Microsoft

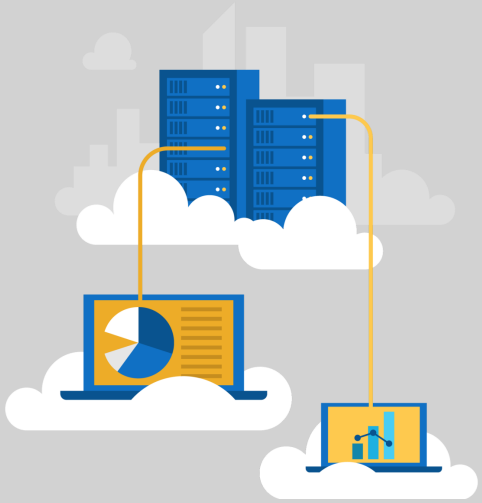
Module 12: Instrument solutions to support monitoring and logging

Microsoft

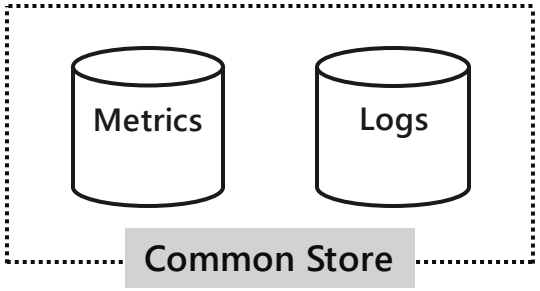
Topics

- Overview of monitoring in Azure
- Configure instrumentation in an app or service
- Analyzing and troubleshooting apps
- Implement code that handles transient faults

Lesson 01: Overview of monitoring in Azure

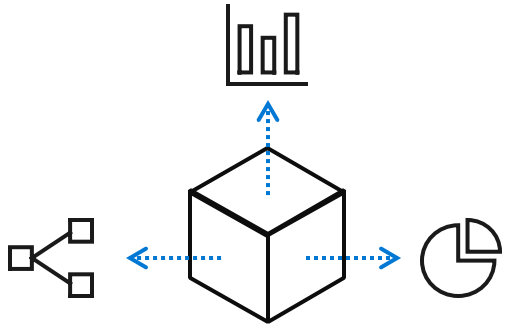


Azure Monitor



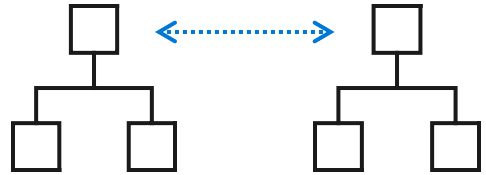
Unified Monitoring

A common platform for all metrics, logs and other monitoring telemetry



Data Driven Insights

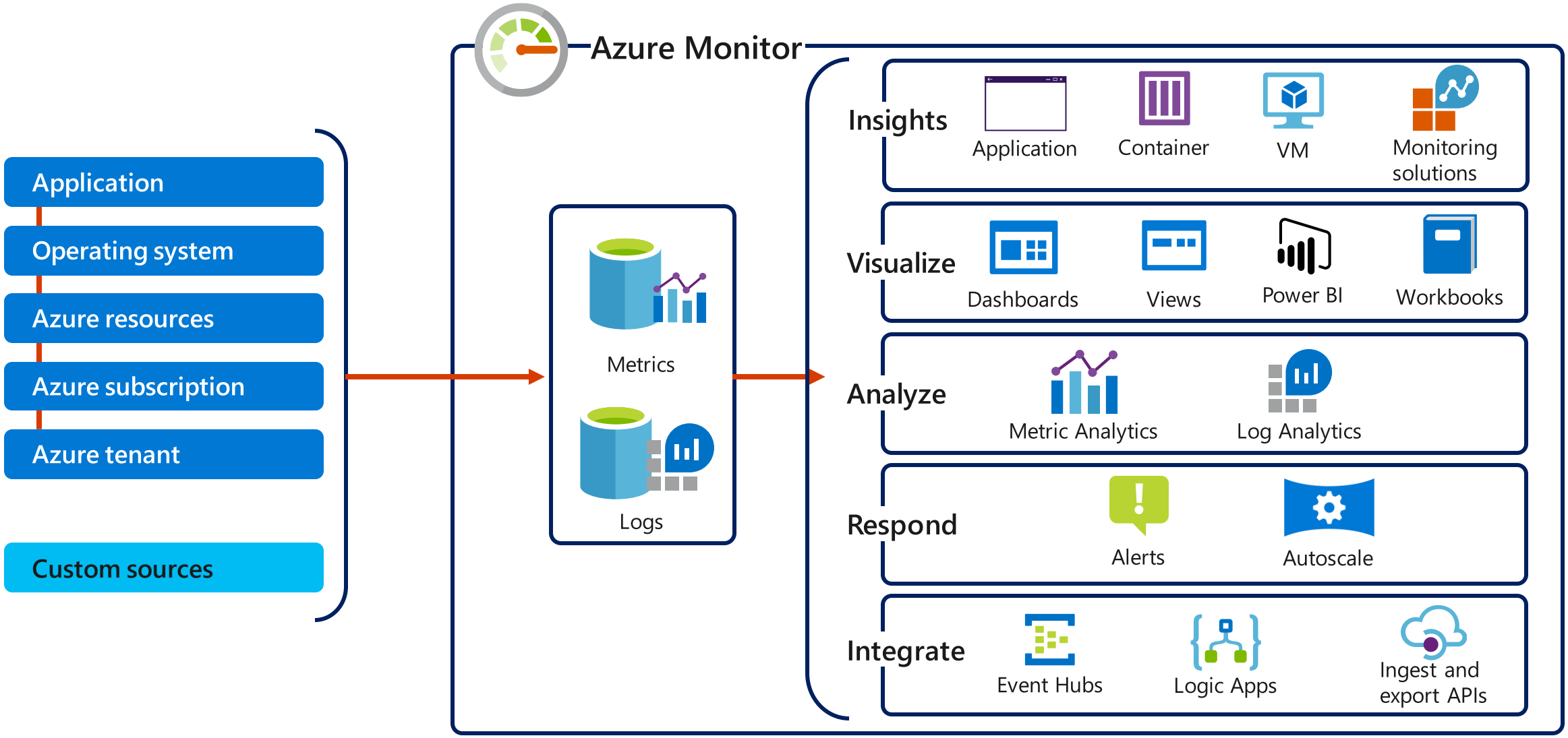
Advanced diagnostics and analytics powered by machine learning capabilities



