**AZ-400: Implement continuous feedback**

**Advanced**

**Administrator**

**Developer**

**DevOps Engineer**

**Security Engineer**

**Security Operations Analyst**

**Service Adoption Specialist**

**Solution Architect**

**Technology Manager**

**Azure**

**Azure Artifacts**

**Azure Boards**

**Azure Cloud Services**

**Azure DevOps**

**Azure Pipelines**

**Azure Repos**

**Azure Test Plans**

**GitHub**

**This learning path introduces the continuous feedback concept and describes how to implement it in your DevOps cycle.**

**This learning path helps prepare you for**[**Exam AZ-400: Designing and Implementing Microsoft DevOps Solutions**](https://learn.microsoft.com/credentials/certifications/exams/az-400/)**.**

**Prerequisites**

Introduction

Completed100 XP

* 1 minute

This module introduces you to continuous feedback practices and tools to track usage and flow, such as Azure Logs Analytics, Kusto Query Language (KQL), and Application Insights.

Learning objectives

After completing this module, students and professionals can:

* Implement tools to track feedback.
* Plan for continuous monitoring.
* Implement Application Insights.
* Use Kusto Query Language (KQL).

Prerequisites

* Understanding of what DevOps is and its concepts.
* Familiarity with version control principles is helpful but isn't necessary.
* Beneficial to have experience in an organization that delivers software.

Next unit: Understand the inner loop

[Next](https://learn.microsoft.com/en-us/training/modules/implement-tools-track-usage-flow/2-understand-inner-loop/)

Understand the inner loop

Completed100 XP

* 1 minute

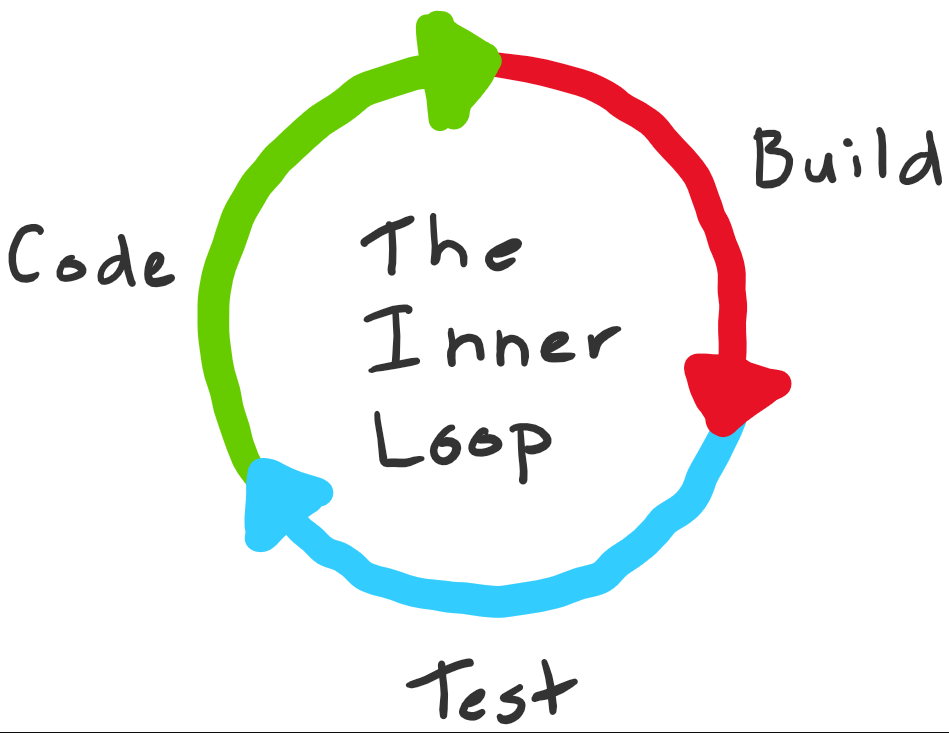
It isn't clear who coined the term "inner loop" in the context of software engineering, but within Microsoft, at least, the word seems to have stuck.

Many of the internal teams I work with see it as something they want to keep as short as possible - but what is the inner loop?

Definitions

The easiest way to define the inner loop is the iterative process that a developer does when writing, building, and debugging code.

