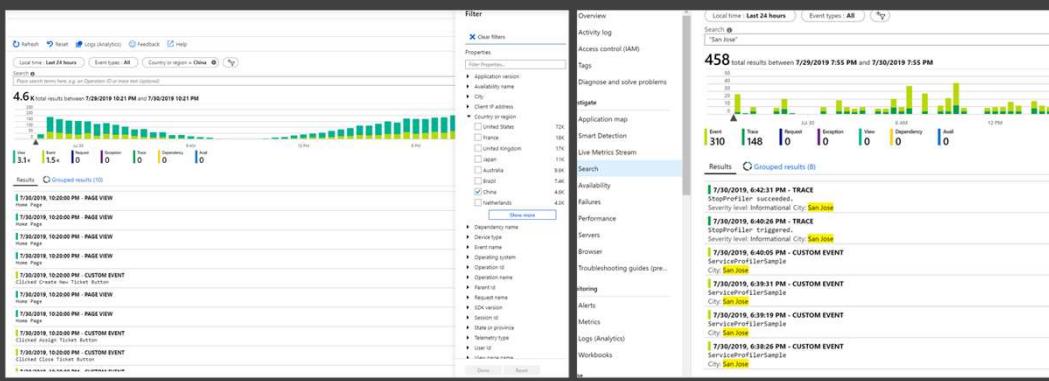
```

Investigate: Live Metrics Stream

Like Task Manager for the cloud
Based on live data coming into Application Insights

Sample Telemetry: see trace data as it's being processed
Super useful when you're investigating an issue happening in the moment
I use it when running batch operations in Azure Functions to monitor status

Investigate: Search



Search and filter across all of the telemetry processed

Simple filtering based on items like

- Country or region
- OS
- Device type
- User ID

Or just enter some text in the search bar

- An error message
- A URL
- An entity ID

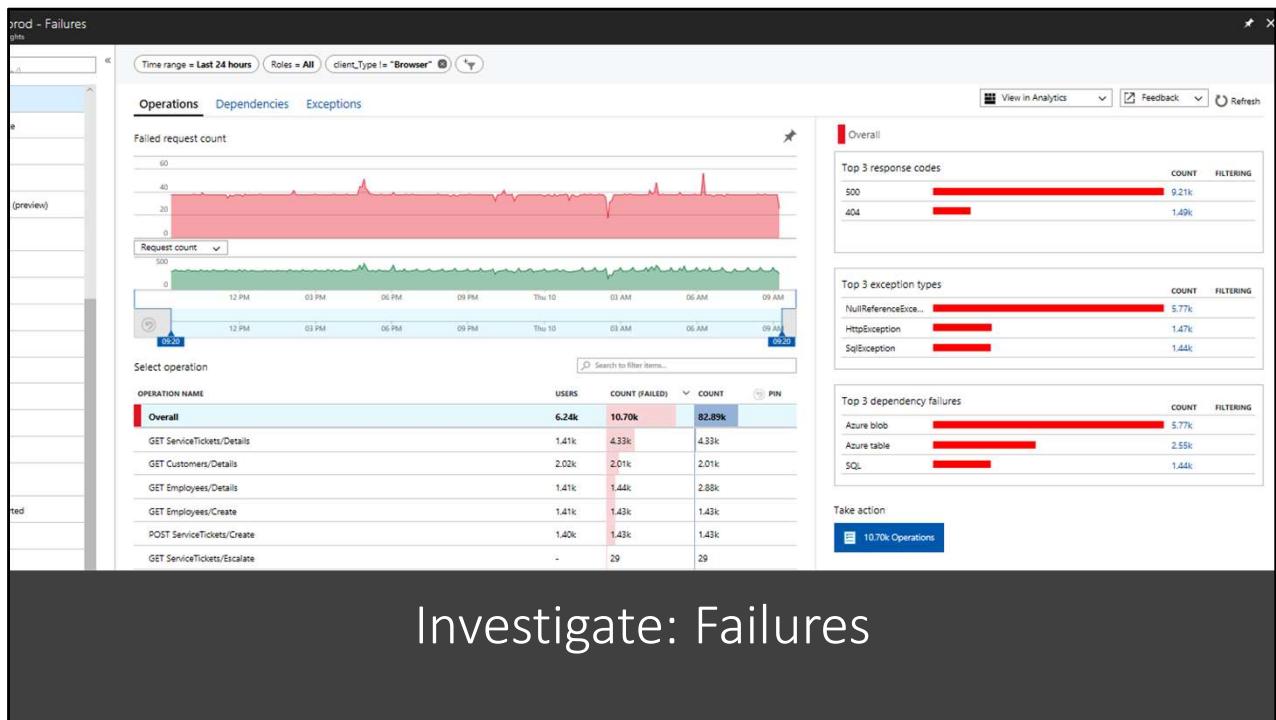
Investigate: Availability

The screenshot shows the Microsoft Application Insights interface for the 'Fabrikamprod' application. On the left, the navigation menu includes 'Overview', 'Activity log', 'Access control (IAM)', 'Tags', 'Diagnose and solve problems', 'Investigate' (with 'Availability' selected), 'Performance', 'Servers', 'Browser', and 'Troubleshooting guides (pre)'). The main area displays the 'Availability' dashboard with a line chart showing 100% availability from May 26 to Jun 09. Below the chart, a table lists availability tests: Overall (100.00%, 99.61%, 3.94 sec), fabrikamprod availability (100.00%, 99.57%, 4.88 sec), and fabrikamdev availability (100.00%, 99.64%, 3.00 sec). To the right, a 'Create test' dialog is open, showing basic information (Test name: 'Give your test a name'), test type ('URL ping test'), URL ('http://fabrikamprod.com'), and other settings like 'Parse dependent requests' (unchecked), 'Enable retries for availability test failures' (checked), and 'Test frequency' (5 minutes). It also shows 'Test locations' (5 locations configured) and 'Success criteria' (HTTP response: 200, Test Timeout: 120 seconds). A 'Create' button is at the bottom.

Create availability tests, pinging your application from around the world

- Sixteen test locations
- Every 5 / 10 / 15 minutes

Note: the “Multi-step web test” type has been deprecated. You’ve got a few years, but Microsoft currently doesn’t have a replacement.



Where I spend most of my time in App Insights

Quickly see:

- Endpoints with the most issues
- Most common exceptions
- Failing downstream dependencies

Can filter operations:

- Only the 500 errors since 404's are common
- Only the NullReferenceExceptions
- SQL failures

Includes both server and browser errors

The screenshot shows the Application Insights interface for an 'End-to-end transaction'. The transaction ID is 88d92a47671d4e83b231810887abd358. A large dark gray circle on the left contains the text 'Investigate: Failures'.

EVENT	RES.	DURATION
aiconnect2 GET Employees/Create	404	2.1 ms

EXCEPTION System.Web.HttpException
VM Employees Creation West US 100 ms

Request Properties:

- Event time: 6/14/2018 6:19:25 AM
- Request name: GET Employees/Create
- Response code: 404
- Successful request: False
- Response time: 2.1 ms
- Request URL: http://aiconnect2.cloudapp.net/FabrikamProd/Employees/Create

Call Stack:

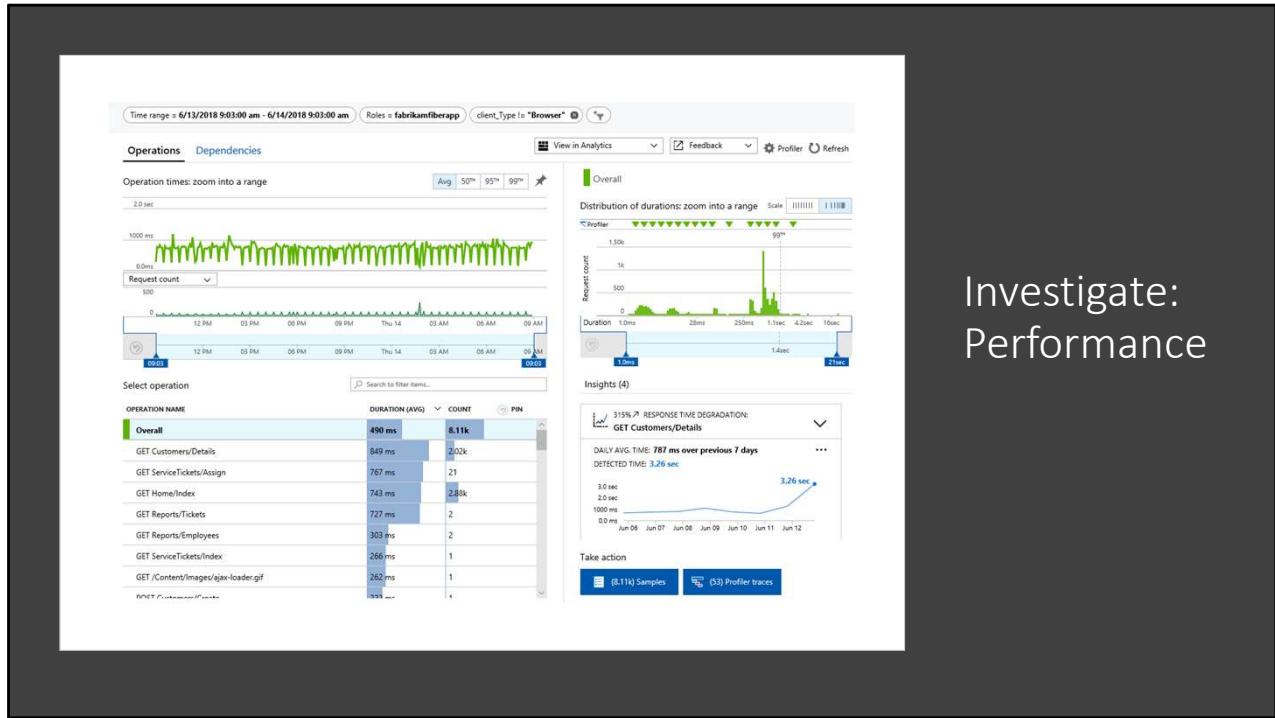
```