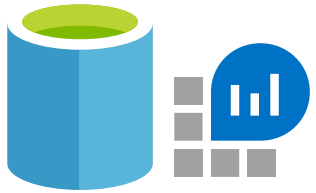
Workflow Integrations

Rich ecosystem of popular DevOps, issue management, SIEM, and ITSM tools

Azure Monitor overview

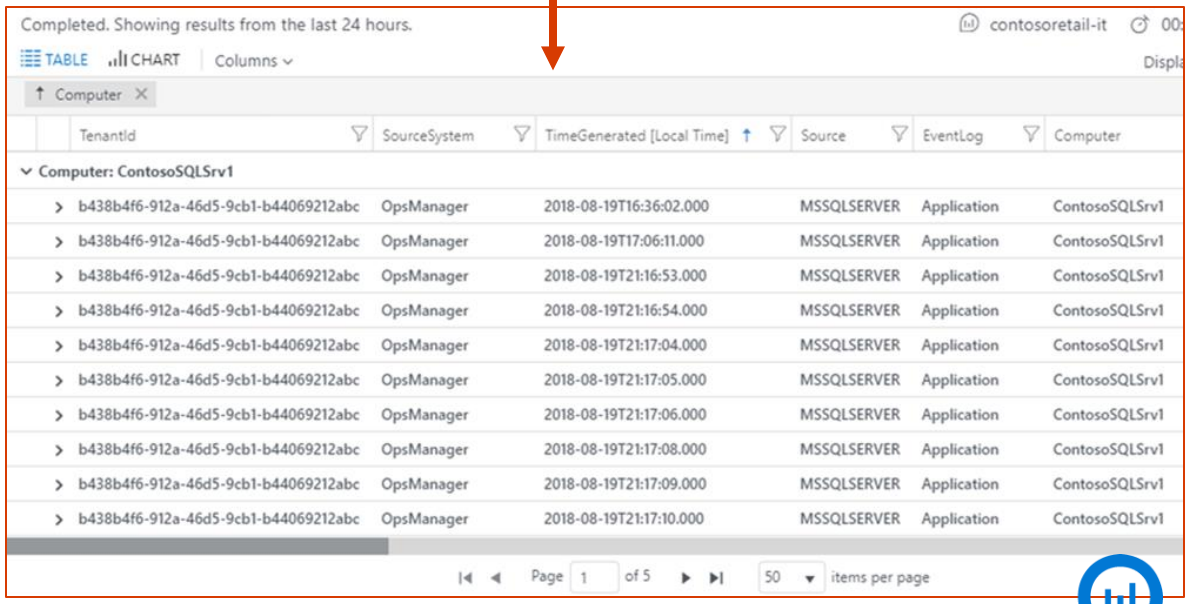


Monitoring data platform

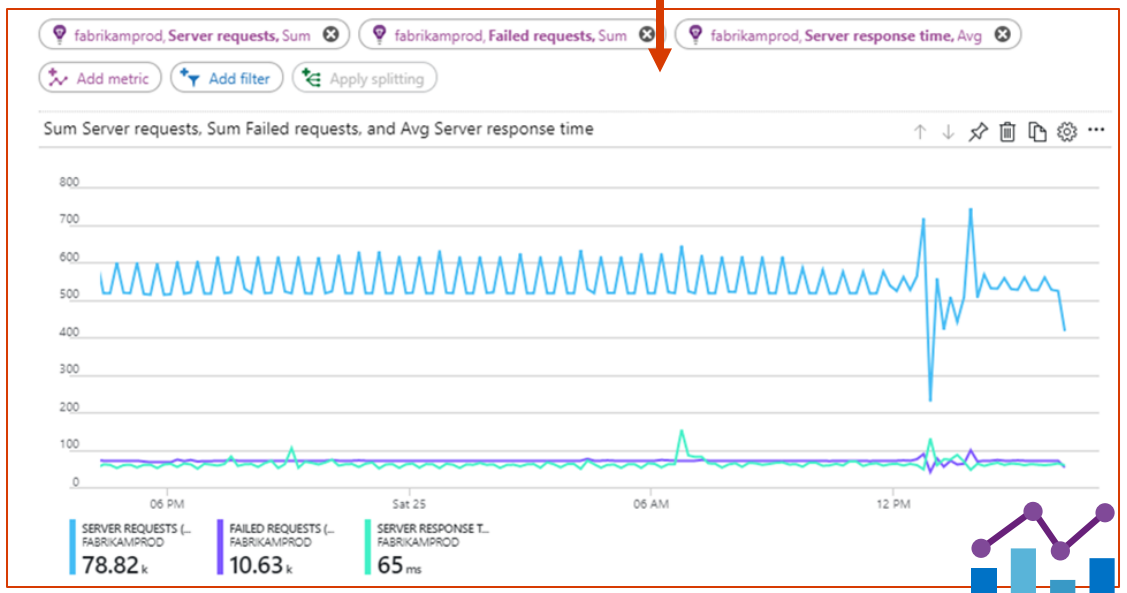


Logs

Metrics



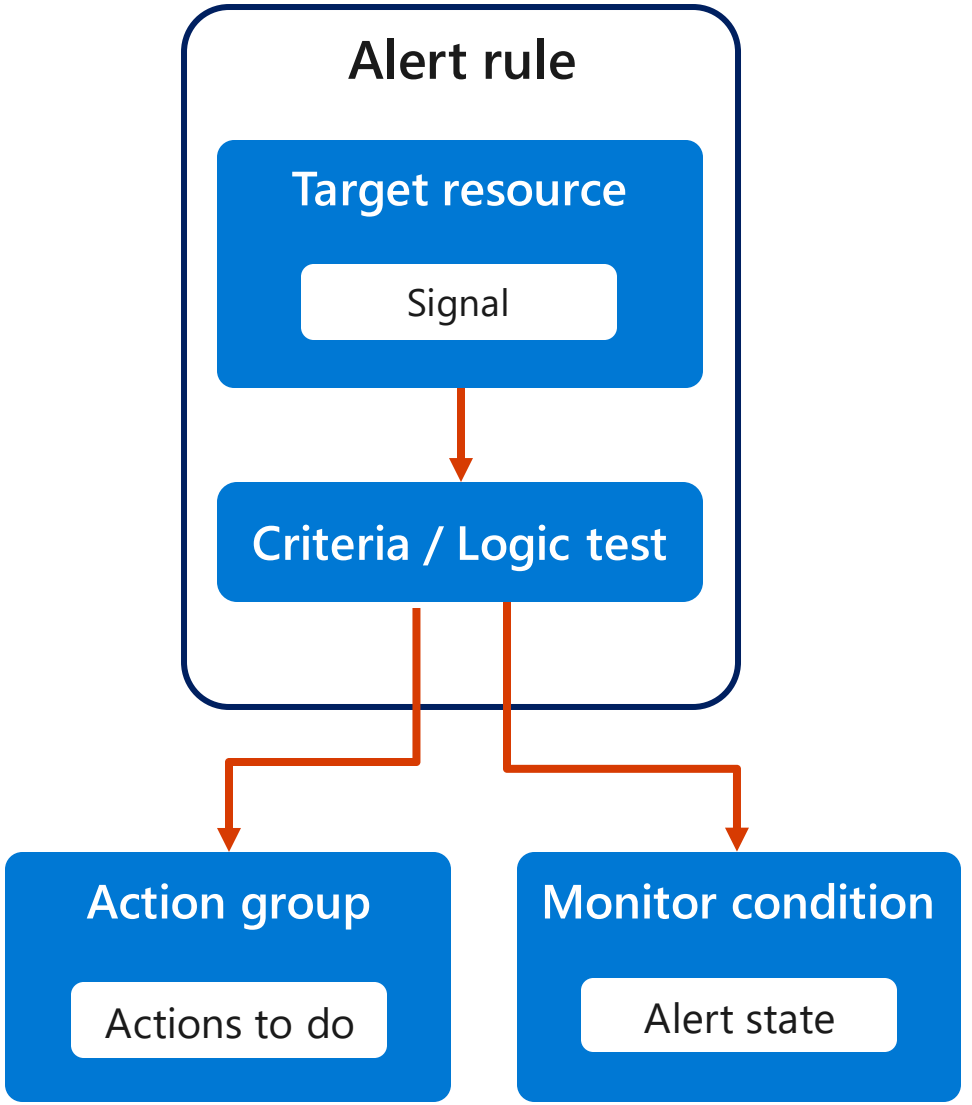
Log Analytics



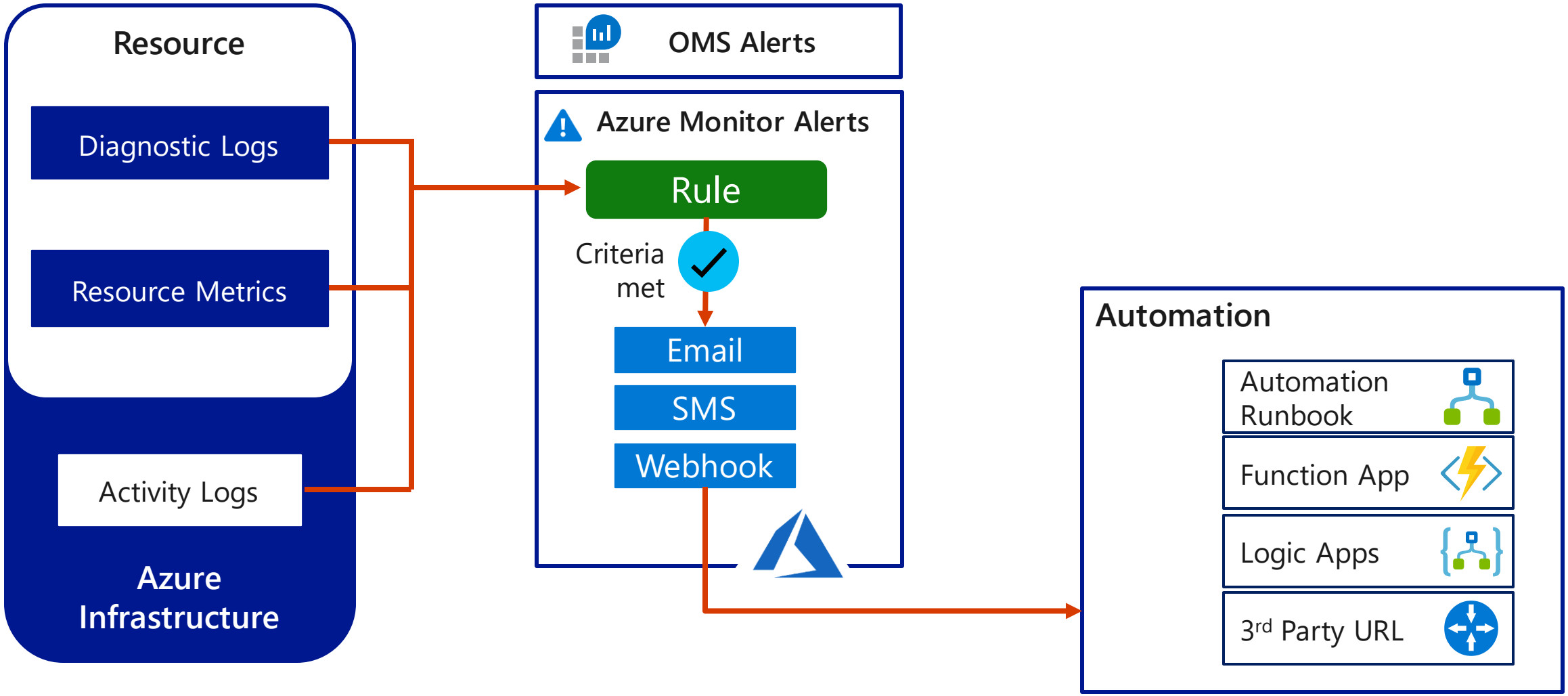
Metrics Explorer

Alerts

- Proactively notify you when conditions are met
