
+5, Insightful: Azure Application Insights to improve your code
CODEMASH
2020
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?*

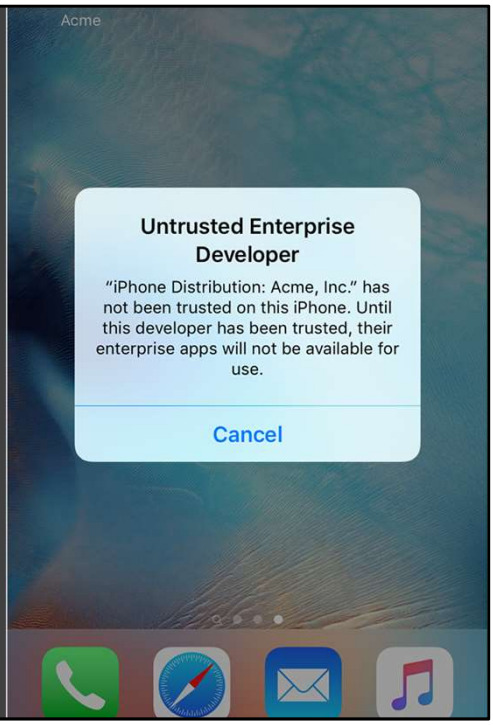
Ops

- *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

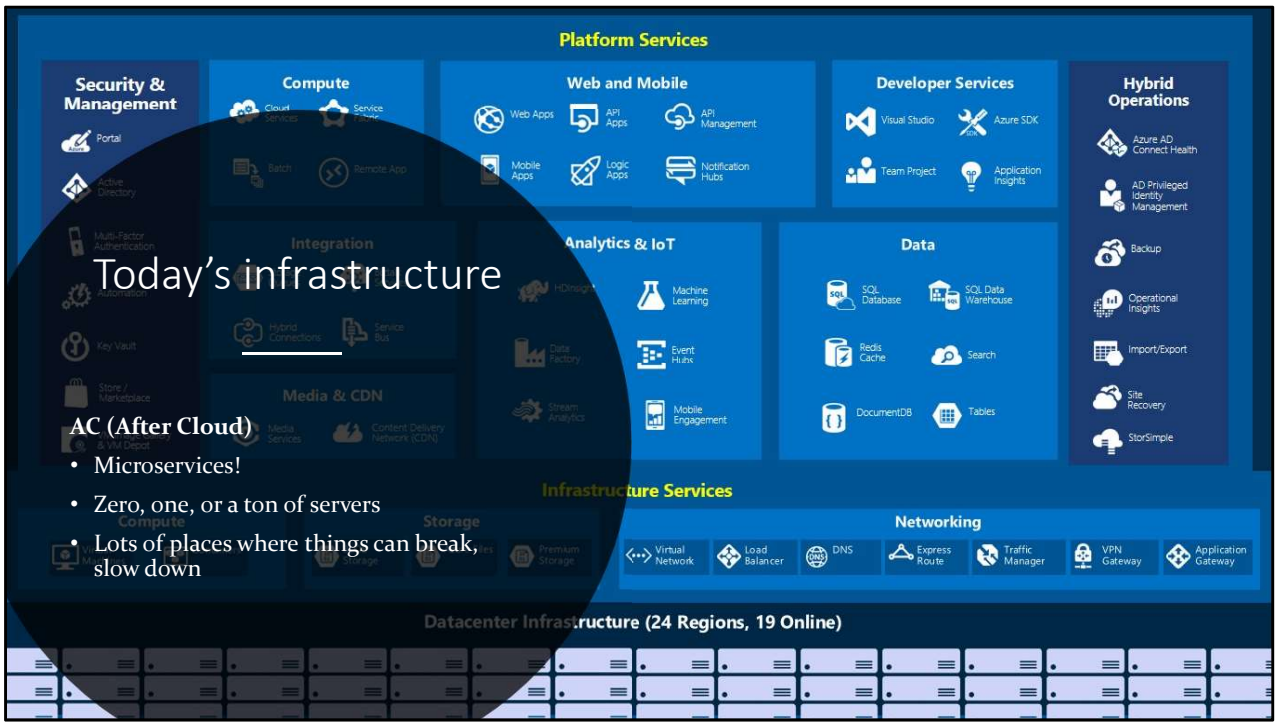
Work on an e-commerce project. Bugs or issues that prevent people buying stuff are bad.

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

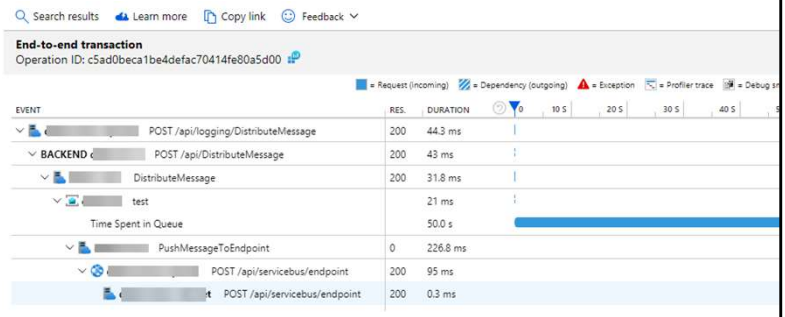
- Tools like Splunk or ELK stack

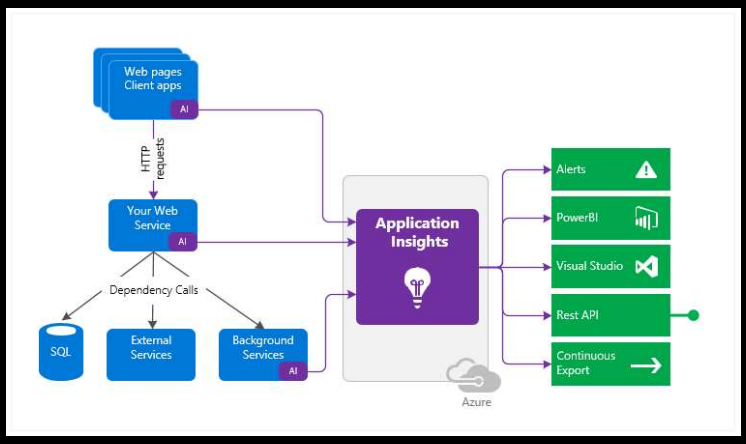
- Eventually, some structured logging
- We still can't agree on the one true way
  - Log4net, Serilog, NLog

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





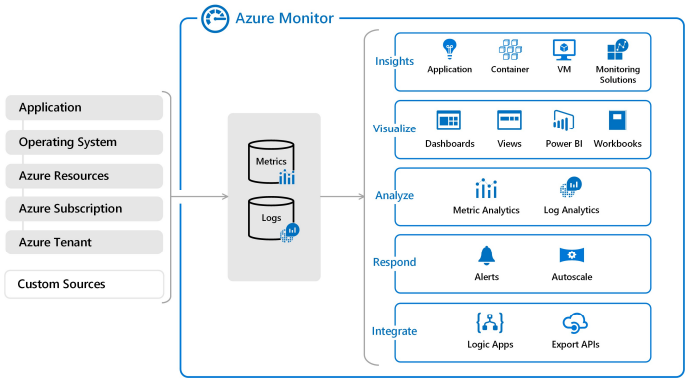
Azure Application Insights

- APM-aaS
- Used on cloud *or* on-prem/hosted apps
- Any stage of dev/test/release
  - Local dev: Visual Studio integration
  - Server(less): Azure portal
- Logs and metrics
  - Exceptions
  - Custom (domain) events

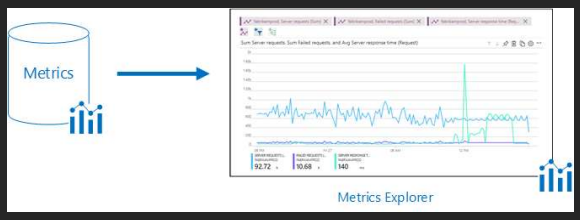
Bigger picture: Azure Monitor

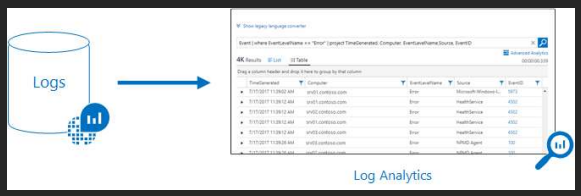
- Suite of services focused on application telemetry