System.Web.HttpException:
at System.Web.Mvc.Controller.HandleUnknownAction (System.Web.Mvc, Version=5.2.3.0)
at System.Web.Mvc.Controller+<>c.<BeginExecuteCore>b__152_1 (System.Web.Mvc.Async.AsyncResultWrapper+WrappedAsyncResult`1<System.Threading.Tasks.Task> result)
at System.Web.Mvc.Controller.EndExecuteCore (System.Web.Mvc, Version=5.2.3.0)
at System.Web.Mvc.Async.AsyncResultWrapper+WrappedAsyncResult`1<System.Threading.Tasks.Task>.EndExecuteCore (System.Web.Mvc, Version=5.2.3.0)
at System.Web.Mvc.Controller.EndExecute (System.Web.Mvc, Version=5.2.3.0)
at System.Web.Mvc.MvcHandler+<>c.<BeginProcessRequest>b__20_1 (System.Web.Mvc.Async.AsyncResultWrapper+WrappedAsyncResult`1<System.Threading.Tasks.Task> result)
at System.Web.Mvc.MvcHandler.EndProcessRequest (System.Web.Mvc, Version=5.2.3.0)
at System.Web.HttpApplication+CallHandlerExecutionStep.System.Web.HttpApplication+HttpHandlerCallStep.Invoke (System.Web.HttpApplication, System.EventArgs)
at System.Web.HttpApplication.ExecuteStep (System.Web, Version=4.0)
    