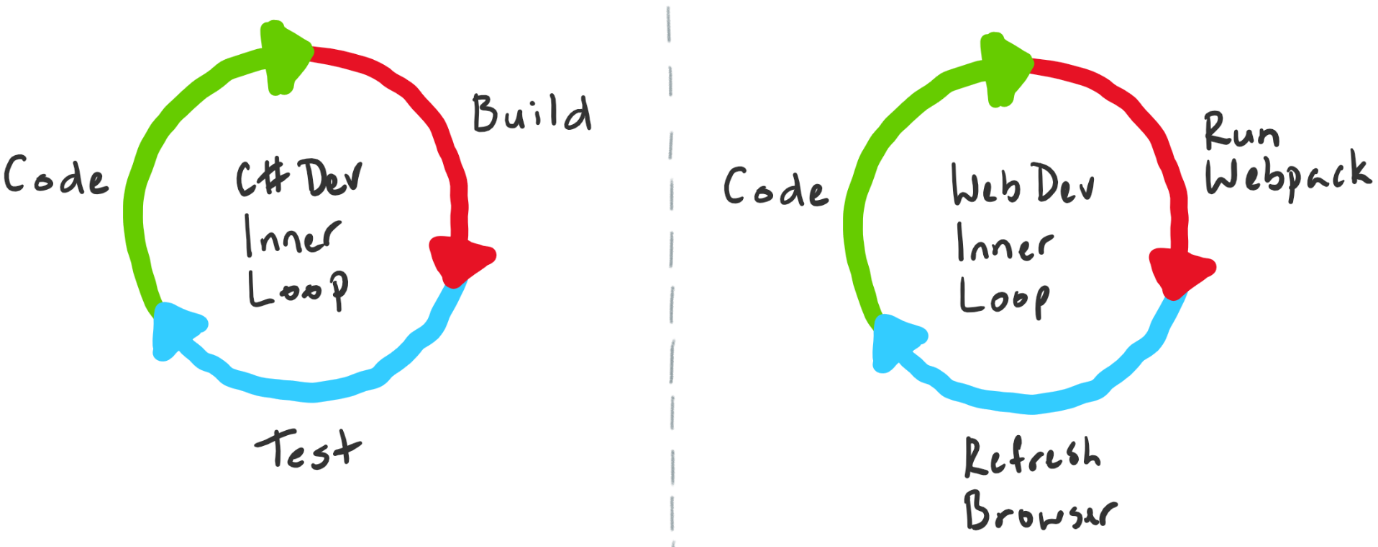
There are other things that a developer does. It's the right set of steps to be done repeatedly before sharing their work with their team or the rest of the world.



Exactly what goes into an individual developer's inner loop will depend significantly on the technologies they're working with, the tools used, and their preferences.

If I were working on a library, my inner loop would include coding, building, testing execution & debugging with regular commits to my local Git repository.

On the other hand, if I were doing web front-end work, I would probably be optimized around hacking on HTML & JavaScript, bundling, and refreshing the browser (followed by regular commits).



Most codebases comprise multiple-moving parts, so the definition of a developer's inner loop on any single codebase might alternate depending on what is being worked on.

Understanding the loop

The steps within the inner loop can be grouped into three broad activity buckets - experimentation, feedback collection, and tax.

If we flick back to the library development scenario I mentioned, I said four steps and how to bucket them.