  - Defined in alert rules
- Now unified across multiple services
  - Application Insights
  - Log Analytics
  - Azure Monitor



Alerts workflow



Alert state

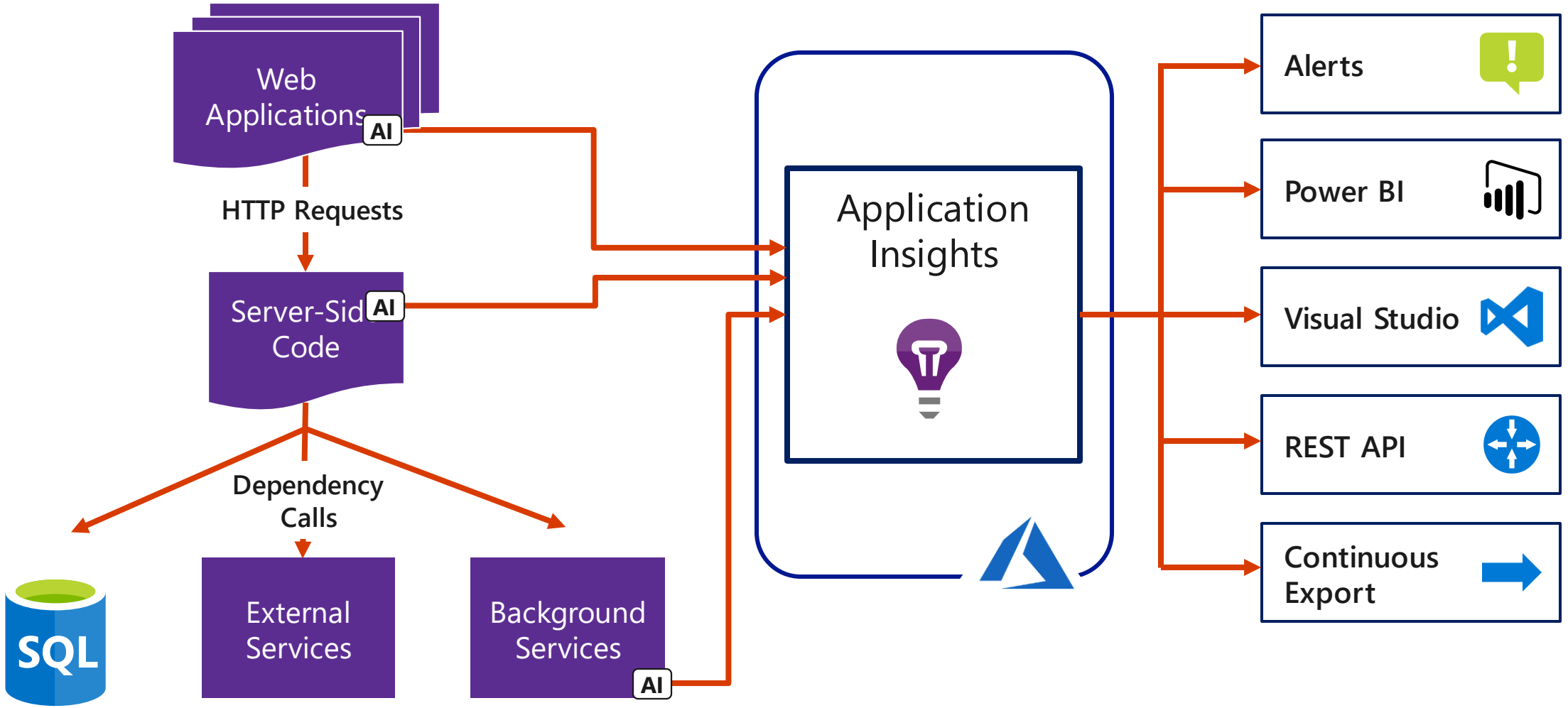
State	Description
New	The issue has just been detected and has not yet been reviewed.
Acknowledged	An administrator has reviewed the alert and started working on it.
Closed	The issue has been resolved. After an alert has been closed, you can reopen it by changing it to another state.