```

Diving into a sample transaction:

- What happened
- When it happened
- Additional diagnostic info; in this case, a call stack for the exception

Investigate: Performance



Similar to the failures page

Quickly find the slowest calls across your application, or just one endpoint

Some smart detection here, seeing response times jump 300% over the past week

Investigate: Troubleshooting guides

The screenshot shows the 'AlbumViewerNetCore - Troubleshooting guides (preview)' page in Application Insights. The left sidebar includes links for Overview, Activity log, Access control (IAM), Tags, and Diagnose and solve problems. Under 'Investigate', there are links for Application map, Smart Detection, Live Metrics Stream, Search, Availability, Failures, Performance, Servers, Browser, and Troubleshooting guides (preview). The main content area displays troubleshooting guides for Request Failures, Dependency Failures, Exceptions, and Performance. A 'Did you know?' section at the bottom right provides information about contributing workbook templates.

New feature in preview

Out of the box troubleshooting guides for request or dependency failures, exceptions

The screenshot shows the Microsoft Power BI service interface. On the left, there are two workbooks displayed: "Virtual Machines" and "Applications". The "Virtual Machines" workbook contains a hexagonal heatmap visualization and several charts. The "Applications" workbook contains a table and a chart. On the right, there is a navigation pane with sections for "Quick start", "Recently modified workbooks", "Virtual Machines", "Applications", and "Log Analytics workspaces".

Monitoring: Workbooks

Troubleshooting guides are a subset of Workbooks

- Pre-baked templates
- Custom reports
- Cool visualizations

Out of the box reports include:

- Failure analysis
- Performance analysis, including things like the Apdex (Application Performance Index) score
- Usage
- User retention

Monitoring: Alerts

The screenshot shows the Azure Application Insights Alerts page for the 'fabrikamprod' resource. On the left, there's a navigation bar with links like Events, Funnels, User Flows, Retention, Impact, Cohorts, More, Properties, and Work Items. The 'Alerts' link is highlighted. The main area displays three alert rules:

Name	Condition	Status	Target Resource	Target Resource Type	Signal Type
dashtest nrt alert-fabrikamprod	Failed locations >= 2	Enabled	dashtest nrt alert-f...	Availability tests	Metrics
Data volume alert email using ...	billingMeters/telemetrySize GreaterThan 1000	Enabled	FabrikamProd	Application Insights	Metrics
Data Volume Log alert test	billingMeters/telemetrySize GreaterThan 1000	Enabled	FabrikamProd	Application Insights	Metrics

To the right, a sidebar titled 'Configure signal logic' lists various signals with their types and services. One signal, 'Custom log search', is highlighted with a red border.

Create your own alerts based on data (signals) in Application Insights

- Response time
- Queue length

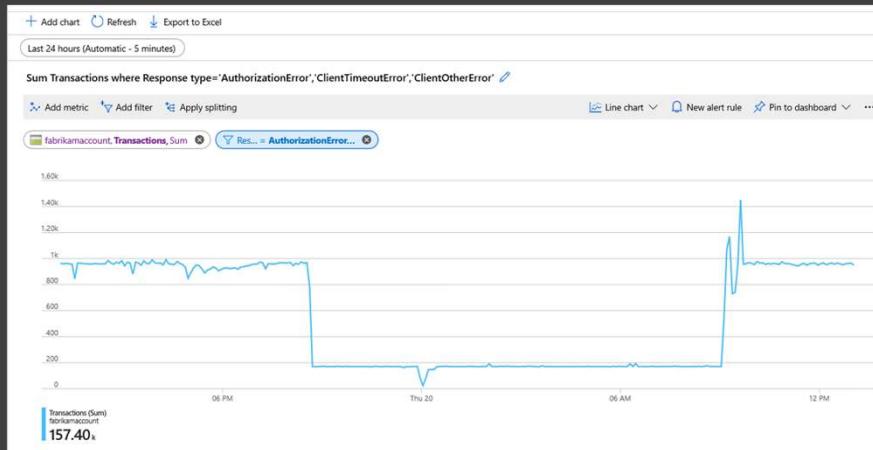
Add actions:

- Sending emails or text messages
- Triggering Azure functions, Logic Apps, or web hooks
- Creating a ticket in ServiceNow or other ITSM software

This page shows an alert based on availability tests failing from more than one location.

Also shows sending alerts based on the amount of data processed by Application Insights.

Monitoring: Metrics



Time-series data – those metrics coming from your application or other Azure resources

Items like

- Application availability
- Browser-reported data, like load time
- Failures, like server or browser exceptions

This shows client errors for past 24 hours, 5 minute intervals

The screenshot shows the Azure Application Insights Kusto Query Editor interface. The query pane contains the following Kusto Query Language (KQL) code:

```
// exception count by problem ID
let start=datetime("2020-01-06T02:57:00.000Z");
let end=datetime("2020-01-07T02:57:00.000Z");
let timeGrain=5m;
let dataset=exceptions
// additional filters can be applied here
| where timestamp > start and timestamp < end
| where client_Type == "Browser"
| dataset
| summarize count_=sum(itemCount), impactedUsers=dcount(user_Id) by problemId
| calculate exception count for all exceptions
| union(dataset
| summarize count_=sum(itemCount), impactedUsers=dcount(user_Id)| extend problemId="Overall")
| where count_ > 0
| order by count_desc
```

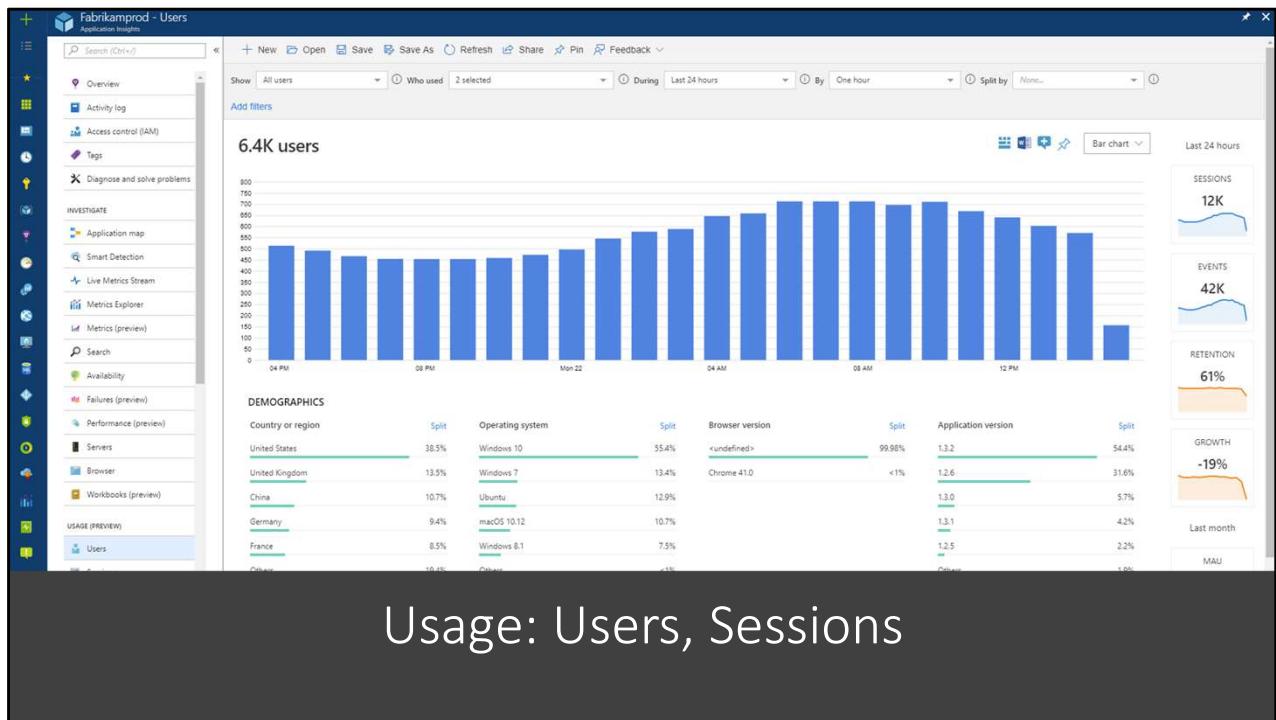
The results pane shows a table titled "Completed" with one row of data:

problemId	count_	impactedUsers
Overall	2	1
obs.subscribe is not a function at ApplicationInsightsErrorHandler.push.../src/app/business/application-insights.errorhandler.ts.ApplicationInsightsErrorHandler.handleError	2	1

At the bottom of the interface, there is a footer bar with navigation icons and a status message: "00:00:02.390".

The raw log data ingested by Application Insights
 Most useful for deep dives, investigations
 Uses Kusto Query Language

This is a sample query to count exceptions by the problem ID (error message)



Demographics like Google Analytics, plus more

See where our users come from, their OS, their browser.

- Super useful to know if and when you can stop supporting IE 11.

The screenshot displays the Microsoft Application Insights portal interface. On the left, a dark sidebar contains navigation links: Usage (selected), Users, Sessions, Events (highlighted with a red box), Funnels, User Flows, Retention, Impact, and Cohorts. The main area is divided into two sections: 'EVENT STATISTICS' and a detailed view of a 'CUSTOMEVENT' named 'Searched'.

EVENT STATISTICS

NAME	UNIQUE USERS	UNIQUE SESSIONS	COUNT
Overall	40	42	90
Azure Overview - Azure Overview	39 (97.5%)	41 (97.6%)	43 (70%)
FAQ - Azure Overview	2 (5%)	2 (4%)	2 (2%)
Searched	1 (2.5%)	1 (2.4%)	4 (6%)
ServiceClick	1 (2.5%)	1 (2.4%)	21 (33%)