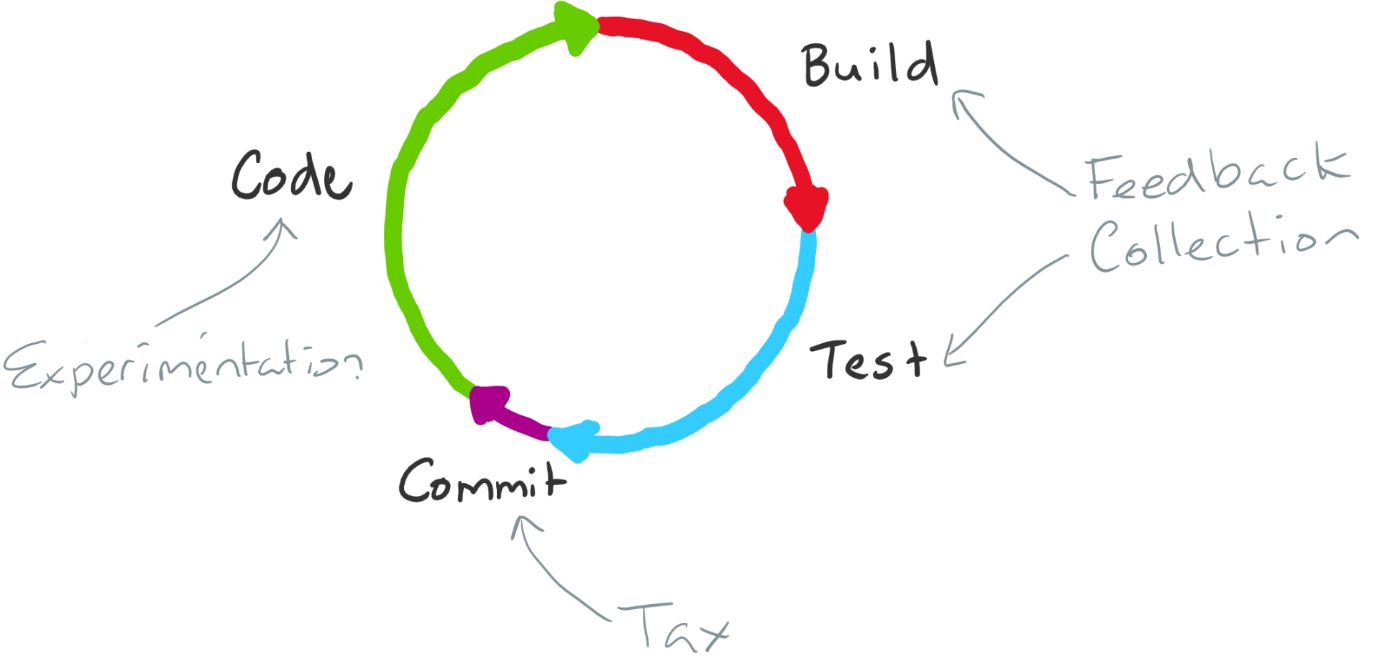
* Coding (Experimentation)
* Building (Feedback Collection)
* Testing / Debugging (Feedback Collection)
* Committing (Tax)

Of all the steps in the inner loop, coding is the only one that adds customer value.

Building and testing code is essential, but ultimately, we use them to give the developer feedback about their writing to see if it delivers sufficient value.

Putting committing code in the tax bucket is perhaps a bit harsh, but the purpose of the bucket is to call out those activities that neither add value nor provide feedback.

Tax is necessary to work. If it's unnecessary work, it's waste and should be eliminated.



Loop optimization

Having categorized the steps within the loop, it's now possible to make some general statements:

* You want to execute the loop as fast as possible and for the total loop execution time to be proportional to the changes made.
* You want to minimize the time feedback collection takes but maximize the quality of the feedback that you get.
* You want to minimize the tax you pay by eliminating it where it's unnecessary to run through the loop (can you defer some operations until you commit, for example).
* As new code and more complexity is added to any codebase, the amount of outward pressure to increase the size of the inner loop also increases. More code means more tests, which means more execution time and slow execution of the inner loop.

Suppose you have ever worked on a large monolithic codebase. In that case, it's possible to get into a situation where even small changes require a disproportionate amount of time to execute the feedback collection steps of the inner loop. It's a problem, and you should fix it.

There are several things that a team can do to optimize the inner loop for larger codebases:

* Only build and test what was changed.
* Cache intermediate build results to speed up to complete builds.
* Break up the codebase into small units and share binaries.

How you tackle each one of those is probably a blog post.

At Microsoft, for some of our genuinely massive monolithic codebases.

We're investing heavily in #1 and #2, but #3 requires a special mention because it can be a double-edged sword and can have the opposite of the wished impact if done incorrectly.

Tangled loops

To understand the problem, we need to look beyond the inner loop. Let us say that our monolithic codebase has an application-specific framework that does much heavy lifting.