Application Insights

Application performance management service:

- Collects application telemetry
- Processes telemetry by using Machine Learning technology
- Detects and diagnoses problems automatically
- Enables you to use queries to answer complex questions

Application Insights architecture



Application Insights for console applications

- Install latest **Microsoft.ApplicationInsights** NuGet package
- Install latest **Microsoft.ApplicationInsights.DependencyCollector** NuGet package
- Set the instrumentation key
  - By using the static **TelemetryConfiguration.Active.InstrumentationKey** property
  - By using the **APPINSIGHTS_INSTRUMENTATIONKEY** environment variables
  - By using the **ApplicationInsights.config** file

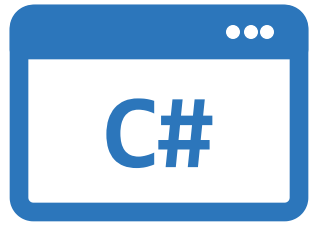
Application Insights for console applications - files

```
using System.IO;

// Reads ApplicationInsights.config file if present
TelemetryConfiguration config = TelemetryConfiguration.Active;

TelemetryConfiguration configuration =
TelemetryConfiguration.CreateFromConfiguration(File.ReadAllText("C:\\ApplicationInsights.config"));

var telemetryClient = new TelemetryClient(configuration);
```

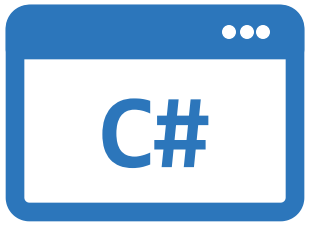


Application Insights for desktop apps

- Can be configured in a manner very similar to Application Insights for console apps
- Install latest **Microsoft.ApplicationInsights** NuGet package
- Install latest **Microsoft.ApplicationInsights.DependencyCollector** NuGet package
- Set the instrumentation key
  - By using the static **TelemetryConfiguration.Active.InstrumentationKey** property
  - By using the **ApplicationInsights.config** file

Application Insights for desktop apps - code

```
public partial class Form1 : Form
{
    private TelemetryClient tc = new TelemetryClient();
    private void Form1_Load(object sender, EventArgs e)
    {
        // Alternative to setting ikey in config file:
        tc.InstrumentationKey = "key copied from portal";
        // Set session data:
        tc.Context.User.Id = Environment.UserName;
        tc.Context.Session.Id = Guid.NewGuid().ToString();
        tc.Context.Device.OperatingSystem = Environment.OSVersion.ToString();
        // Log a page view:
        tc.TrackPageView("Form1");
        ...
    }
}
```



Application Insights platforms

- Official support – languages
  - .NET (C# & Microsoft Visual Basic)
  - Java
  - JavaScript
  - Node.js
- Unofficial support – languages
  - F#
  - PHP
  - Python
  - Ruby
- Official support –platforms/frameworks
  - ASP.NET (including ASP.NET)
  - Android
  - Angular
  - Azure (Azure App Service, Azure Cloud Services, Azure Functions)
  - Docker
  - Glimpse
  - iOS
  - Java 2 Platform Enterprise Edition (J2EE)
  - OS X
  - Spring
  - Universal Windows Platform (UWP)
  - Windows Communication Foundation (WCF)

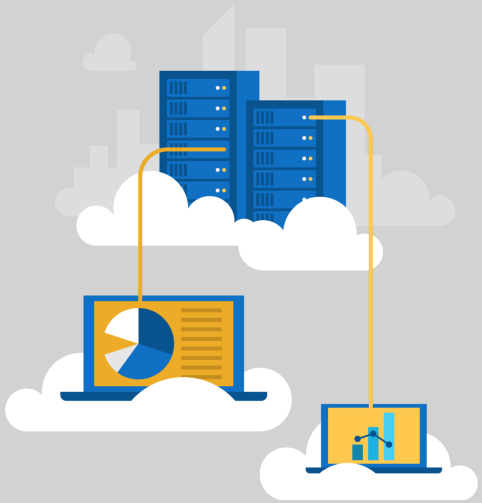
Other monitoring tools

- Cloudyn:
  - Manages and optimizes multi-platform, hybrid cloud deployments to help enterprises fully realize their cloud potential. The software as a service (SaaS) solution delivers insight into usage, performance, and cost. It also provides insights and actionable recommendations for smart optimization and cloud governance.
- AppDynamics:
  - Application Performance Management (APM), which enables application owners to rapidly troubleshoot performance bottlenecks and optimize the performance of their applications running in an Azure environment.
- Datadog:
  - A monitoring service that gathers monitoring data from your containers within your Azure Container Service cluster. Datadog has a Docker Integration Dashboard, where you can view specific metrics within your containers. Metrics gathered from your containers are organized by CPU, memory, network, and I/O.

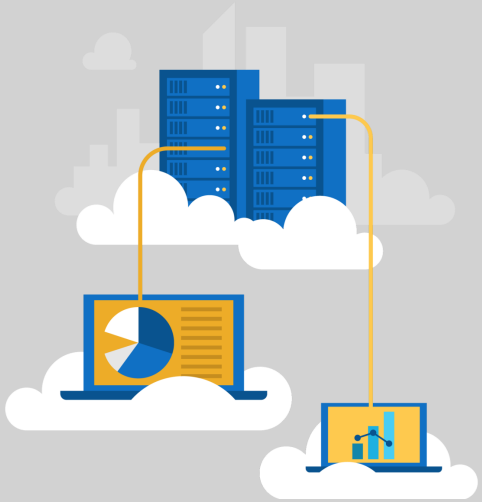
Other monitoring tools (continued)

- The Elasticsearch, Logstash, and Kibana (ELK) stack
  - Combination of Elasticsearch, Logstash, and Kibana that provides an end-to-end stack that can be used to monitor and analyze logs in your cluster. The ELK stack is popular for monitoring container clusters, because the monitoring stack itself is open source and already containerized.
- New Relic Application Performance Management
  - Popular APM add-in for .NET applications. New Relic Application Performance Management can be used similarly in the Azure platform and has first-class support in role-based access control scenarios.