- Collect
- Analyze
- See
- Act



Data points: metrics & logs



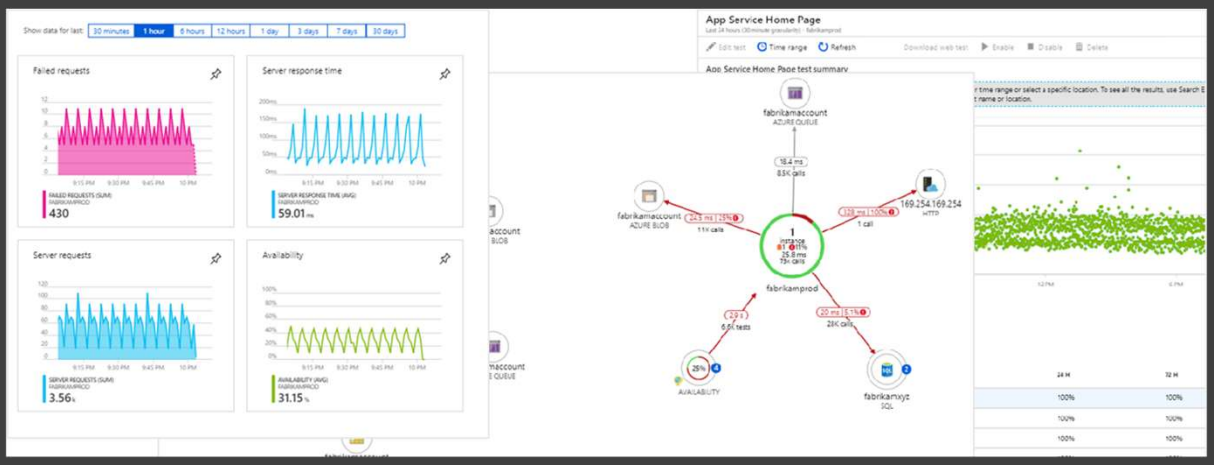


Visualize metrics in Metrics Explorer
- Numeric data: CPU utilization, requests
- Data that can be aggregated

Review logs in Log Analytics
- Event traces, exceptions
- Structured (or not)

Lots of data, but we want _information_

Actionable information | Application Insights



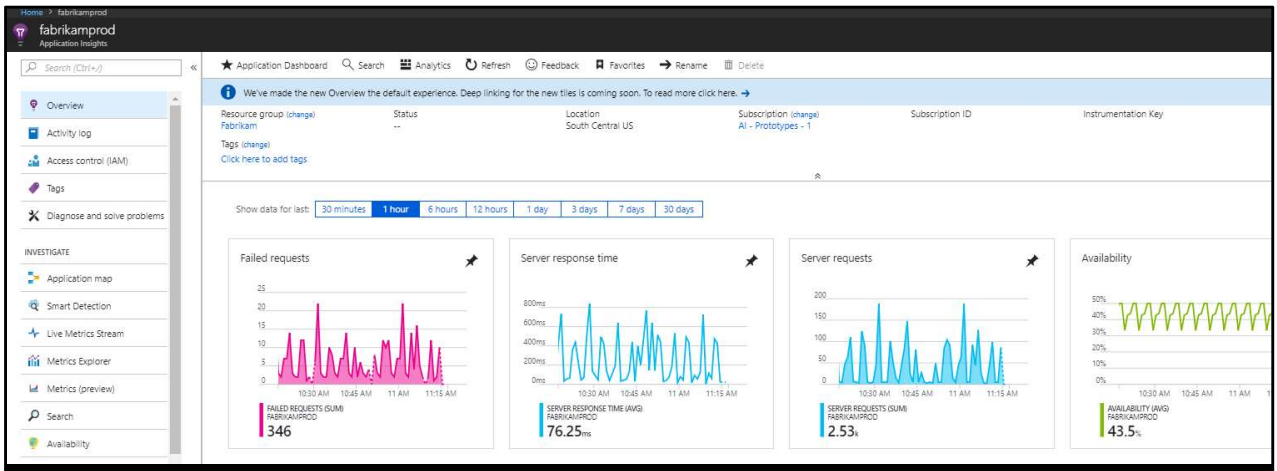
Quickly see issues

- 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

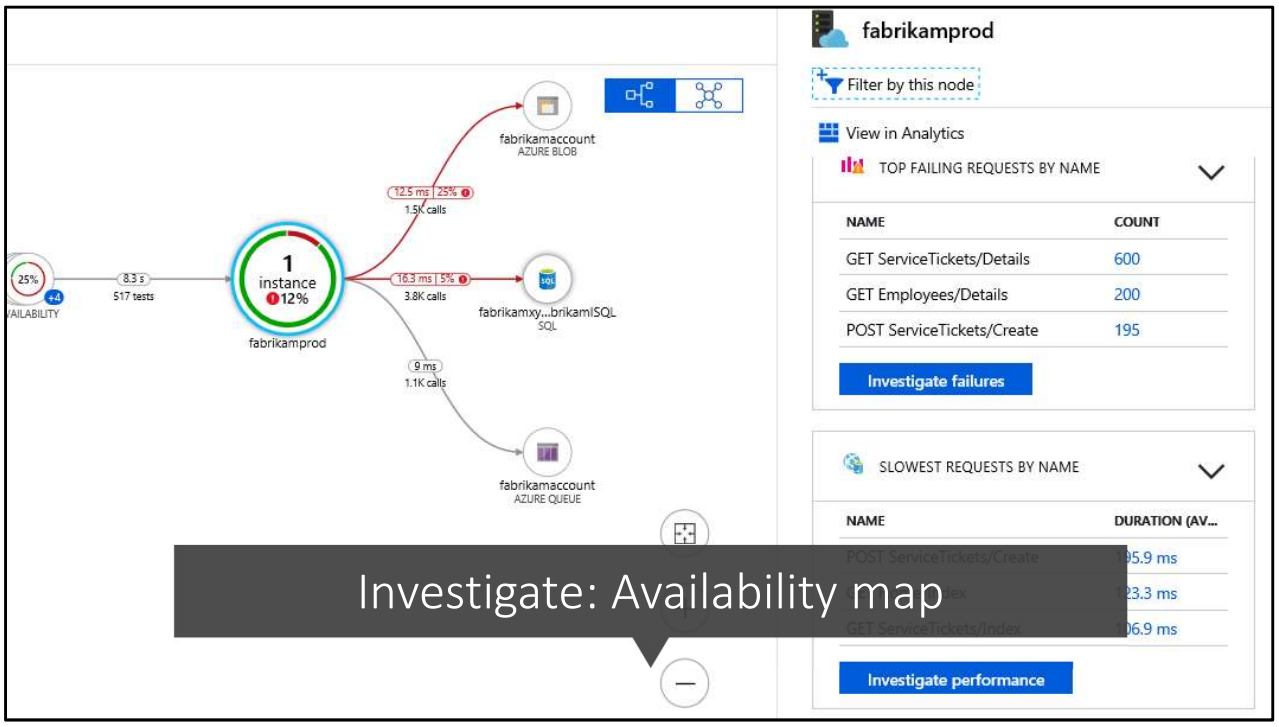


Overview

Landing page in Application Insights resource

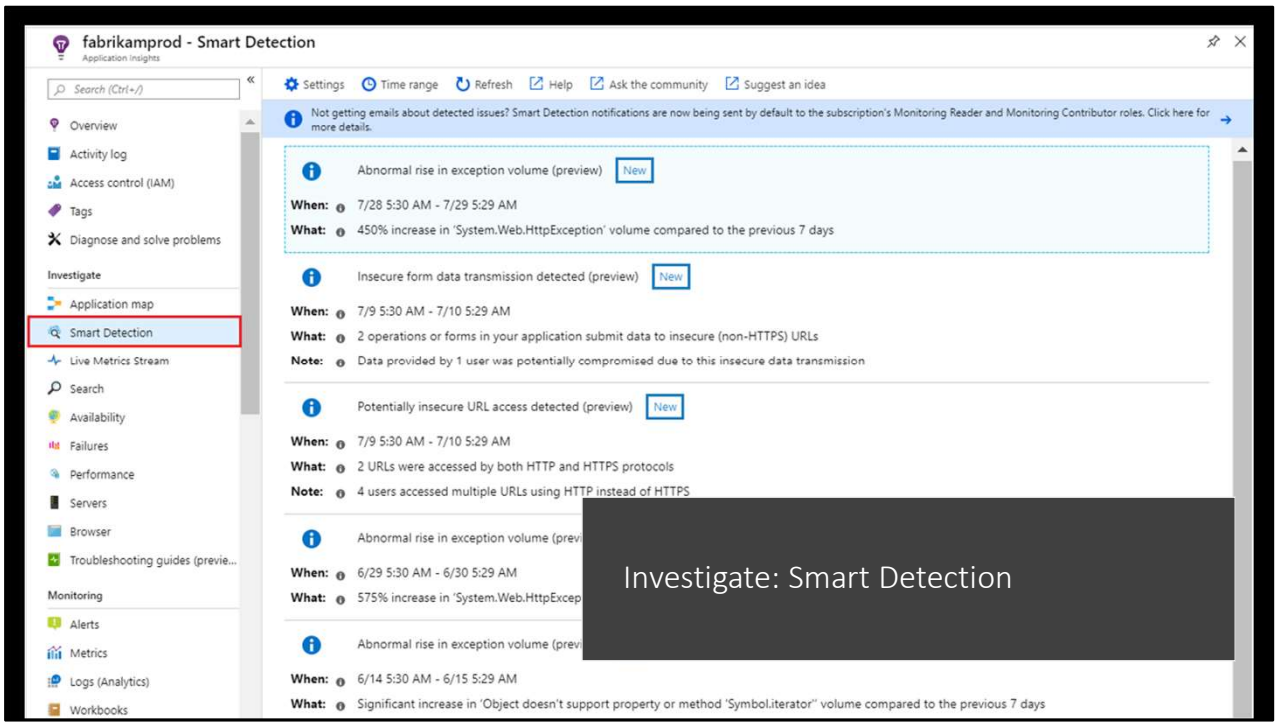
Quickly see some of the most important metrics for web applications