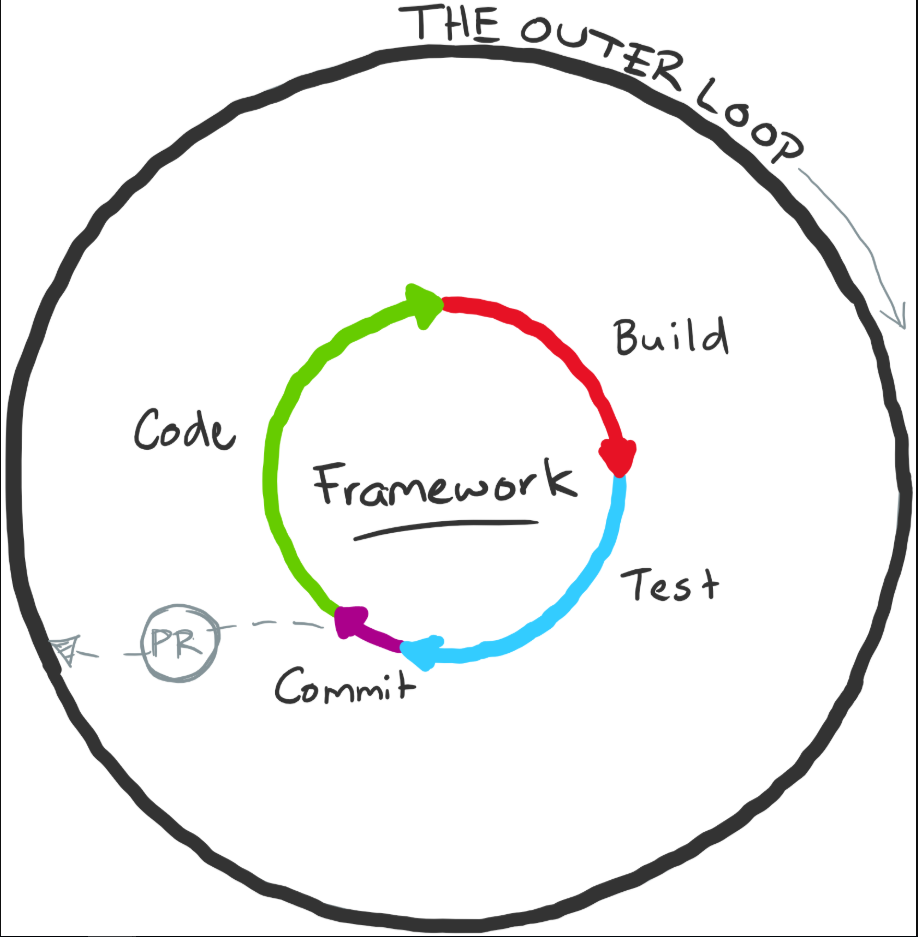
It would be tempting to extract that framework into a set of packages.

To do this, you would pull that code into a separate repository (optional, but this is generally how it's done), then set up a different CI/CD pipeline that builds and publishes the package.

A different pull-request process would also front this separate build and release pipeline to inspect changes before the code is published.

When someone needs to change this framework code, they clone down the repository, make changes (a separate inner loop), and submit a PR that transitions the workflow from the inner loop to the outer loop.

The framework package would then be available to be pulled into dependent applications (in this case, the monolith).



Initially, things might work out well. However, at some point in the future, you'll likely want to develop a new feature in the application that requires extensive new capabilities to be added to the framework.

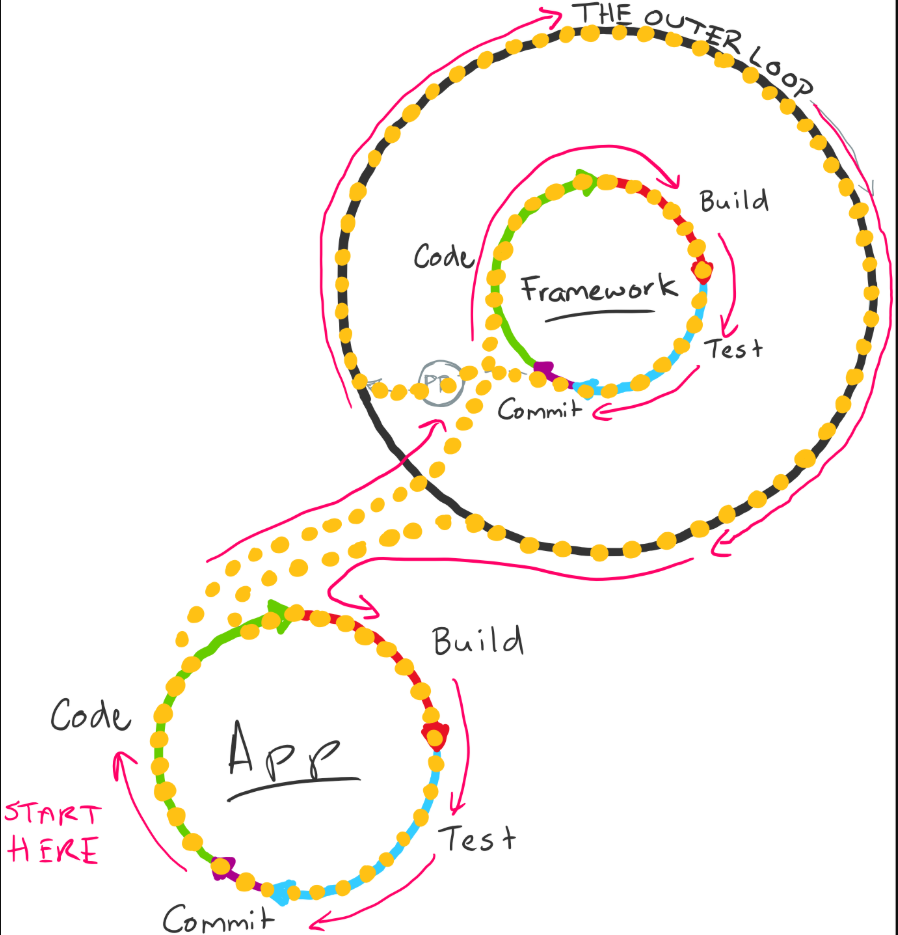
It's where teams that have broken up their codebases in suboptimal ways will start to feel pain.