Demonstration: Instrumenting an ASP.NET app for monitoring in Application Insights

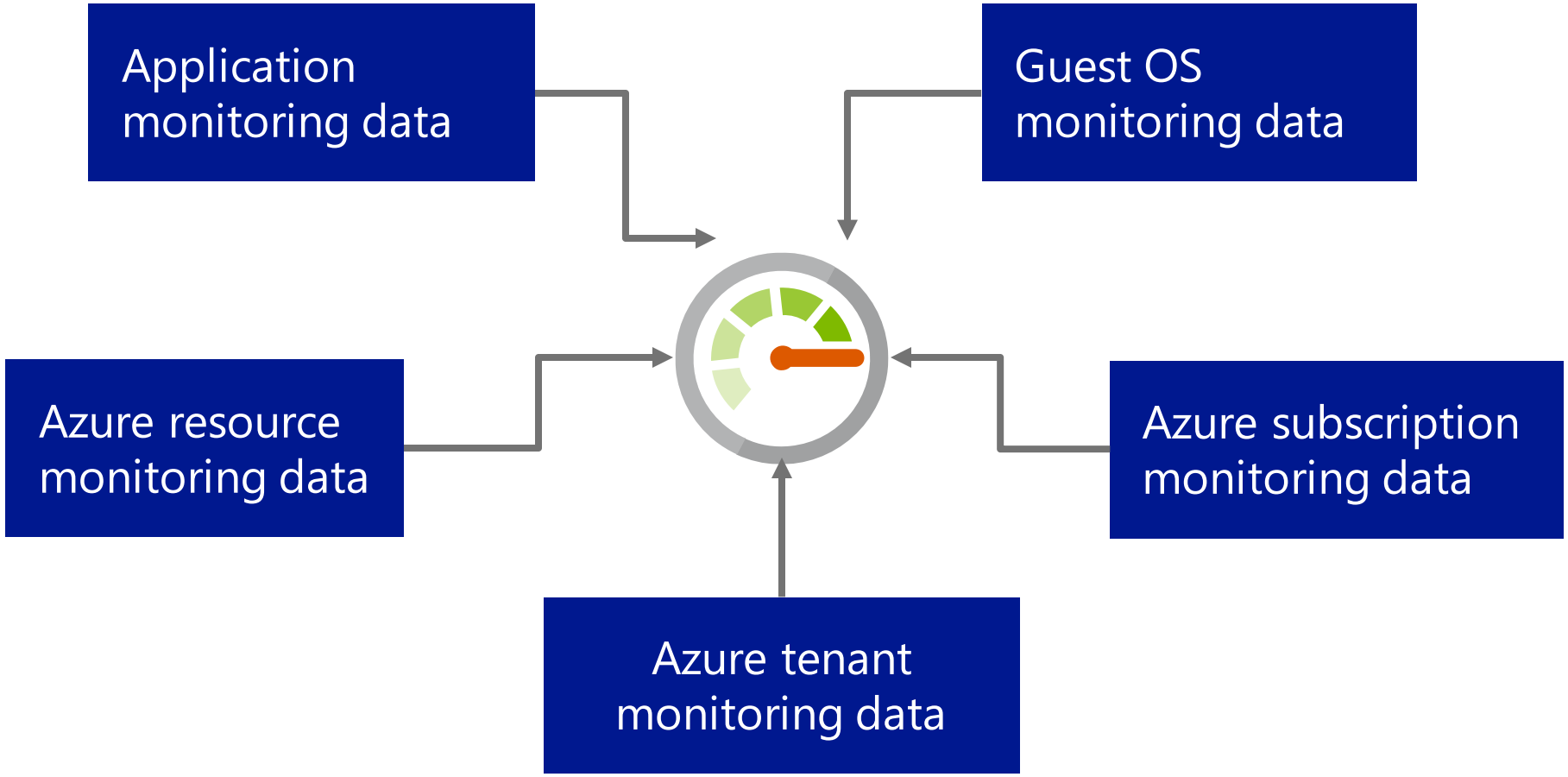


Lesson 03: Analyzing and troubleshooting apps

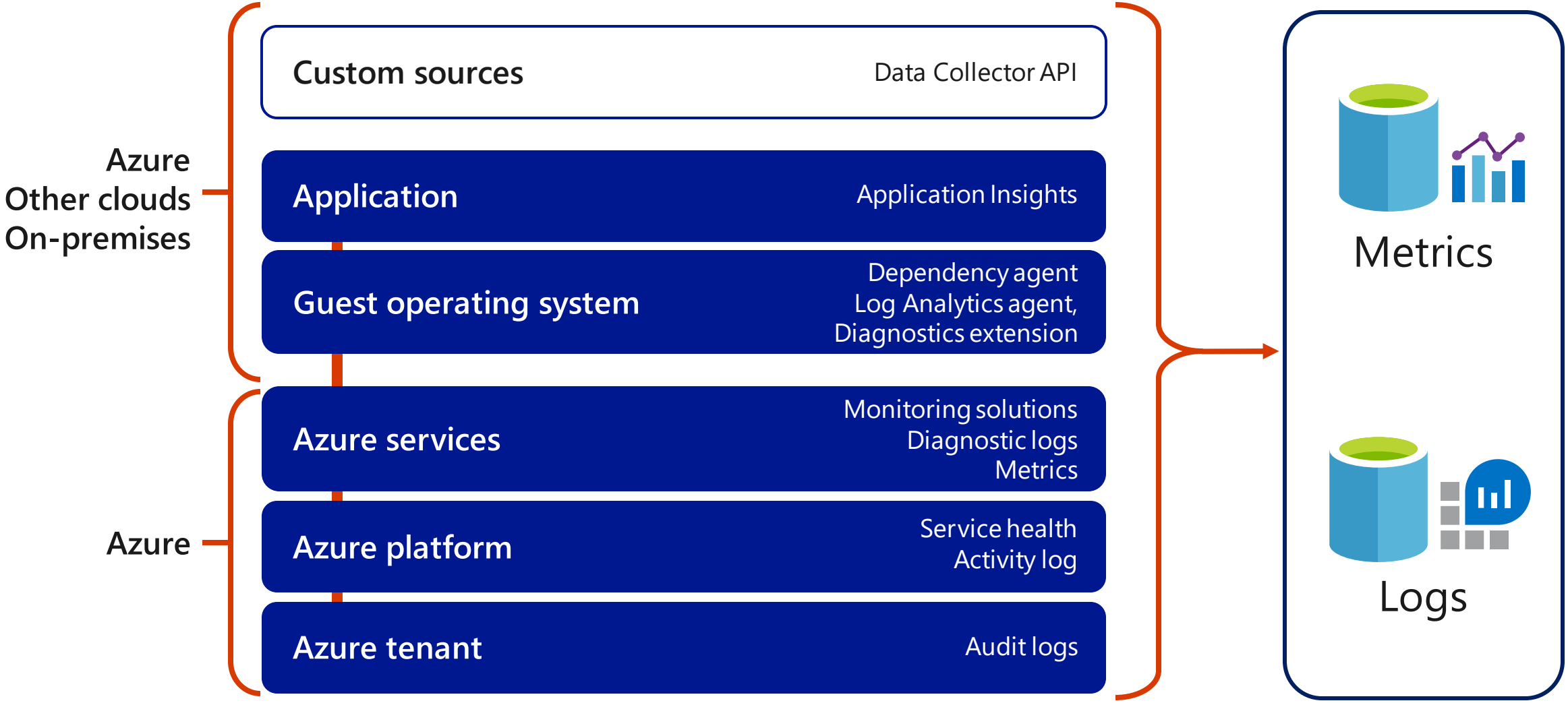


What data does Azure Monitor collect?