- Failures
- Response times
- Inbound traffic
- Availability



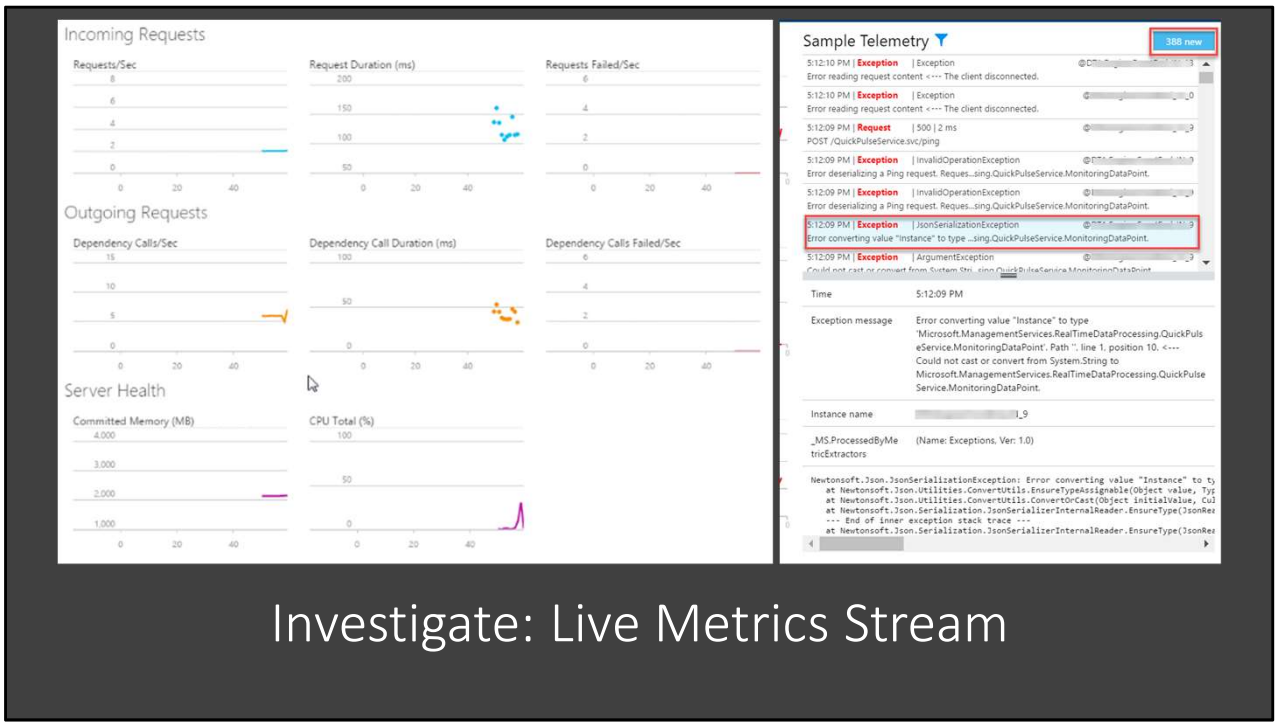
Quickly investigate issues in and outside our control:

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



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

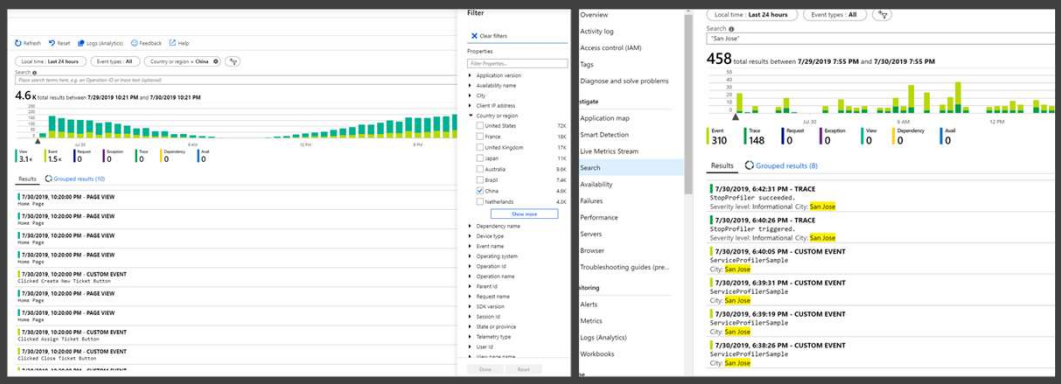


Investigate: Live Metrics Stream

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

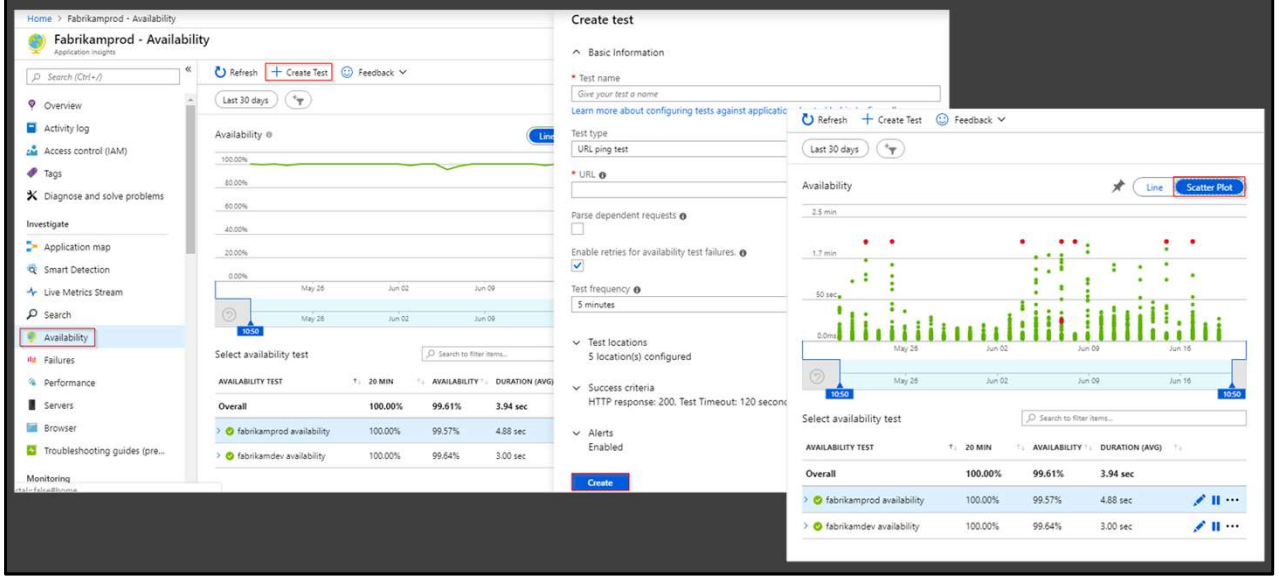


Search and filter across all of the telemetry processed

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

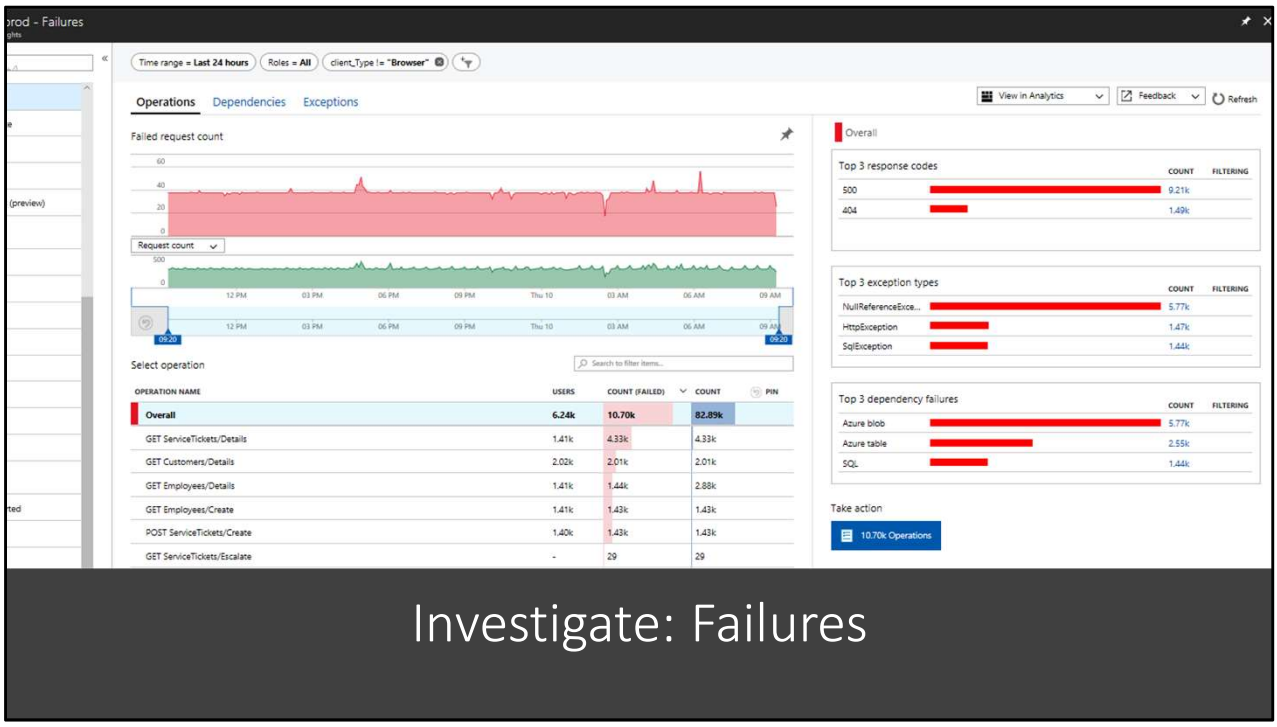
Investigate: Availability



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.



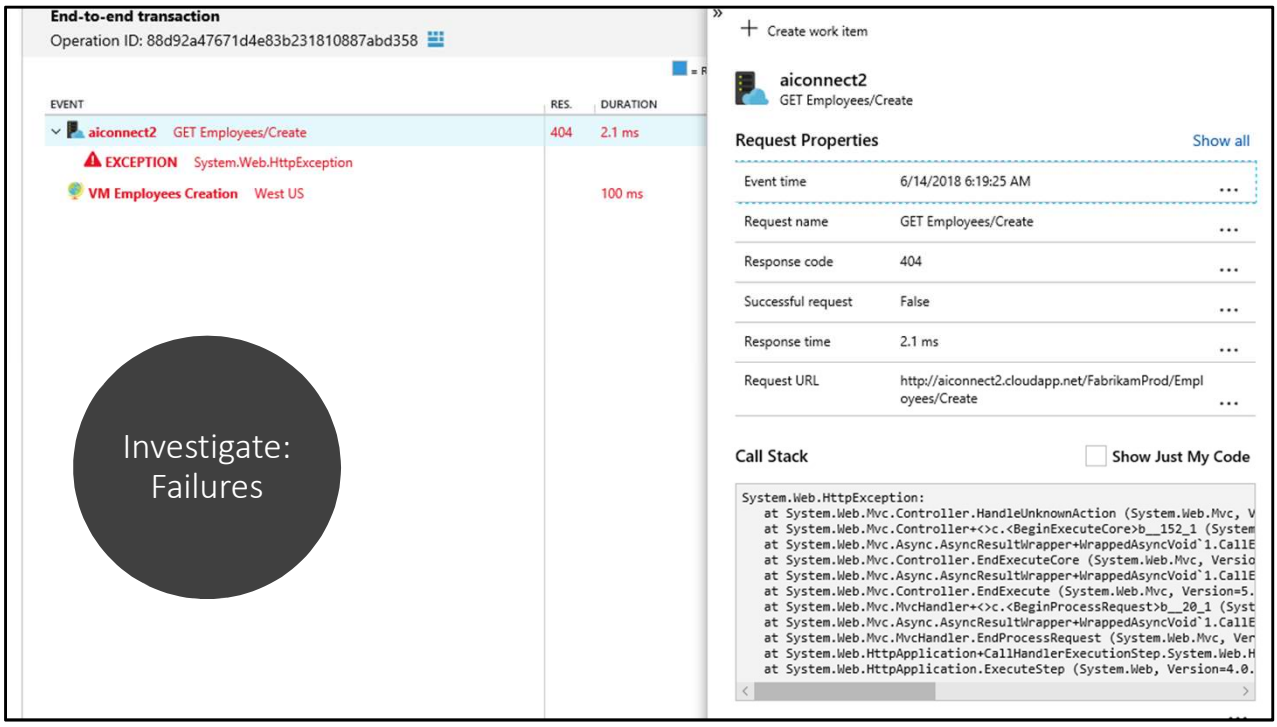
Investigate: Failures

Where I spend most of my time in App Insights

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

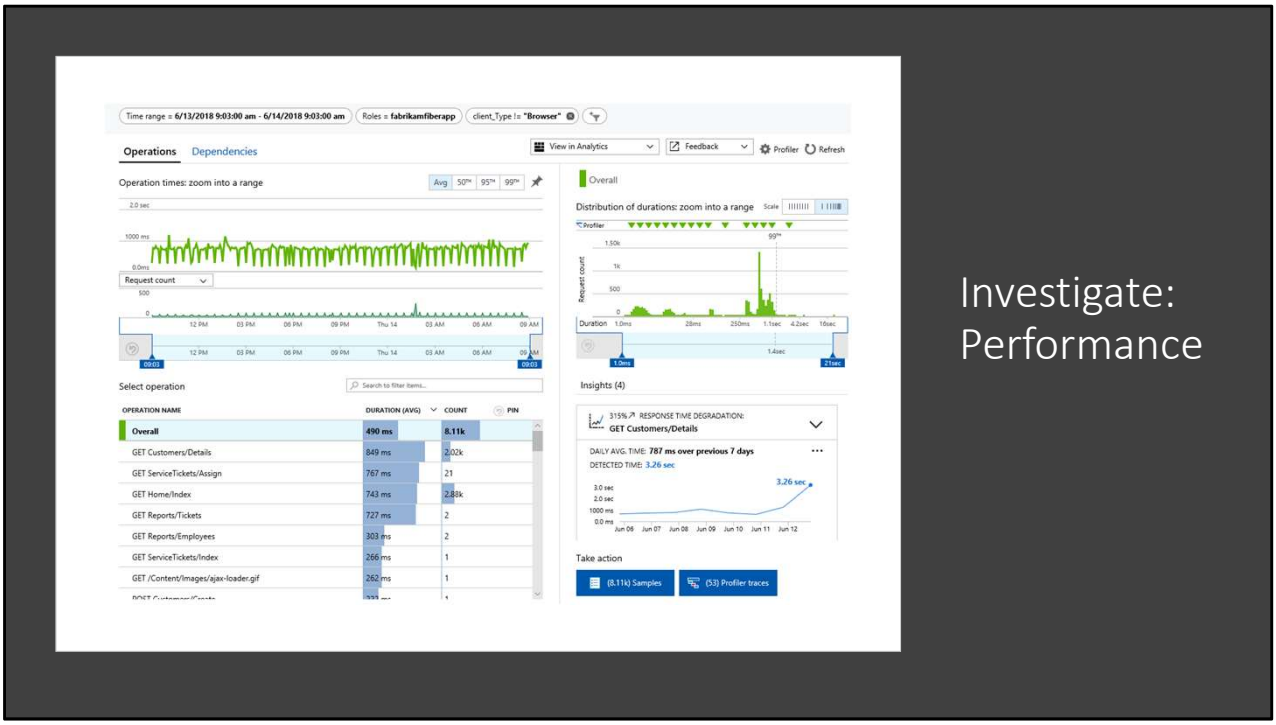
Can filter operations:
- Only the 500 errors since 404's are common
- Only the NullReferenceExceptions
- SQL failures

Includes both server and browser errors



Diving into a sample transaction:

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



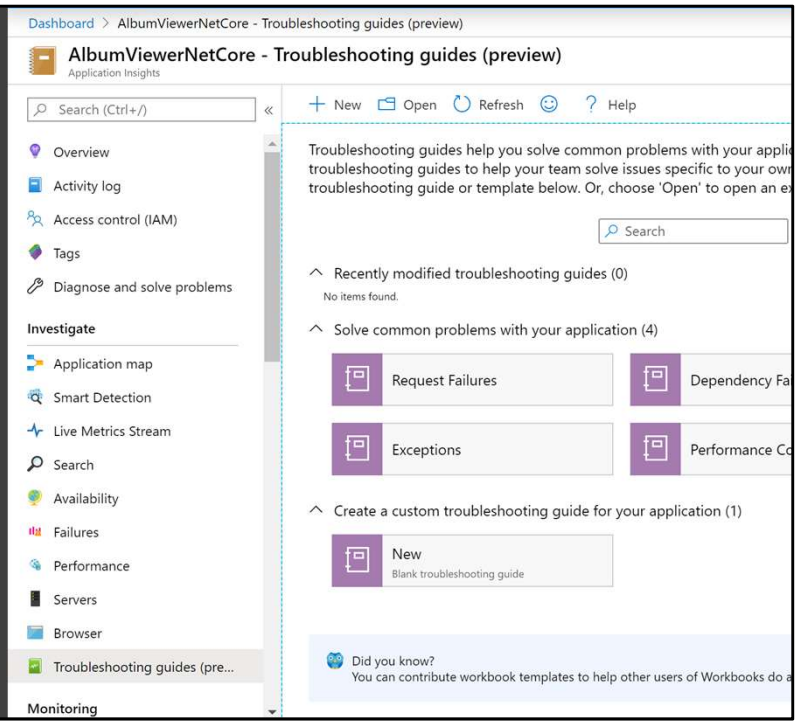
Investigate: Performance

Similar to the failures page

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

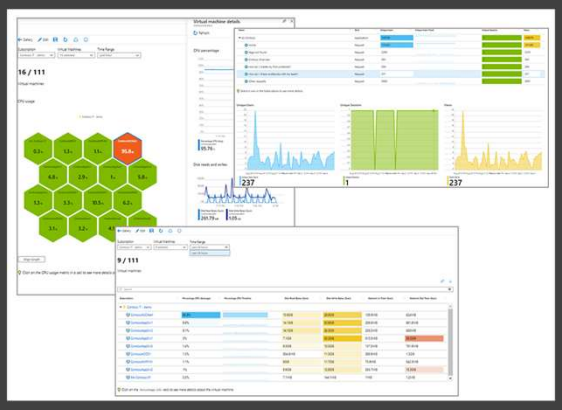
Some smart detection here, seeing response times jump 300% over the past week

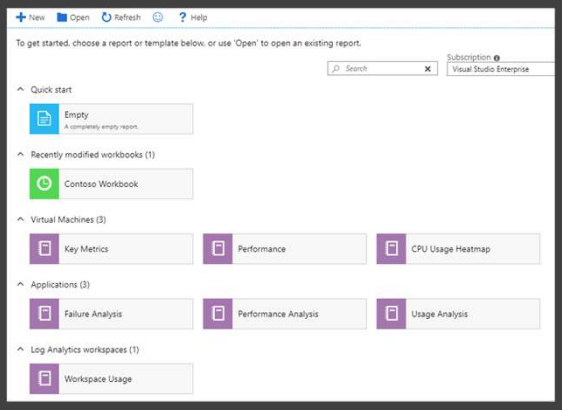
Investigate: Troubleshooting guides



New feature in preview

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





Monitoring: Workbooks

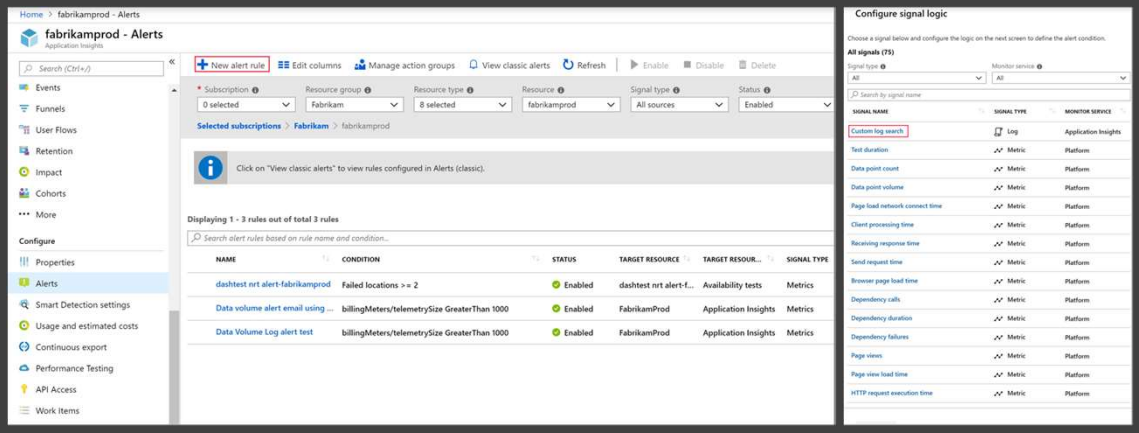
Troubleshooting guides are a subset of Workbooks

- 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

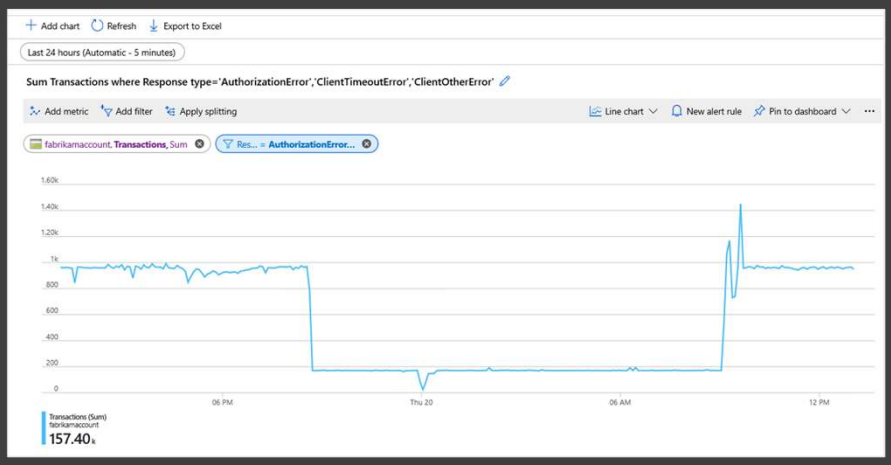


Create your own alerts based on data (signals) in Application Insights
- Response time
- Queue length

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.