CUSTOMEVENT Searched

Custom Event Properties

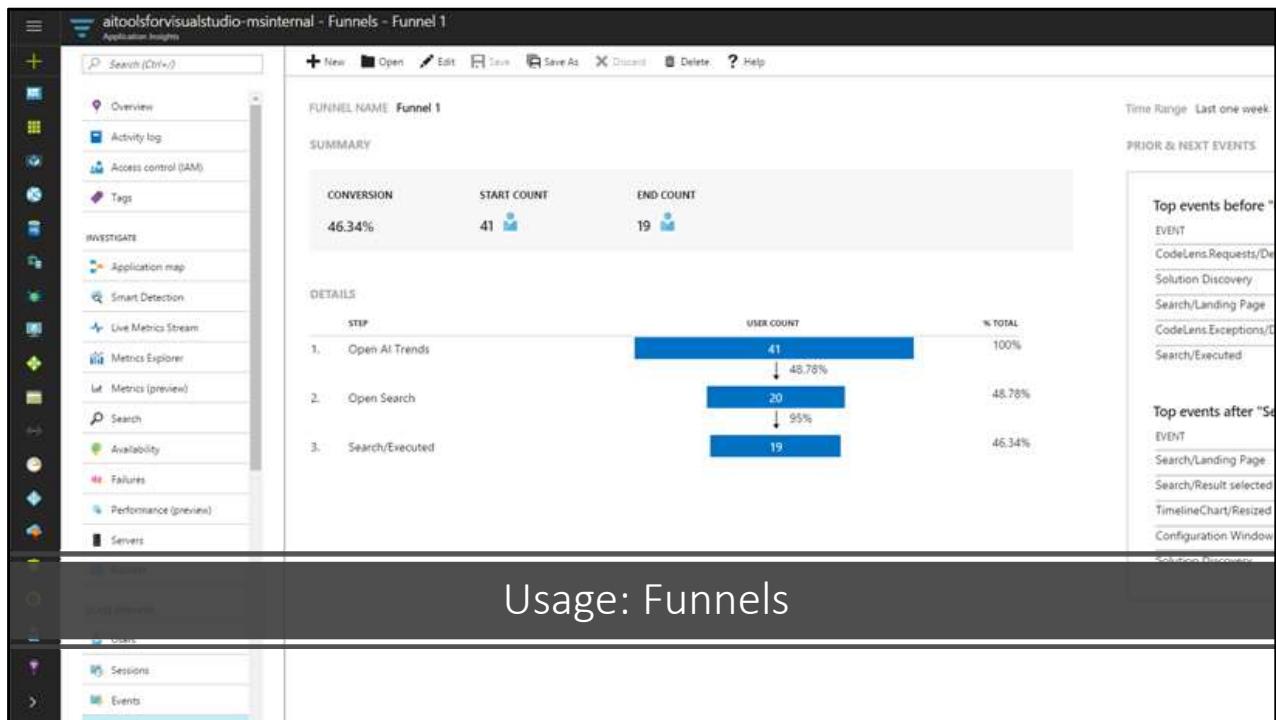
Event time	13-11-2018 10:55:56	...
Event name	Searched	...

Custom Data

AspNetCoreEnvironment	Production	...
term	API Management	...

Server requests, page views, custom events tracked by Application Insights

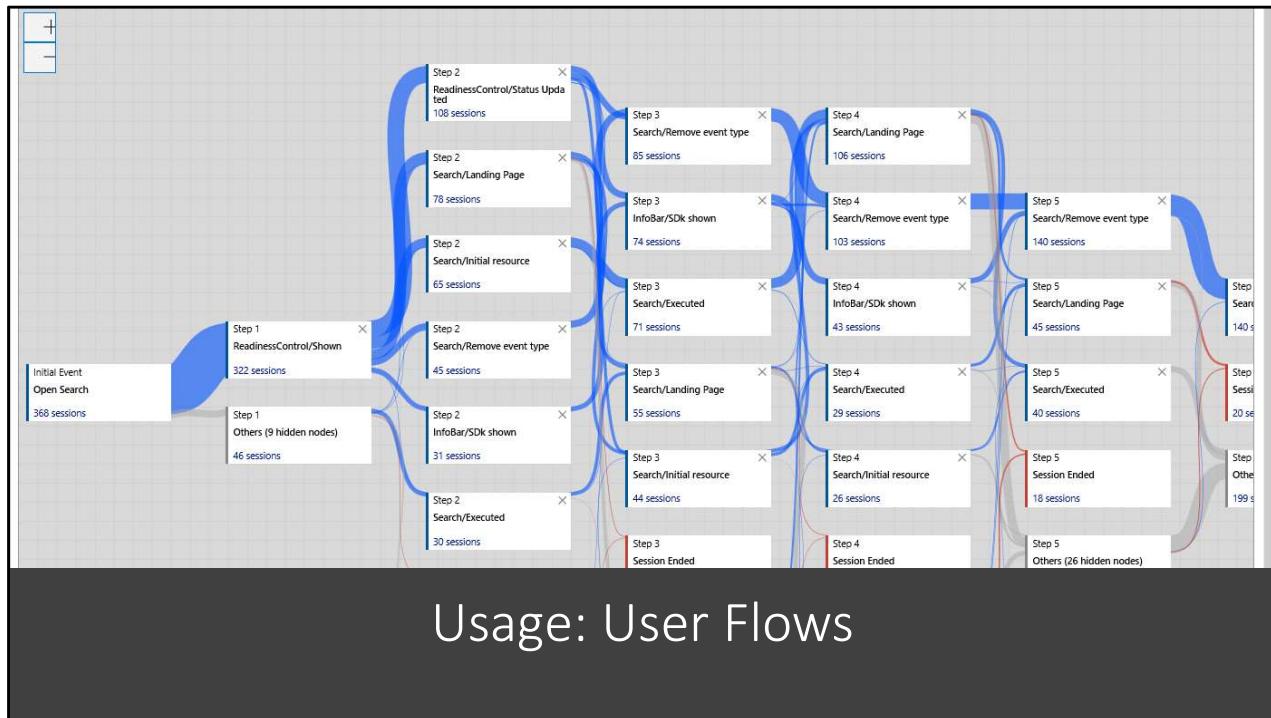
This is a custom search event from the application, includes logging the search term.



Track completion of a workflow by users, understand where they stop in a particular flow.

This shows users opening a trends page through executing a search.

- Of the 41 users that opened the page, only 19 eventually clicked search.

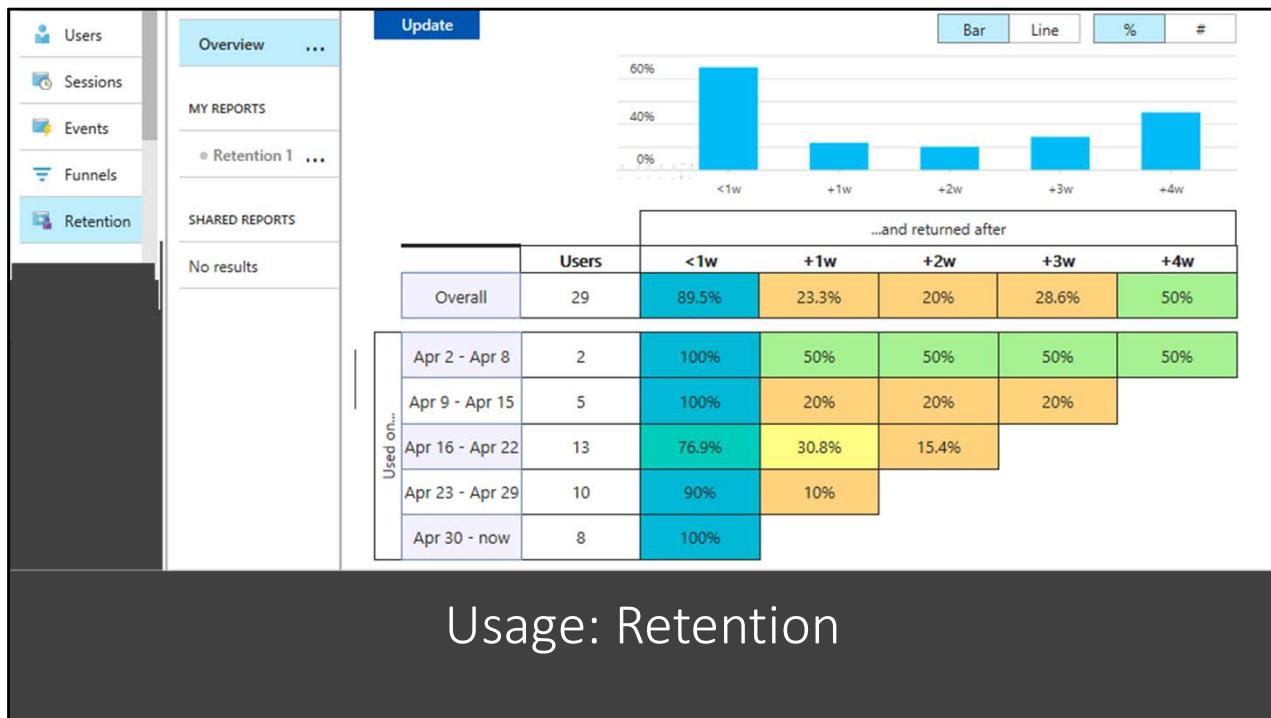


Usage: User Flows

Visualize actions of actual users across many paths.

Example: search

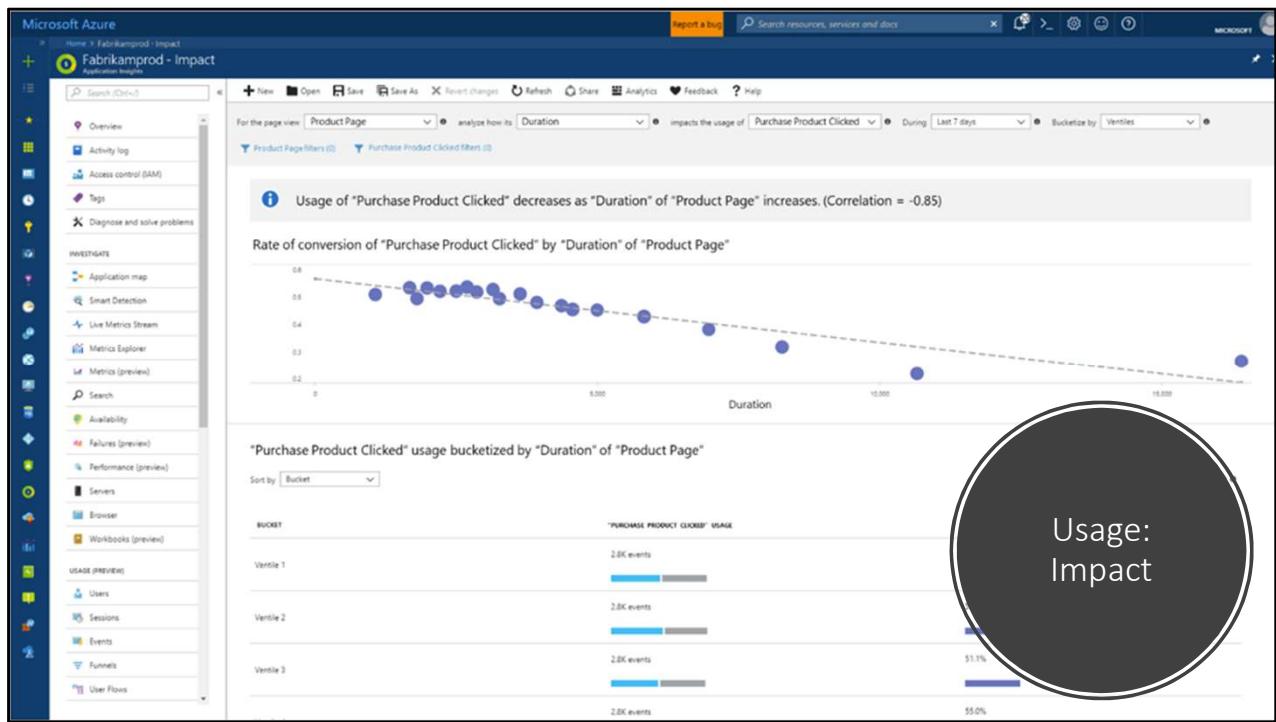
- Initial Event: Open Search
- Most sessions: search control displays
- Afterward
 - Some change the status
 - Some move to another page
 - Others just click search



See how many users return to your application and how frequently.

In this case we see:

- Almost all users come back to our application again in the first week
- But most users don't return to our application after that.



See what variables impact application usage – test our a hypothesis

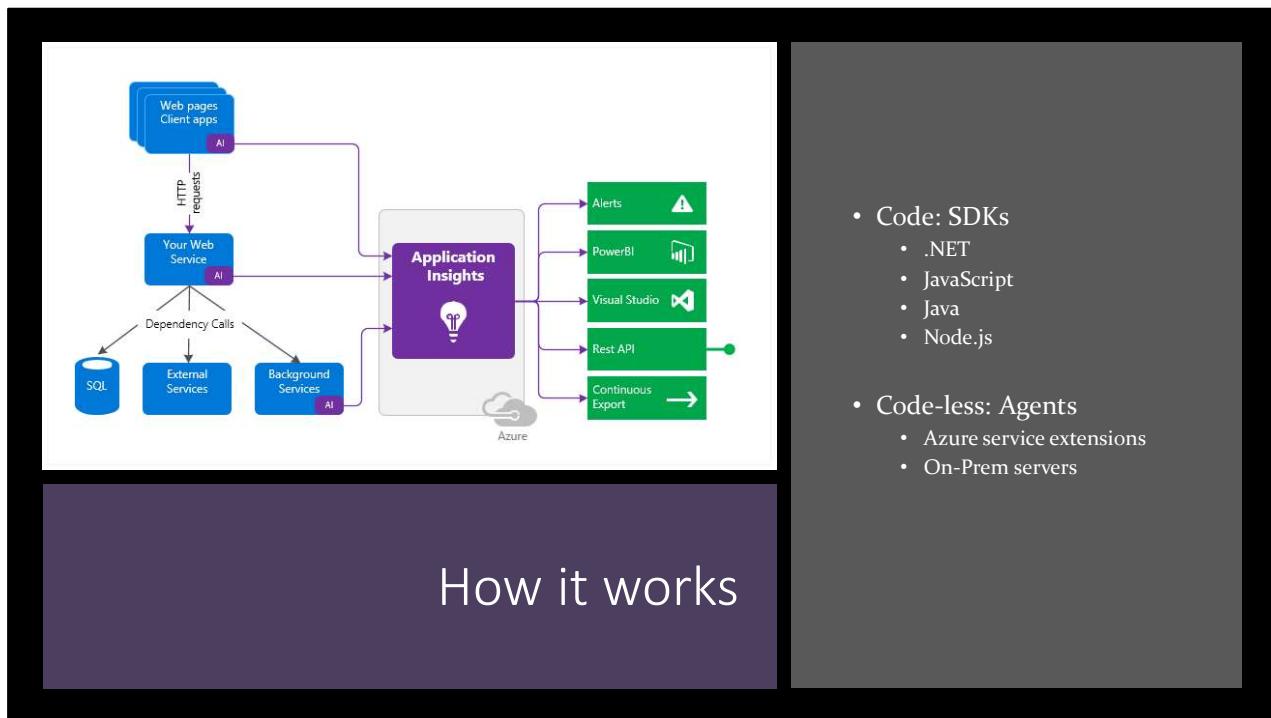
- Example: fewer purchase clicks as the page load increases
- Intuitively makes sense, but how do you quantify your gut?
- Application Insights identifies correlation
- Give real information to stakeholders – help prioritize tech debt or optimizations

The screenshot shows the Microsoft Application Insights interface for managing cohorts. The top navigation bar includes 'Home', 'fabrikamprod - Cohorts - Engaged Users (5+ Days)', and 'Application Insights'. The main title is 'Engaged Users -- by Days Used'. A sidebar on the left lists 'USAGE (PREVIEW)' sections: 'Users', 'Sessions', 'Events', 'Funnels', 'User Flows', 'Retention', 'Impact', and 'Cohorts', with 'Cohorts' currently selected. The main content area