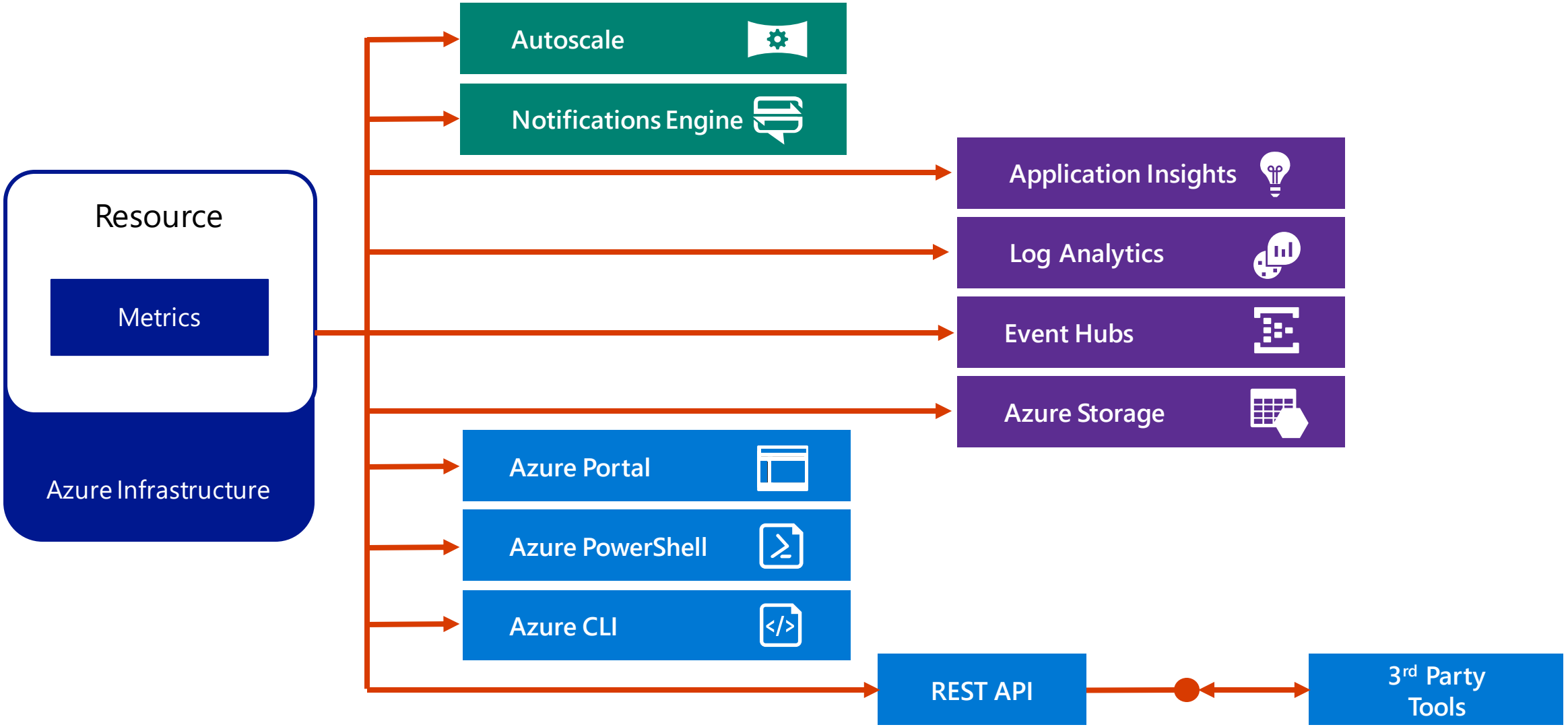
- Azure Monitor can collect data from:



Data sources

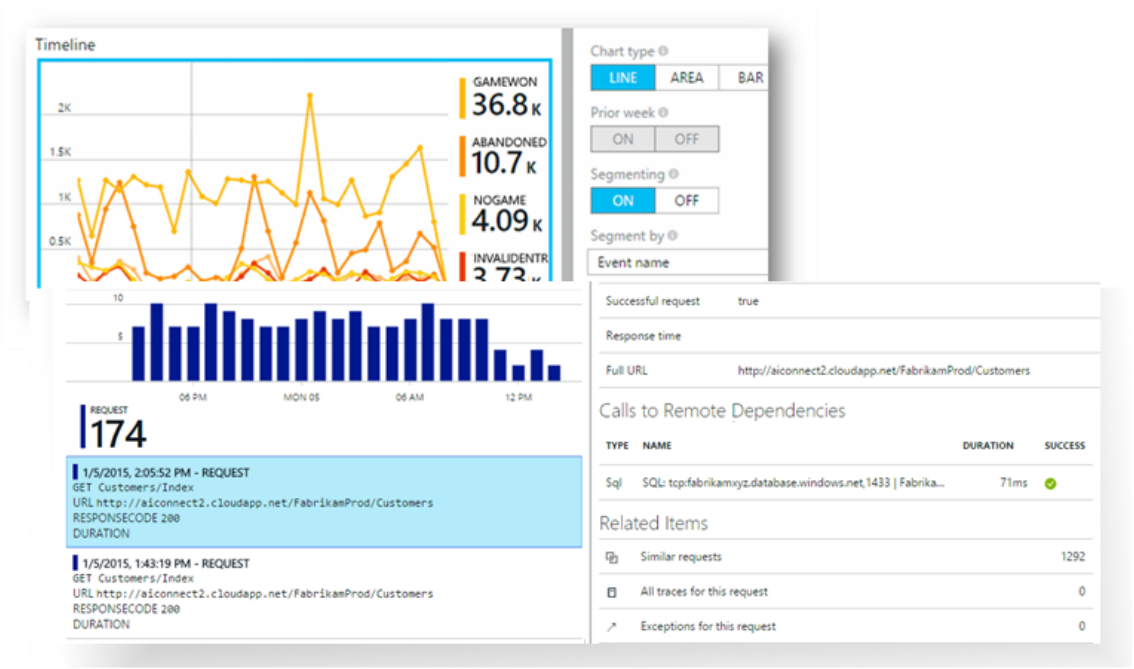


Azure Monitor sources



Application Insights overview

- Extensible application performance monitoring service
- Can be used to:
  - Monitor a live web application
  - Automatically detect performance anomalies
  - Diagnose issues by using analytical tools
  - Understand real-world user behavior by using custom queries and metric visualizations