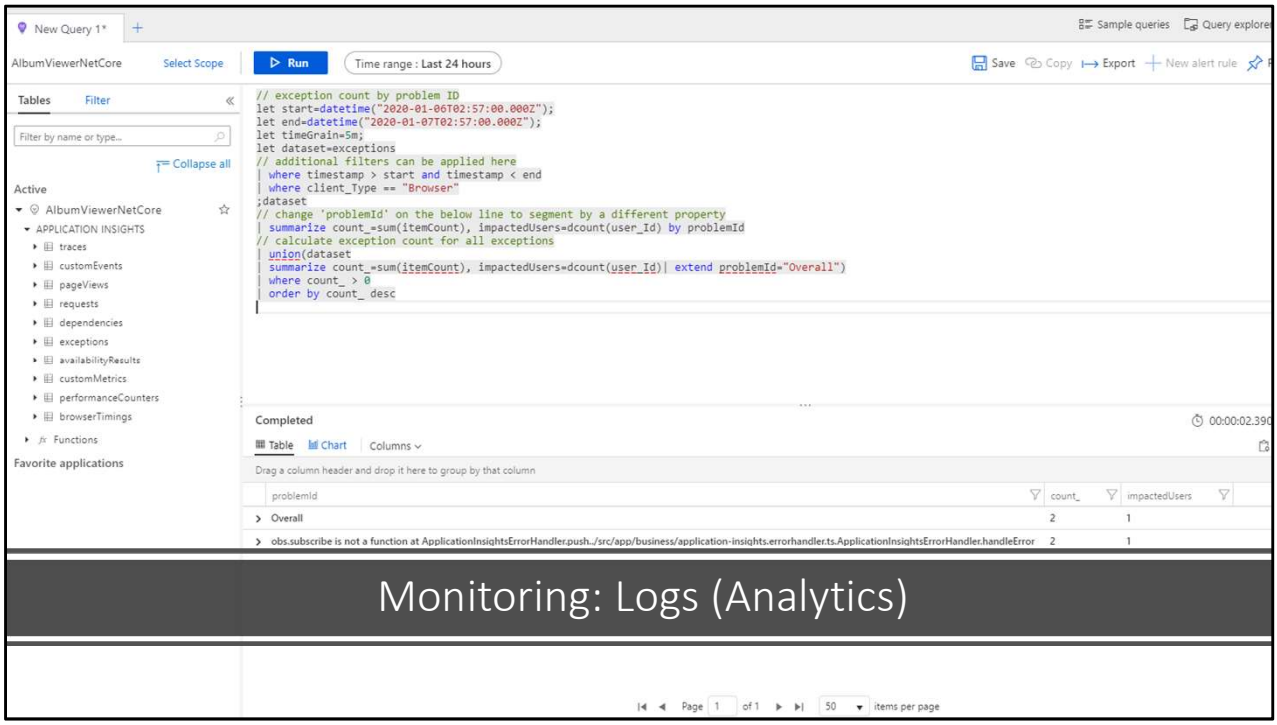
Monitoring: Metrics



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

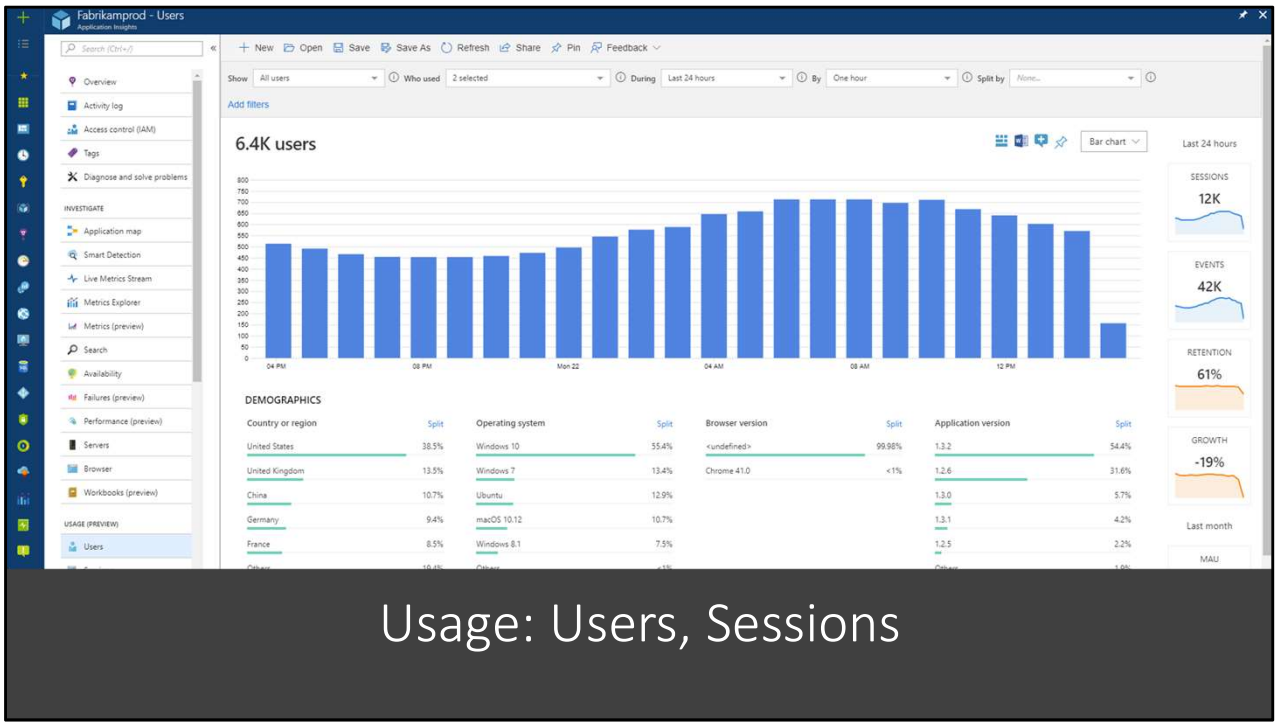
This shows client errors for past 24 hours, 5 minute intervals



Monitoring: Logs (Analytics)

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)

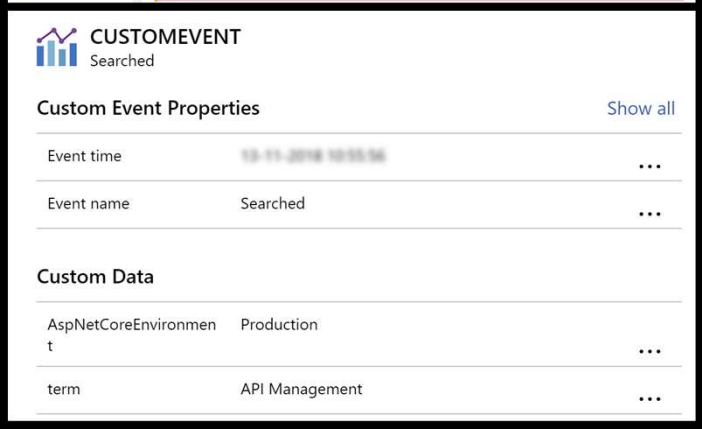


Usage: Users, Sessions

Demographics like Google Analytics, plus more

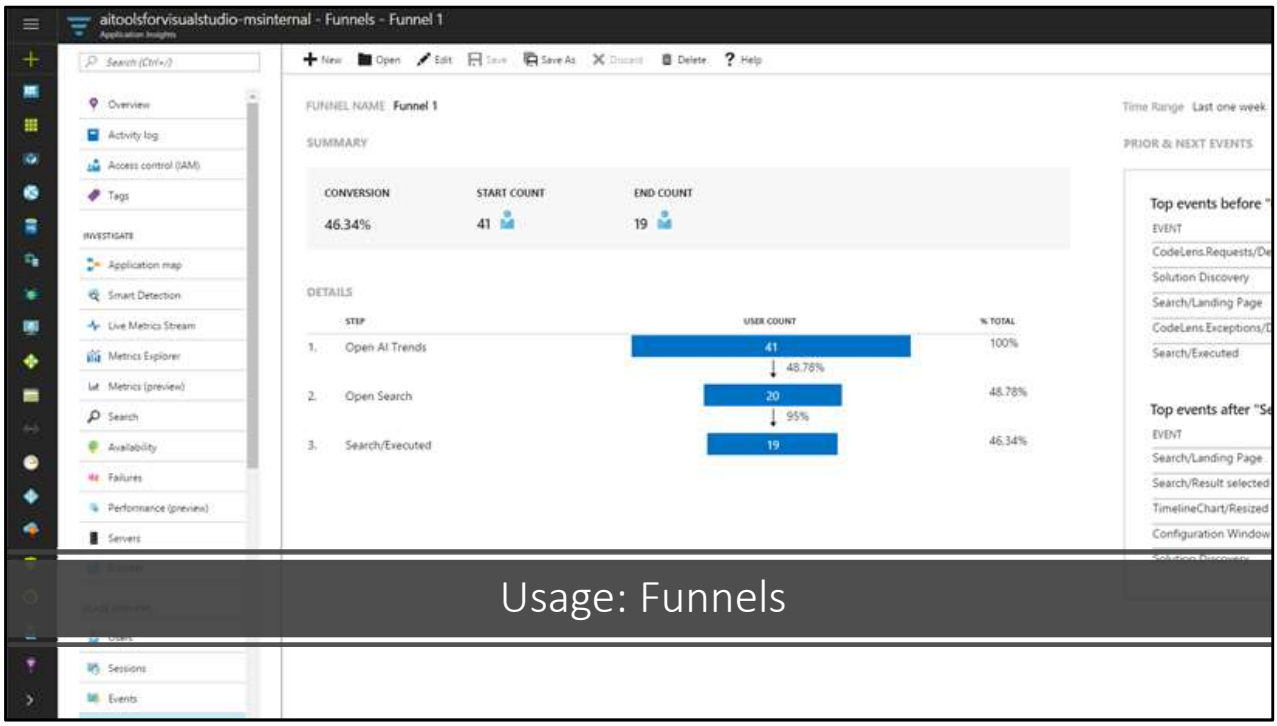
See where our users come from, their OS, their browser.
- Super useful to know if and when you can stop supporting IE 11.

Usage: Events



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

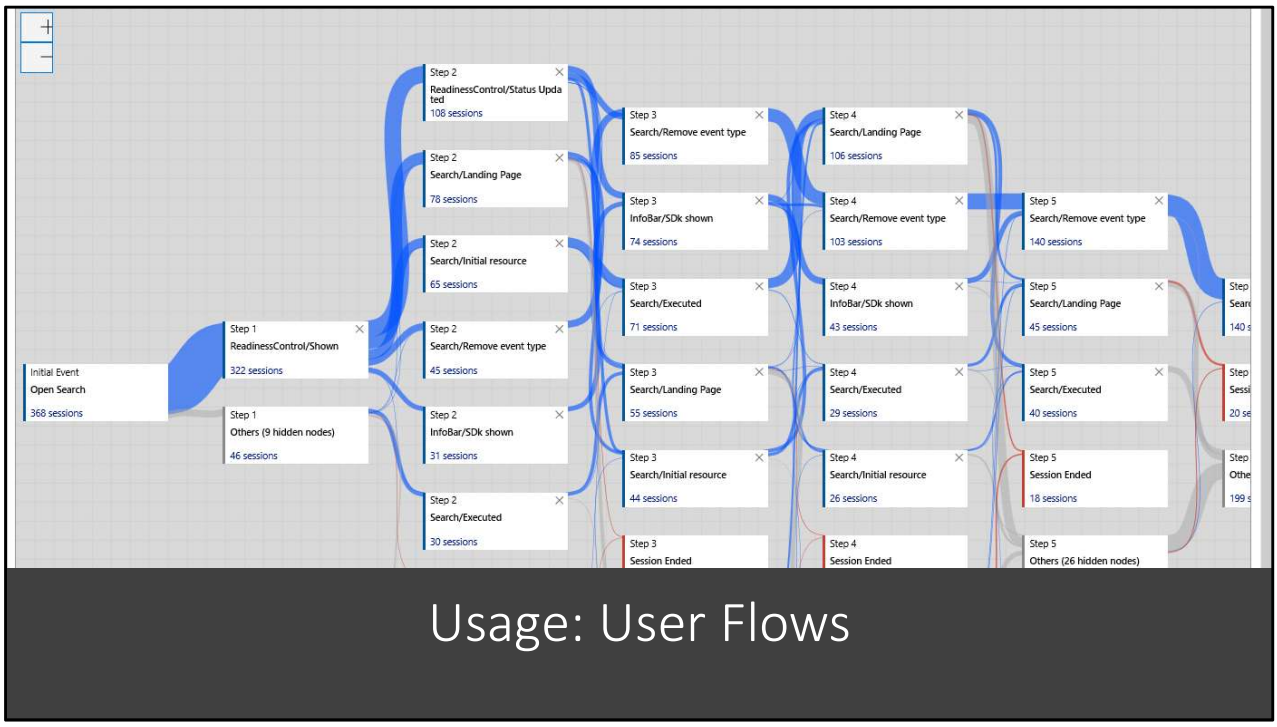
This is a custom search event from the application, includes logging the search term.