If you have to coevolve code in two separate repositories where a binary/library dependency is present, you'll experience some friction.

In loop terms, the original codebase's inner loop now (temporarily at least) includes the outer loop of the previously broken-out framework code.

Outer loops include tax, including code reviews, scanning passes, binary signing, release pipelines, and approvals.

You don't want to pay that every time you've added a method to a class in the framework and now want to use it in your application.



What generally ends up happening next is a series of local hacks by the developer to stitch the inner loops together so they can move forward efficiently - but it gets messy quickly. You must pay that outer loop tax at some point.

It isn't to say that breaking code into separate packages is inherently bad - it can work brilliantly; you need to make those incisions carefully.

Closing thoughts

There's no silver bullet solution to ensure that your inner loop doesn't start slowing down, but it's essential to understand when it starts happening, what the cause is, and work to address it.

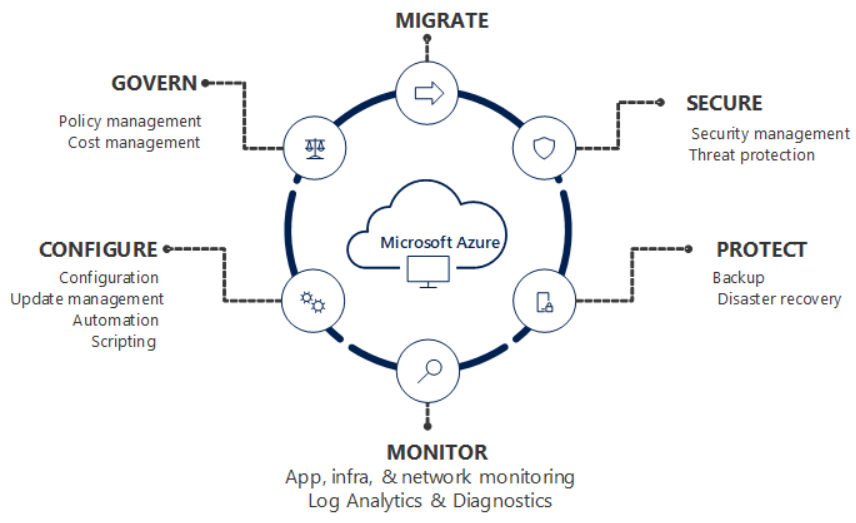
Decisions such as how you build, test, and debug the architecture will all impact developers' productivity. Improving one aspect will often cause issues in another.

Next unit: Introduction to continuous monitoring

Introduction to continuous monitoring

Completed100 XP

* 4 minutes



Continuous monitoring refers to the process and technology required to incorporate monitoring across each DevOps and IT operations lifecycles phase.

It helps to continuously ensure your application's health, performance, reliability, and infrastructure as it moves from development to production.

Continuous monitoring builds on the concepts of Continuous Integration and Continuous Deployment (CI/CD), which help you develop and deliver software faster and more reliably to provide continuous value to your users.

[Azure Monitor](https://learn.microsoft.com/en-us/azure/azure-monitor/overview) is the unified monitoring solution in Azure that provides full-stack observability across applications and infrastructure in the cloud and on-premises.