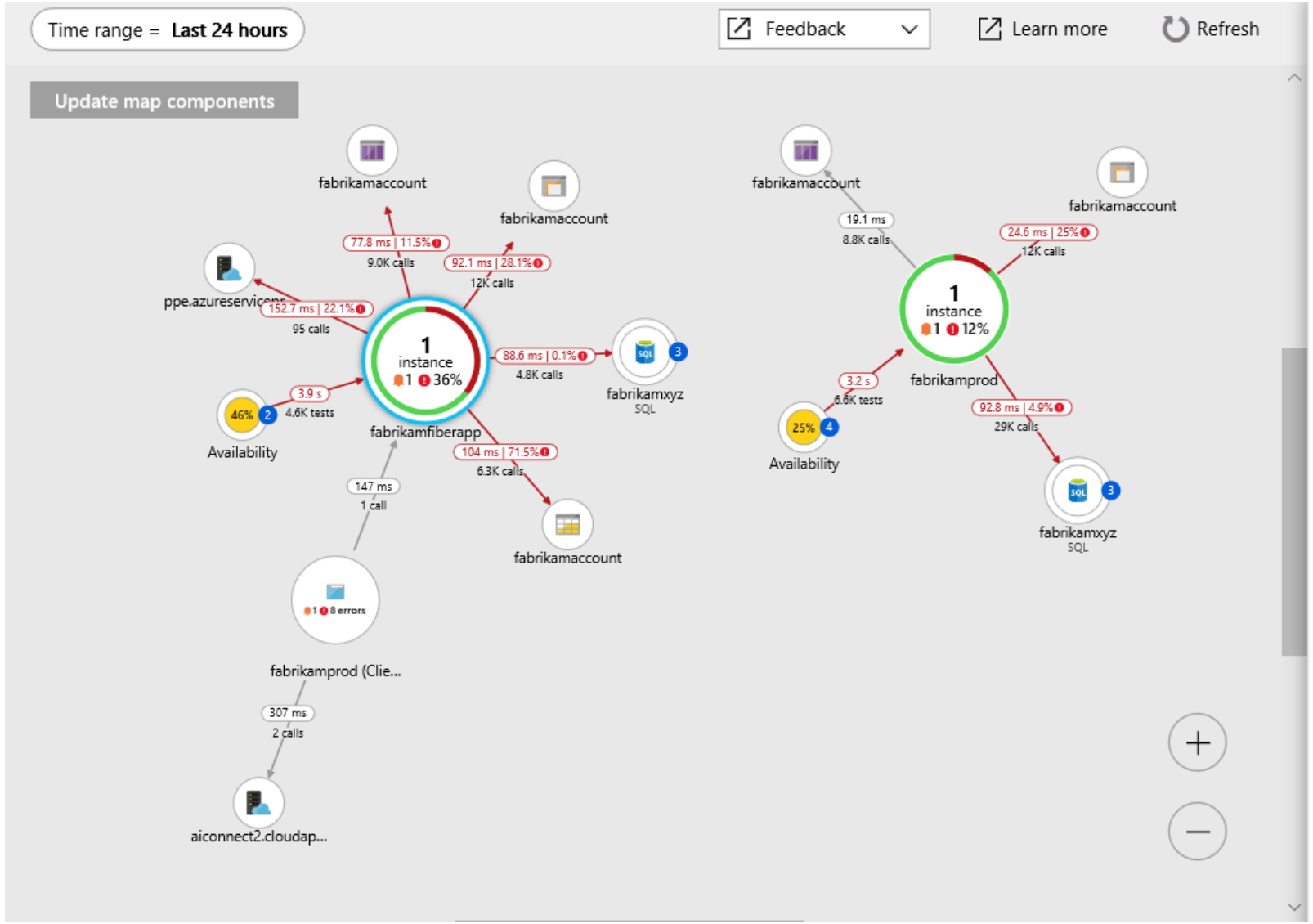
Monitored metrics

- Request rates, response times, and failure rates
  - Find out which pages are most popular, what times of day are most popular, and where your users are. Observe which pages perform the best. If your response times and failure rates go high when there are more requests, you might have a resourcing problem.
- Dependency rates, response times, and failure rates
  - Find out whether external services are slowing you down
- Exceptions
  - Analyze the aggregated statistics, or pick specific instances and drill into the stack trace and related requests. Both server and browser exceptions are reported.
- Page views and load performance
  - Directly reported by your users' browsers
- User and session counts

Monitored metrics (continued)

- Asynchronous JavaScript and XML (AJAX) calls
  - Rates, response times, and failure rates for these webpage-based calls
- Performance counters
  - Measured from your Windows Server or Linux server machines, such as counters for CPU, memory, and network usage
- Host diagnostics
  - Ingested from Docker or Azure
- Diagnostic trace logs
  - Logs from your app so that you can correlate trace events with requests
- Custom events and metrics
  - Custom metrics that you write yourself in the client or server code to track business events, such as the number of items sold or games won

Application Map



fabrikamfiberapp

TOP FAILING REQUESTS BY NAME

	COUNT
GET Customers/Details	2.0K
GET Home/Index	362
GET /Content/fonts/segoewp-...	9

Investigate failures

SLOWEST REQUESTS BY NAME

	DURATION (AV...
GET ServiceTickets/Assign	4.2 s
GET Home/Index	4 s
GET Customers/Details	3.6 s

Investigate performance

Resource links

View in Analytics | Alerts (1)

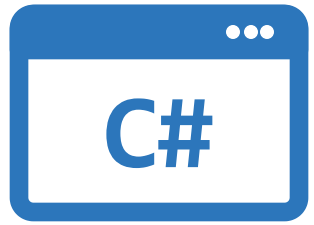
Components

- Components are independently deployable parts of your distributed/microservices application
  - Components are different from "observed" external dependencies
  - Components run on any number of server/role/container instances
- Components can be separate Application Insights instrumentation keys, or different roles reporting to a single Application Insights instrumentation key

Application Map - code

```
using Microsoft.ApplicationInsights.Channel;
using Microsoft.ApplicationInsights.Extensibility;

namespace CustomInitializer.Telemetry
{
    public class MyTelemetryInitializer : ITelemetryInitializer
    {
        public void Initialize(ITelemetry telemetry)
        {
            if (string.IsNullOrEmpty(telemetry.Context.Cloud.RoleName))
            {
                //set custom role name here
                telemetry.Context.Cloud.RoleName = "RoleName";
            }
        }
    }
}
```

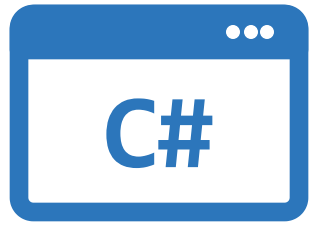


Application Map - configuration

```
<ApplicationInsights>
    <TelemetryInitializers>
        <!-- Fully qualified type name, assembly name: -->
        <Add Type="CustomInitializer.Telemetry.MyTelemetryInitializer, CustomInitializer"/>
        ...
    </TelemetryInitializers>
</ApplicationInsights>
```

```
using Microsoft.ApplicationInsights.Extensibility;
using CustomInitializer.Telemetry;

protected void Application_Start()
{
    ...
    TelemetryConfiguration.Active.TelemetryInitializers.Add(
        new MyTelemetryInitializer()
    );
}
```



View activity logs to audit actions on resources

- Through activity logs, you can determine:
  - What operations were taken on the resources in your subscription
  - Who initiated the operation (although operations initiated by a back-end service do not return a user as the caller)
  - When the operation occurred
  - The status of the operation
  - The values of other properties that might help you research the operation

Auditing in Azure PowerShell

```
Get-AzLog -ResourceGroup ExampleGroup

Get-AzLog -ResourceGroup ExampleGroup -StartTime 2015-08-28T06:00 -EndTime 2015-09-10T06:00

Get-AzLog -ResourceGroup ExampleGroup -StartTime (Get-Date).AddDays(-14)

Get-AzLog -ResourceGroup ExampleGroup -StartTime (Get-Date).AddDays(-14) | Where-Object OperationName -eq Microsoft.Web/sites/stop/action

Get-AzLog -ResourceGroup deletedgroup -StartTime (Get-Date).AddDays(-14) -Caller someone@contoso.com

Get-AzLog -ResourceGroup ExampleGroup -Status Failed
```