Usage: Funnels

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.

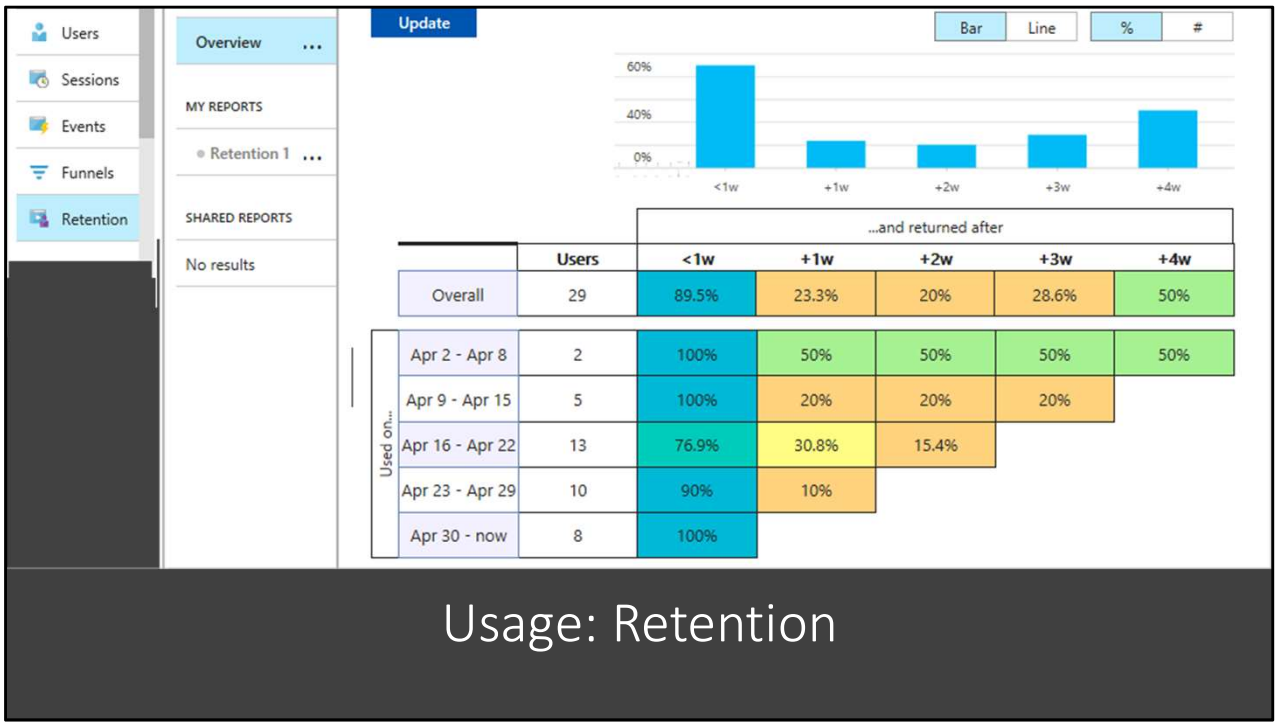


Usage: User Flows

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

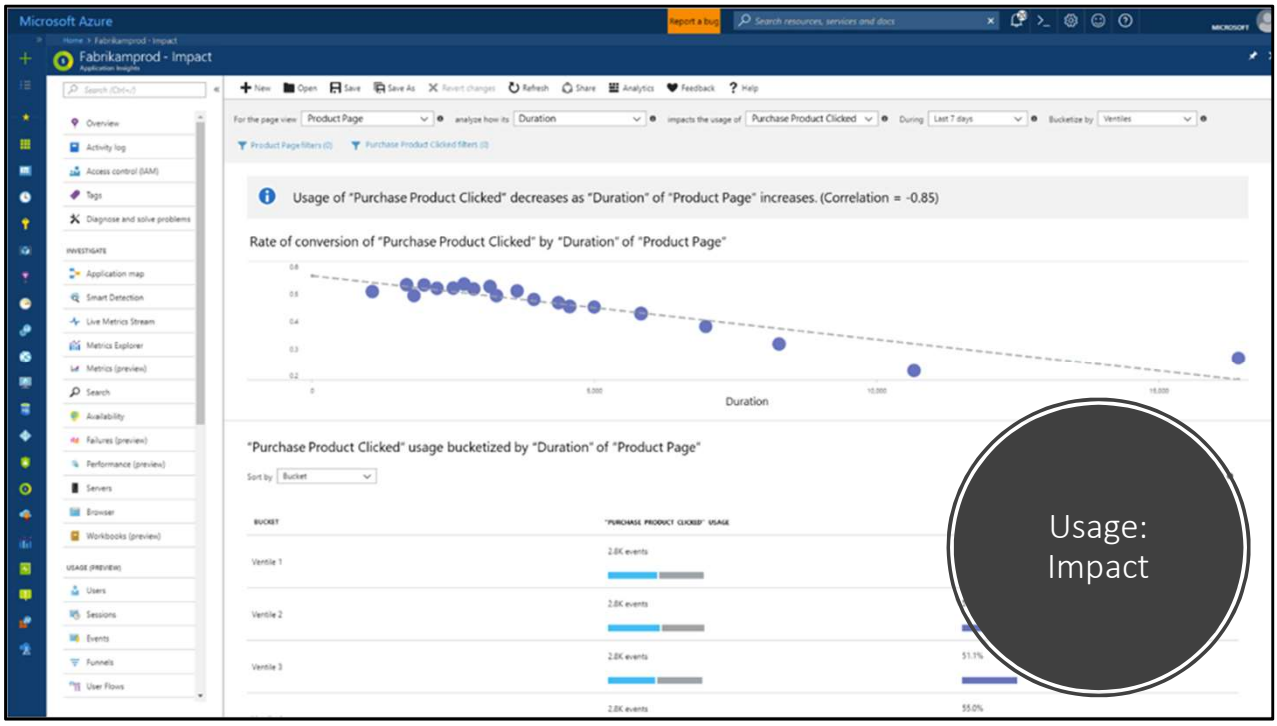


Usage: Retention

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

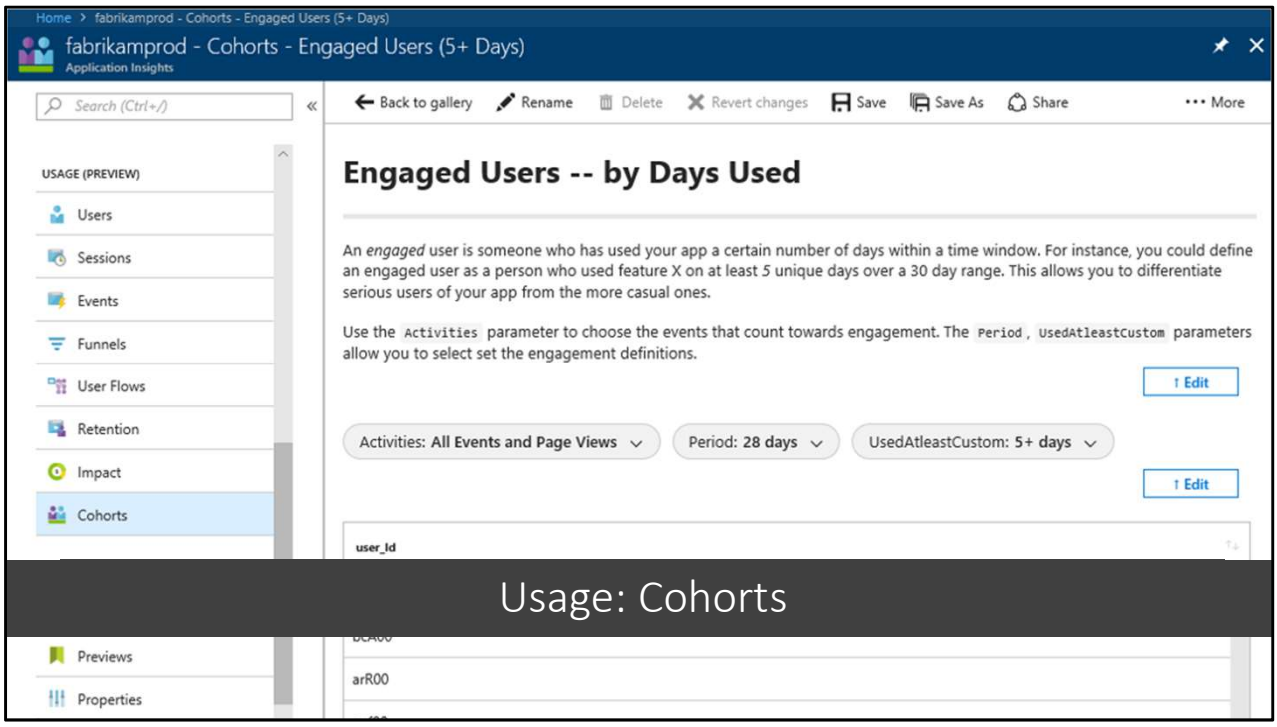
In this case we see:

- 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

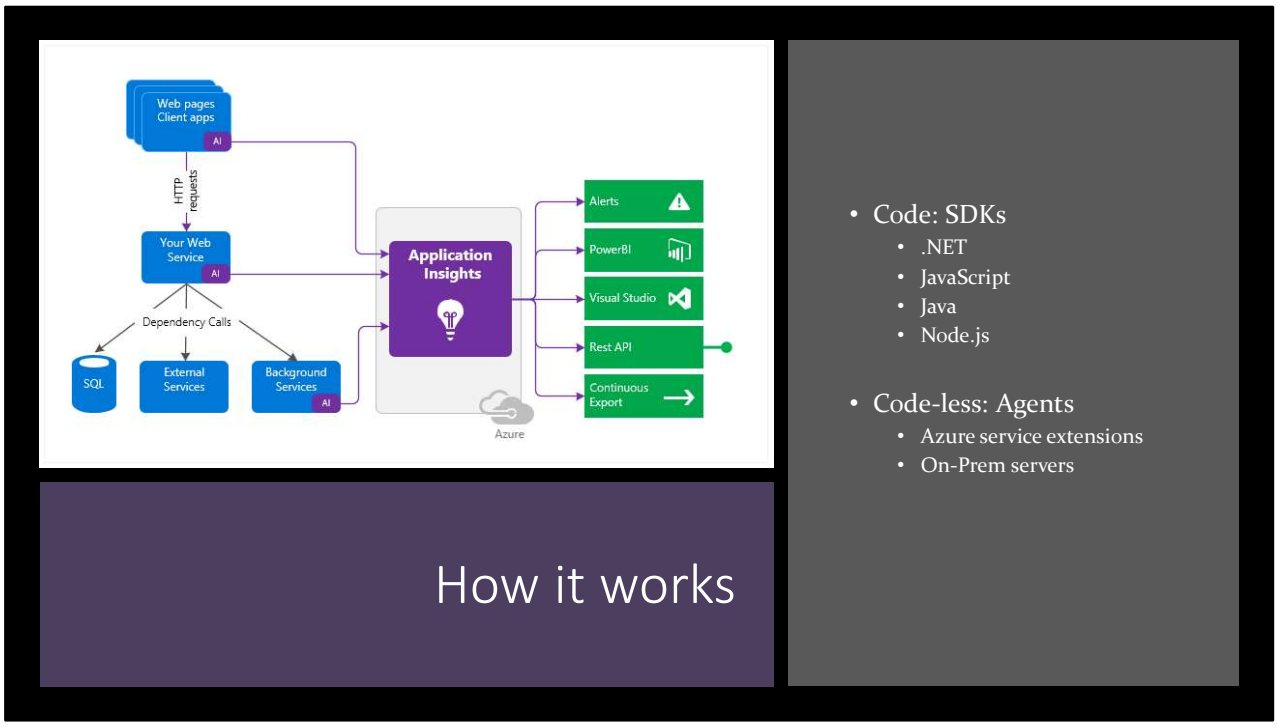


Usage: Cohorts

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

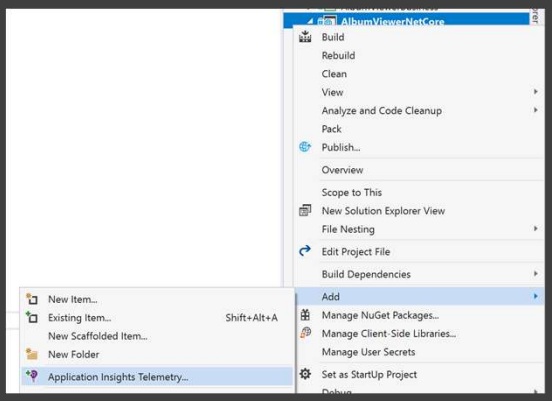
Extensions – VM, App Service

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()`…
   - Add `services.AddApplicationInsightsTelemetry();`

3. Add instrumentation key…
   - appsettings.json
   - Environment variable
     - APPINSIGHTS_INSTRUMENTATIONKEY
     - ApplicationInsights:InstrumentationKey

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[sd
{
  instrumentationKey:"INSTRUMENTATION_KEY"
}
);window[aiName]=aisdk,aisdk.queue&&0===aisdk.queue.length&&aisdk.trackPageView({});
</script>
```

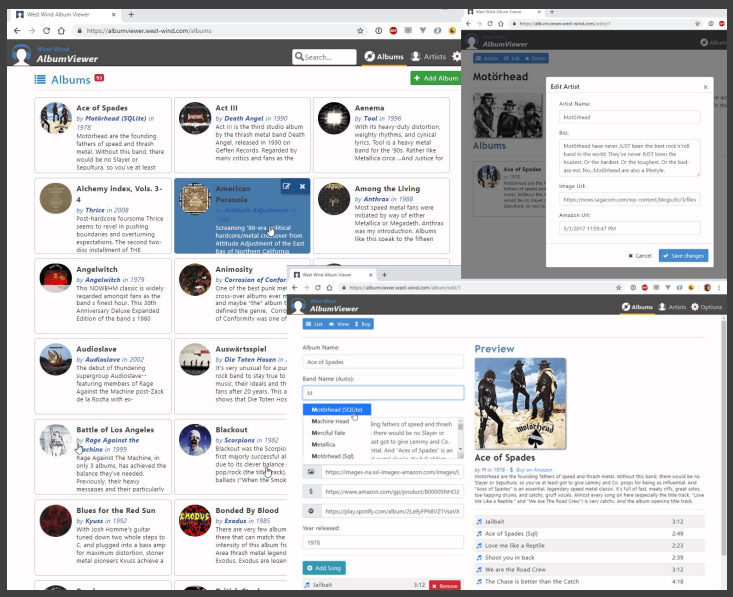
https://docs.microsoft.com/en-us/azure/azure-monitor/app/javascript

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

Test drive: AlbumViewerVNext

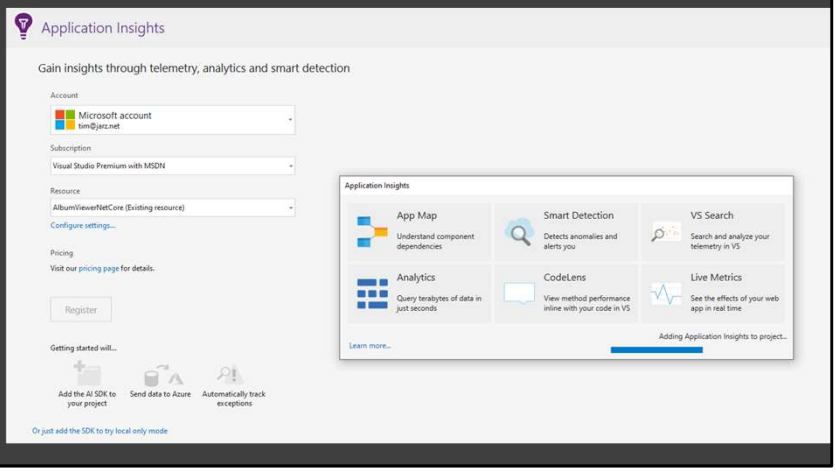
https://github.com/RickStrahl/AlbumViewerVNext

- ASP.NET Core
- Angular



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

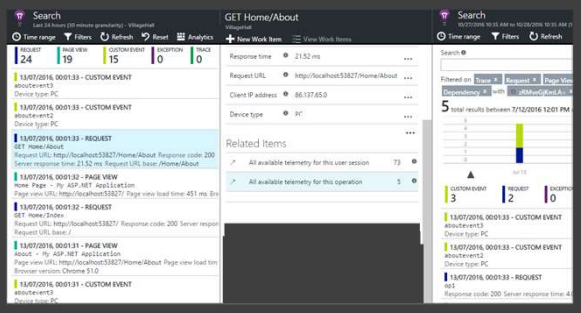
- *Can* manually track HTTP requests
- Recommendation: use operation contexts instead
  - Handles correlation

```
// 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