contains a descriptive text about engaged users and engagement parameters, with three dropdown menus: 'Activities: All Events and Page Views', 'Period: 28 days', and 'UsedAtleastCustom: 5+ days'. Below this is a table titled 'Usage: Cohorts' with columns 'user_id', 'Previews', and 'Properties'. The table shows two rows: one with user_id 'BCA00' and another with 'arR00'.

Create groups of your users, use them for search and filtering in the usage section.

Cohorts could be:

- Authenticated or unauthenticated users
- Most engaged users based on the number of days they've used your application
- Based on a particular event, like failed requests.



Code: Microsoft SDKs

- Python, Go preview using OpenCensus (now OpenTelemetry)
- RESTful API

NuGet packages, npm packages _or_ JS snippet

- Community packages for frameworks
- Angulartics can combine App Insights for techies, Google Analytics for marketing

Extensions – VM, App Service

- Code: SDKs

- .NET
- JavaScript
- Java
- Node.js

- Code-less: Agents

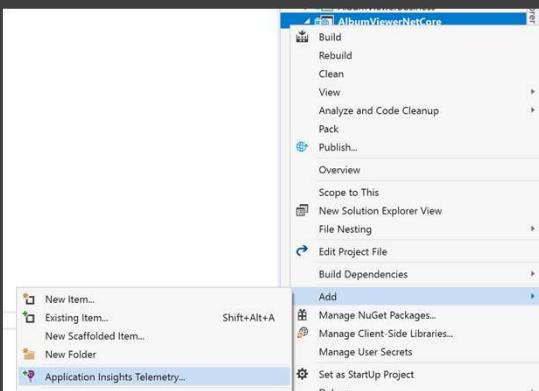
- Azure service extensions
- On-Prem servers

Application Insights libraries

- NuGet packages
 - ASP.NET Core: Microsoft.ApplicationInsights.**AspNetCore**
 - ASP.NET (pre-Core): Microsoft.ApplicationInsights.**Web**
 - Worker/Background: Microsoft.ApplicationInsights.**WorkerService**
 - Requires .NET Standard 2.0
- Node.js npm package: applicationinsights
- JavaScript
 - Snippet in <head> of HTML
 - npm: @microsoft/applicationinsights-web

The main Microsoft-provided libraries

Adding Application Insights to ASP.NET Core



Easy in Visual Studio:

1. Right click project
2. Add >> Application Insights Telemetry...
3. Follow wizard

What happens / manual way

1. Add `Microsoft.ApplicationInsights.AspNetCore` NuGet package
2. In your `Startup` class's `ConfigureServices()`...
 - `services.AddApplicationInsightsTelemetry();`
3. Add instrumentation key...
 - `appsettings.json`
 - Environment variable
 - `APPINSIGHTS_INSTRUMENTATIONKEY`
 - `ApplicationInsights:InstrumentationKey`