Auditing in Azure PowerShell – retrieve specific operation

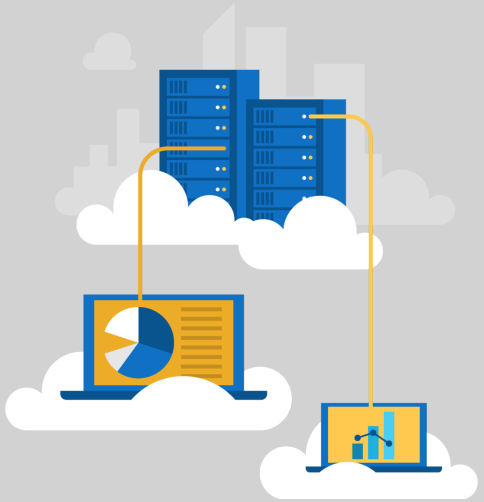
```
((Get-AzLog -Status Failed -ResourceGroup ExampleGroup -
DetailedOutput).Properties[1].Content["statusMessage"] | ConvertFrom-Json).error

code message
---- -------
DnsRecordInUse DNS record dns.westus.cloudapp.azure.com is already used by another
public IP.
```

Monitor availability and responsiveness of a website

- Use test to monitor availability and responsiveness
  - Tests send web requests to the application at regular intervals
  - Tests send requests from around the world
- Tests can point to any HTTP or HTTPs endpoint
  - Even if it's not hosted on Azure
- Two types of availability tests
  - URL ping test
    - Simple test that you can create in the Azure portal
  - Multi-step web test
    - Created in Visual Studio Enterprise and uploaded to the portal

Lesson 04: Implement code that handles transient faults



Transient errors

- Transient faults are temporary faults
  - Could be caused by environmental issues
    - Loss of network connectivity
    - Busy hardware components
    - Temporary unavailability of a connected service
    - Server timeouts
  - Typically are self-correcting

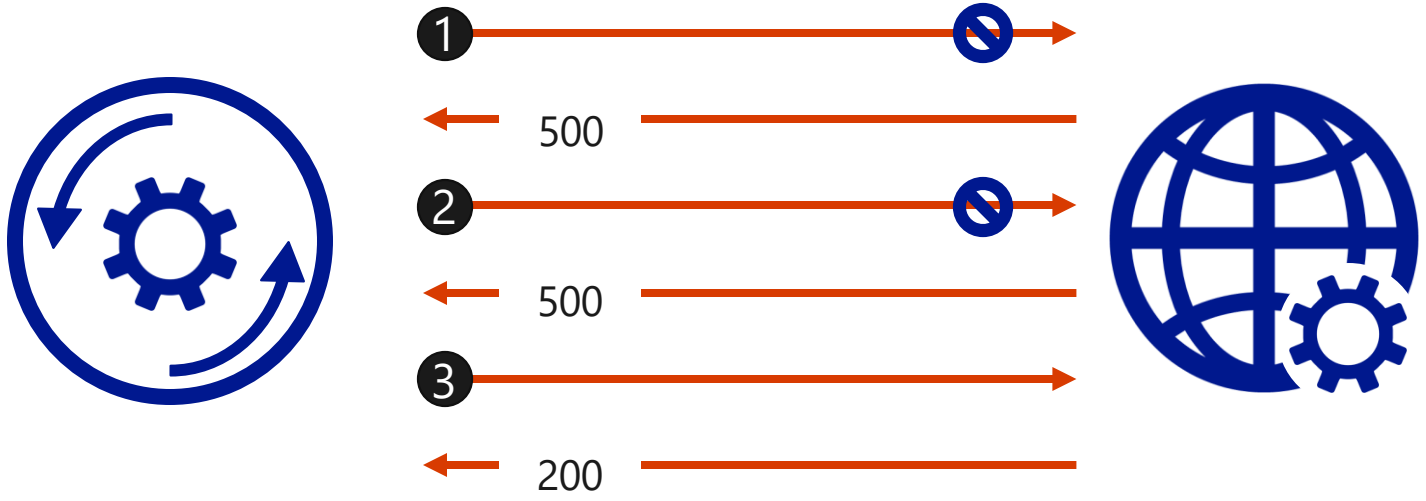
Handling transient errors

If an application detects a failure when it tries to send a request to a remote service, it can handle the failure by using the following strategies:

- Cancel
  - If the fault indicates that the failure isn't transient or is unlikely to be successful if repeated, the application should cancel the operation and report an exception. For example, an authentication failure caused by providing invalid credentials is not likely to succeed no matter how many times it's attempted.
- Retry
  - If the specific fault reported is unusual or rare, it might have been caused by unusual circumstances, such as a network packet becoming corrupted while it was being transmitted. In this case, the application could retry the failing request again immediately, because the same failure is unlikely to be repeated and the request will probably be successful.
- Retry after a delay
  - If the fault is caused by one of the more commonplace connectivity or busy failures, the network or service might need a short period of time while the connectivity issues are corrected or the backlog of work is cleared. The application should wait for a suitable amount of time before retrying the request.

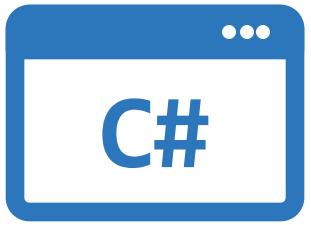
Retrying after a transient error

1. The application invokes an operation on a hosted service. The request fails, and the service host responds with HTTP response code 500 (internal server error).
2. The application waits for a short interval and tries again. The request still fails with HTTP response code 500.
3. The application waits for a longer interval and tries again. The request succeeds with HTTP response code 200 (OK).



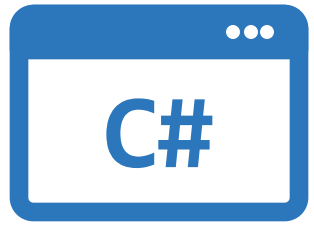
Handling transient errors in code

```
int retryCount = 3; readonly TimeSpan delay = TimeSpan.FromSeconds(5);
public async Task OperationWithBasicRetryAsync()
{
    int currentRetry = 0;
    for (;;)
    {
        try
        { await TransientOperationAsync(); break; }
        catch (Exception ex)
        {
            Trace.TraceError("Operation Exception");
            if (currentRetry++; > this.retryCount || !IsTransient(ex))
            { throw; }
        }
        await Task.Delay(delay);
    }
}
```



Detecting if an error is transient

```
private bool IsTransient(Exception ex)
{
    if (ex is OperationTransientException) return true;
    var webException = ex as WebException;
    if (webException != null)
    {
        return new[] {
            WebExceptionStatus.ConnectionClosed,
            WebExceptionStatus.Timeout,
            WebExceptionStatus.RequestCanceled
        }.Contains(webException.Status);
    }
    return false;
}
```



Lab: Monitoring services that are deployed to Azure

Duration

Lab sign-in information