It works seamlessly with [Visual Studio and Visual Studio Code](https://visualstudio.microsoft.com/) during development and testing and integrates with [Azure DevOps](https://learn.microsoft.com/en-us/azure/devops/user-guide/index) for release management and work item management during deployment and operations.

It even integrates across your ITSM and SIEM tools to help track issues and incidents within your existing IT processes.

This article describes specific steps for using Azure Monitor to enable continuous monitoring throughout your workflows.

It includes links to other documentation that provides details on implementing different features.

Enable monitoring for all your applications

To gain observability across your entire environment, you need to enable monitoring on all your web applications and services.

It will allow you to visualize end-to-end transactions and connections across all the components easily.

* [Azure DevOps Projects gives you a simplified experience with your existing code and Git repository or choose](https://learn.microsoft.com/en-us/azure/devops-project/overview) from one of the sample applications to create a Continuous Integration (CI) and Continuous Delivery (CD) pipeline to Azure.
* [Continuous monitoring in your DevOps release pipeline](https://learn.microsoft.com/en-us/azure/application-insights/app-insights-vsts-continuous-monitoring) allows you to gate or roll back your deployment based on monitoring data.
* [Status Monitor](https://learn.microsoft.com/en-us/azure/application-insights/app-insights-monitor-performance-live-website-now) allows you to instrument a live .NET app on Windows with Azure Application Insights without modifying or redeploying your code.
* If you have access to the code for your application, then enable complete monitoring with [Application Insights](https://learn.microsoft.com/en-us/azure/application-insights/app-insights-overview) by installing the Azure Monitor Application Insights SDK for [.NET](https://learn.microsoft.com/en-us/azure/application-insights/quick-monitor-portal), [Java](https://learn.microsoft.com/en-us/azure/application-insights/app-insights-java-quick-start), [Node.js](https://learn.microsoft.com/en-us/azure/application-insights/app-insights-nodejs-quick-start), or [any other programming language](https://learn.microsoft.com/en-us/azure/application-insights/app-insights-platforms). It lets you specify custom events, metrics, or page views relevant to your application and business.

Enable monitoring for your entire infrastructure

Applications are only as reliable as their underlying infrastructure.

Monitoring enabled across your entire infrastructure will help you achieve full observability and make discovering a potential root cause easier when something fails.

Azure Monitor helps you track the health and performance of your entire hybrid infrastructure, including resources such as VMs, containers, storage, and network.

* You automatically get [platform metrics, activity logs, and diagnostics logs](https://learn.microsoft.com/en-us/azure/azure-monitor/data-sources) from most of your Azure resources with no configuration.
* Enable deeper monitoring for VMs with [Azure Monitor](https://learn.microsoft.com/en-us/azure/azure-monitor/insights/vminsights-overview).
* Enable deeper monitoring for AKS clusters with [Azure Monitor for containers](https://learn.microsoft.com/en-us/azure/azure-monitor/insights/container-insights-overview).
* Add [monitoring solutions](https://learn.microsoft.com/en-us/azure/azure-monitor/insights/solutions-inventory) for different applications and services in your environment.

[Infrastructure as code](https://learn.microsoft.com/en-us/azure/devops/learn/what-is-infrastructure-as-code) manages infrastructure in a descriptive model, using the same versioning as DevOps teams use for source code.

It adds reliability and scalability to your environment and allows you to apply similar processes to manage your applications.

* Use [Resource Manager templates](https://learn.microsoft.com/en-us/azure/azure-monitor/platform/template-workspace-configuration) to enable monitoring and configure alerts over a large set of resources.
* Use [Azure Policy](https://learn.microsoft.com/en-us/azure/governance/policy/overview) to enforce different rules over your resources. It ensures those resources comply with your corporate standards and service level agreements.

Combine resources in Azure Resource Groups

Today, a typical application on Azure includes multiple resources such as VMs and App Services or microservices hosted on Cloud Services, AKS clusters, or Service Fabric.

These applications frequently use dependencies like Event Hubs, Storage, SQL, and Service Bus.

* Combine resources in Azure Resource Groups to get complete visibility of all the resources that make up your different applications. [Azure Monitor for Resource Groups](https://learn.microsoft.com/en-us/azure/azure-monitor/insights/resource-group-insights) provides a simple way to keep track of the health and performance of your entire full-stack application and enables drilling down into respective components for any investigations or debugging.