```
{  
  "ApplicationInsights": {  
    "InstrumentationKey": "putinstrumentationkeyhere"  
  },  
  "Logging": {  
    "LogLevel": {  
      "Default": "Warning"  
    }  
  }  
}
```

<https://docs.microsoft.com/en-us/azure/azure-monitor/app/asp-net-core>

Adding Application Insights to client-side JS

npm:

```
import { ApplicationInsights } from '@microsoft/applicationinsights-web'

const appInsights = new ApplicationInsights({ config: {
  instrumentationKey: 'YOUR_INSTRUMENTATION_KEY_Goes_HERE'
  /* ...Other Configuration Options... */
} });
appInsights.loadAppInsights();
appInsights.trackPageView(); // Manually call trackPageView to establish the current user/session/pageview
```

Snippet:

```
<script type="text/javascript">
var sdkInstance="appInsightsSDK";window[sdkInstance] = "appInsights";var aiName=window[sdkInstance];
{
  instrumentationKey:"INSTRUMENTATION_KEY"
}
>window[aiName]=aisdk,aisdk.queue&&0==aisdk.queue.length&&aisdk.trackPageView({});</script>
```

<https://docs.microsoft.com/en-us/azure/azure-monitor/app/javascript>

The npm package allows for greater customization, I'll show that in a little bit.

Test drive: AlbumViewerVNext

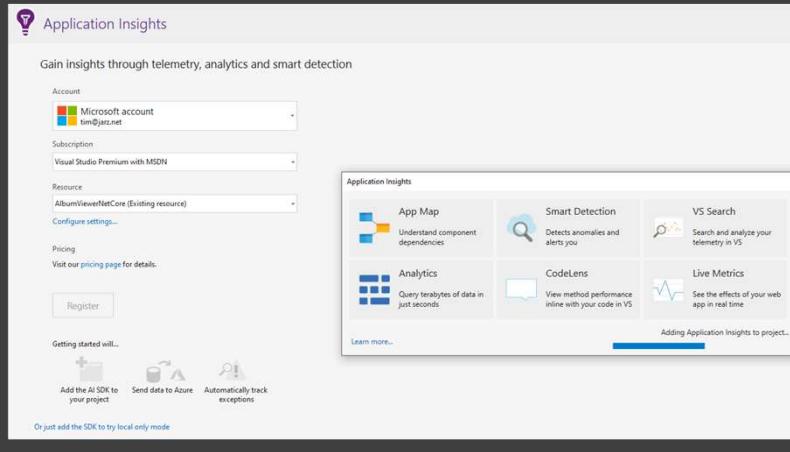
<https://github.com/RickStrahl/AlbumViewerVNext>

- ASP.NET Core
- Angular

The screenshot displays the AlbumViewerVNext application interface. On the left, a grid of album covers from various bands like Motorhead, Anthrax, and Megadeth is shown. Each album cover includes a small thumbnail, the band name, and the album title. To the right of the grid, a detailed view of the Motörhead 'Ace of Spades' album is open. This view includes a larger image of the album cover, a summary text about the album, and a 'Edit Artist' dialog box. The dialog box shows the current artist name as 'Motörhead' and a bio text: 'Motörhead have never sold more than the best rock'n'roll band in the world. They've never sold more than Motörhead. Or the hardest 'n' the toughest. Or the loudest. No, Motörhead are also a legend.' Below the dialog are links for 'Image URL' and 'Amazon URL'. At the bottom of the screen, a preview of the album's tracks is visible.

ASP.NET Core

1. Right-click on csproj, Add >> Application Insights Telemetry
2. Wait...
3. Bam!



Changes made to:

- AlbumViewerNetCore.csproj
 - App Insights Resource ID (Visual Studio integration)
 - Package Reference: Microsoft.ApplicationInsights.AspNetCore
- Startup.cs
 - ConfigureServices method: services.AddApplicationInsightsTelemetry();
- appsettings.json
 - ApplicationInsights:InstrumentationKey

Angular

1. npm i @microsoft/applicationinsights-web
2. Create application-insights.service.ts
 - Wrappers for some Application Insights functions
 - Configuration
3. Update app.component.ts:
 - constructor(private user: UserInfo, private appInsights: ApplicationInsightsService) {

Configuration:

- enableAutoRouteTracking: default = false
 - Automatically can track route changes in a SPA using history API (pushstate, replacestate)
- enableCorsCorrelation
 - Can be problematic, sometimes have to add domains like Auth0 to blacklist

Running the code

- ASP.NET Core project
 - Set to AlbumViewerNetCore profile
 - Press play / F5
- Angular
 - ng serve
 - New tab, open DevTools, <http://localhost:4200>
- Check out Visual Studio
- Check out Azure portal
- Check out DevTools

1. Run backend
2. ng serve
3. Open new tab, F12 dev tools
4. Go to localhost:4200, page loads albums
5. See the *track* calls to App Insights
6. Check out back-end calls in VS
7. Open Azure portal

Visual Studio

- Shows data from current debug session
- Instantaneous, or close enough
- Great for quickly checking out integration

Portal

- All the features!
- Probably spend more time here in day-to-day

DevTools

- Look at all that red!
- Check out the traces sent (track xhr)
- Why don't we see that TypeError?

Enhancing Application Insights integration

- ASP.NET Core project
 - Add middleware (`ApplicationInsightsLoggingMiddleware.cs`)
 - Logs request and response body for HTTP Status Code ≥ 400
 - Add telemetry initializer
 - Adds user from JWT
- Angular project
 - Track the logged-in user
 - Report on unhandled exceptions

1. New Album, no title
2. Show errors in front and back ends
3. Add middleware to back-end
4. New Album, no title
5. Check logs
6. `TypeError` on front-end not being logged
7. Add error handler
8. Show portal
9. Add telemetry initializer to back-end
10. Add user tracking to front-end

Transaction diagnostics

Adding custom telemetry

- Custom user actions, events
- Page views
- Performance metrics
- Exceptions
- Server requests
- Traces
- External dependencies

TrackEvent

- User actions, events in applications
- Aggregated in *Metrics Explorer*
- Individual occurrences in *Search*
- Examples
 - Adding item to cart
 - Winning a game
 - Cancelling a flow

TrackMetric, GetMetric

- Application metrics
- Not attached to events
- Aggregated
- Example
 - Queue length
 - Duration
- TrackMetric
 - Aggregation handled by you
- GetMetric
 - Aggregation handled by SDK
 - C# only

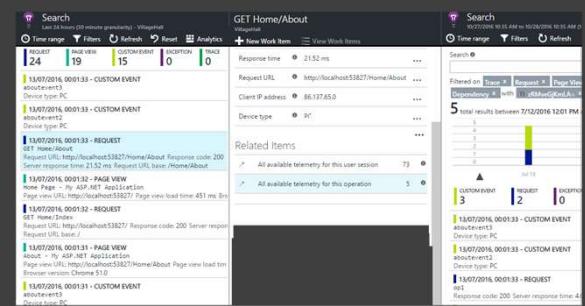
TrackPageView

- Customized page view tracking
- Useful for atypical UX
 - Blades (a la Azure Portal)
 - Tabs inside SPA
- Timing data
 - Set explicitly
 - `startTrackPage()` / `stopTrackPage()`

TrackRequest, Operation contexts

- *Can* manually track HTTP requests
- Recommendation: use operation contexts instead
 - Handles correlation