Ensure quality through continuous deployment

Continuous Integration / Continuous Deployment allows you to automatically integrate and deploy code changes to your application based on automated testing results.

It streamlines the deployment process and ensures the quality of any changes before they move into production.

* Use [Azure Pipelines](https://learn.microsoft.com/en-us/azure/devops/pipelines) to implement Continuous Deployment and automate your entire process from code commit to production based on your CI/CD tests.
* Use Quality Gates to integrate monitoring into your pre-deployment or post-deployment. It ensures that you meet the key health/performance metrics (KPIs) as your applications move from dev to production. Any differences in the infrastructure environment or scale aren't negatively impacting your KPIs.
* [Maintain separate monitoring instances](https://learn.microsoft.com/en-us/azure/application-insights/app-insights-separate-resources) between your different deployment environments, such as Dev, Test, Canary, and Prod. It ensures that collected data is relevant across the associated applications and infrastructure. If you need to correlate data across environments, use [multi-resource charts in Metrics Explorer](https://learn.microsoft.com/en-us/azure/azure-monitor/platform/metrics-charts) or create [cross-resource queries in Log Analytics](https://learn.microsoft.com/en-us/azure/azure-monitor/log-query/cross-workspace-query).

Create actionable alerts with actions

A critical monitoring aspect is proactively notifying administrators of current and predicted issues.

* Create [alerts in Azure Monitor](https://learn.microsoft.com/en-us/azure/azure-monitor/platform/alerts-overview) based on logs and metrics to identify predictable failure states. It would be best if you had a goal of making all alerts actionable, meaning that they represent actual critical conditions and seek to reduce false positives. Use [dynamic thresholds](https://learn.microsoft.com/en-us/azure/azure-monitor/platform/alerts-dynamic-thresholds) to automatically calculate baselines on metric data rather than defining your static thresholds.
* Define actions for alerts to use the most effective means of notifying your administrators. Available [actions for notification](https://learn.microsoft.com/en-us/azure/azure-monitor/platform/action-groups#create-an-action-group-by-using-the-azure-portal) are SMS, e-mails, push notifications or voice calls.
* Use more advanced actions to [connect to your ITSM tool](https://learn.microsoft.com/en-us/azure/azure-monitor/platform/itsmc-overview) or other alert management systems through [webhooks](https://learn.microsoft.com/en-us/azure/azure-monitor/platform/activity-log-alerts-webhook).
* Remediate situations identified in alerts with [Azure Automation runbooks](https://learn.microsoft.com/en-us/azure/automation/automation-webhooks) or [Logic Apps](https://learn.microsoft.com/en-us/connectors/custom-connectors/create-webhook-trigger) that can be launched from an alert using webhooks.
* Use [autoscaling](https://learn.microsoft.com/en-us/azure/azure-monitor/learn/tutorial-autoscale-performance-schedule) to dynamically increase and decrease your compute resources based on collected metrics.

Prepare dashboards and workbooks

Ensuring that your development and operations have access to the same telemetry and tools allows them to view patterns across your entire environment and minimize your Mean Time To Detect (MTTD) and Mean Time To Restore (MTTR).

* Prepare [custom dashboards](https://learn.microsoft.com/en-us/azure/application-insights/app-insights-tutorial-dashboards) based on standard metrics and logs for the different roles in your organization. Dashboards can combine data from all Azure resources.
* Prepare [Workbooks](https://learn.microsoft.com/en-us/azure/application-insights/app-insights-usage-workbooks) to ensure knowledge sharing between development and operations. It could be prepared as dynamic reports with metric charts and log queries or as troubleshooting guides designed by developers to help customer support or operations handle fundamental problems.

Continuously optimize

Monitoring is one of the fundamental aspects of the popular Build-Measure-Learn philosophy, which recommends continuously tracking your KPIs and user behavior metrics and optimizing them through planning iterations.

Azure Monitor helps you collect metrics and logs relevant to your business and add new data points in the following deployment.

* Use tools in Application Insights to [track end-user behavior and engagement](https://learn.microsoft.com/en-us/azure/application-insights/app-insights-tutorial-users).
* Use [Impact Analysis](https://learn.microsoft.com/en-us/azure/application-insights/app-insights-usage-impact) to help you prioritize which areas to focus on to drive to important KPIs.