```
// Establish an operation context and associated telemetry item:  
using (var operation = telemetryClient.StartOperation<RequestTelemetry>("operationName"))  
{  
    // Telemetry sent in here will use the same operation ID.  
    ...  
    telemetryClient.TrackTrace(...); // or other Track* calls  
    ...  
  
    // Set properties of containing telemetry item--for example:  
    operation.Telemetry.ResponseCode = "200";  
  
    // Optional: explicitly send telemetry item:  
    telemetryClient.StopOperation(operation);  
  
} // When operation is disposed, telemetry item is sent.
```



TrackException

- Manually send an exception:

```
try
{
    ...
}
catch (Exception ex)
{
    telemetry.TrackException(ex);
}
```

- Automagically handled by SDKs, extensions, agents

TrackTrace

- Any extra diagnostic data
 - “breadcrumbs”
 - Request/response body
 - Max 32,768 characters
- Log adapters
- Message
- Properties (arbitrary key-value pairs)
- Severity

TrackDependency

- Track response, success rates to external code
- Can be handled automatically by SDKs or agents

```
var success = false;
var startTime = DateTime.UtcNow;
var timer = System.Diagnostics.Stopwatch.StartNew();
try
{
    success = dependency.Call();
}
catch(Exception ex)
{
    success = false;
    telemetry.TrackException(ex);
    throw new Exception("Operation went wrong", ex);
}
finally
{
    timer.Stop();
    telemetry.TrackDependency("DependencyType", "myDependency", "myCall", startTime, timer.Elapsed, success);
}
```

Don't forget to `flush()`!

- Telemetry transmission typically batched
 - 30 seconds
 - Full buffer (500 items)
- Add call to `TelemetryClient.Flush()` when shutting down

Some limits...

- Total data per day: 100 GB
 - Free: 5 GB / organization / month
- Data retention: 30 – 730 days
 - Default / no charge: 90 days

Add some extra functionality

Server-side

- Log request/response bodies for errors
- Ensure we've got the user

Client-side

- Track all exceptions
- Ensure we've got the user

Transaction diagnostics

- Distributed applications
 - Queues
 - SQL databases
 - Functions
- See across all components

<https://docs.microsoft.com/en-us/azure/azure-monitor/app/transaction-diagnostics>

Even ones outside your app / ikey

Some of my recommendations

- Separate resources (keys) per code environment
- Validate your tracking code works
- Log everything! (...at first)
 - Lots of log categories in ASP.NET Core
 - Review logs, then begin tweaking categories and logging levels
- Be aware of sampling
- Azure App Services: enable Application Insights settings
 - Profiler, Snapshot Debugger, SQL commands
- VS Enterprise users: add SnapshotCollector NuGet package

Separate instrumentation keys:

- Different, independent applications - Use a separate resource and ikey for each app.
- Multiple components or roles of one business application - Use a [single shared resource](#) for all the component apps. Telemetry can be filtered or segmented by the cloud_RoleName property.
- Development, Test, and Release - Use a separate resource and ikey for versions of the system in 'stamp' or stage of production.
- A | B testing - Use a single resource. Create a TelemetryInitializer to add a property to the telemetry that identifies the variants.



Thanks for spending some time with me during CodeMash.

Thanks to CodeMash for having me speak.

Appreciate any feedback on the session in the conference app.

Want to know more? Eric Potter from Aptera (a sponsor) is giving a session on Friday, 1:30 in Nile.

Advanced customization

- Add/remove properties with telemetry initializers
- Reduce volume with sampling or telemetry processors

Custom Telemetry Initializers

- Implement `ITelemetryInitializer`

- Add additional data to telemetry
- Override default behavior
 - Default: HTTP 400s = failure
 - Override: HTTP 400s = success
- Telemetry initializers always run
 - All registered will be called

```
public class MyTelemetryInitializer : ITelemetryInitializer
{
    public void Initialize(ITelemetry telemetry)
    {
        var requestTelemetry = telemetry as RequestTelemetry;
        // Is this a TrackRequest() ?
        if (requestTelemetry == null) return;
        int code;
        bool parsed = Int32.TryParse(requestTelemetry.ResponseCode, out code);
        if (!parsed) return;
        if (code >= 400 && code < 500)
        {
            // If we set the Success property, the SDK won't change it:
            requestTelemetry.Success = true;

            // Allow us to filter these requests in the portal:
            requestTelemetry.Properties["Overridden400s"] = "true";
        }
        // else leave the SDK to set the Success property
    }
}
```

Custom Telemetry Processors

- Implement `ITelemetryProcessor`
- Processes / filters all telemetry
 - Includes base request, dependency telemetry
- Can impact statistics
 - Preferred: use sampling
- Only the first might be called

```
public void Process(ITelemetry item)
{
    var request = item as RequestTelemetry;
    if (request != null && request.ResponseCode.Equals("401", StringComparison.OrdinalIgnoreCase))
    {
        // To filter out an item, return without calling the next processor.
        return;
    }

    // Send everything else
    this.Next.Process(item);
}
```

Logging framework support

- ILogger: Microsoft.Extensions.Logging.ApplicationInsights
- NLog: Microsoft.ApplicationInsights.NLogTarget
- Log4Net: Microsoft.ApplicationInsights.Log4NetAppender
- System.Diagnostics: Microsoft.ApplicationInsights.TraceListener
- Microsoft.ApplicationInsights.DiagnosticSourceListener
- Microsoft.ApplicationInsights.EtwCollector
- Microsoft.ApplicationInsights.EventSourceListener