Next unit: Explore Azure Monitor and Log Analytics

Explore Azure Monitor and Log Analytics

100 XP

* 6 minutes

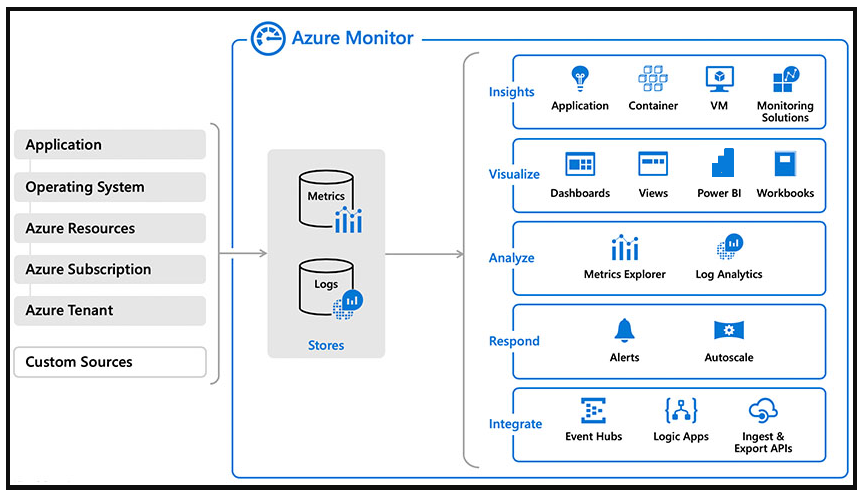
When you run at a cloud scale, you need intelligent logging and monitoring tools that scale to your needs and provide real-time insight into your data.

Azure Monitor is Microsoft's native cloud monitoring solution. Azure Monitor collects monitoring telemetry from different kinds of on-premises and Azure sources.

Azure Monitor provides Management tools, such as those in Azure Security Center and Azure Automation, enabling ingestion of custom log data to Azure.

The service aggregates and stores this telemetry in a log data store optimized for cost and performance.

With Azure Monitor, you can analyze data, set up alerts, and get end-to-end views of your applications. And use machine-learning-driven insights to identify and resolve problems quickly.



In this tutorial, we focus on the Log Analytics part of Azure Monitor. You'll learn how to:

* Set up Log Analytics workspace.
* Connect virtual machines to a log analytics workspace.
* Configure Log Analytics workspace to collect custom performance counters.
* Analyze the telemetry-using Kusto Query Language.

Getting started

1. You need a resource group with one or more virtual machines that you have access to RDP to follow along.
2. Log into [Azure Shell](https://shell.azure.com/powershell). Execute the command following. It creates a new resource group and log analytics workspace. Take a record of the workspaceid of the log analytics workspace as we're using it again.

PowerShellCopy

$ResourceGroup = "azwe-rg-devtest-logs-001"

$WorkspaceName = "azwe-devtest-logs-01"

$Location = "westeurope"

# List of solutions to enable

$Solutions = "CapacityPerformance", "LogManagement", "ChangeTracking", "ProcessInvestigator"

# Create the resource group if needed

try {

Get-AzResourceGroup -Name $ResourceGroup -ErrorAction Stop

} catch {

New-AzResourceGroup -Name $ResourceGroup -Location $Location

}

# Create the workspace

New-AzOperationalInsightsWorkspace -Location $Location -Name $WorkspaceName -ResourceGroupName $ResourceGroup

# List all solutions and their installation status

Get-AzOperationalInsightsIntelligencePacks -ResourceGroupName $ResourceGroup -WorkspaceName $WorkspaceName

# Add solutions

foreach ($solution in $Solutions) {

Set-AzOperationalInsightsIntelligencePack -ResourceGroupName $ResourceGroup -WorkspaceName $WorkspaceName -IntelligencePackName $solution -Enabled $true

}

# List enabled solutions

(Get-AzOperationalInsightsIntelligencePacks -ResourceGroupName $ResourceGroup -WorkspaceName $WorkspaceName).Where({($\_.enabled -eq $true)})

# Enable IIS Log Collection using the agent

Enable-AzOperationalInsightsIISLogCollection -ResourceGroupName $ResourceGroup -WorkspaceName $WorkspaceName

# Windows Event

New-AzOperationalInsightsWindowsEventDataSource -ResourceGroupName $ResourceGroup -WorkspaceName $WorkspaceName -EventLogName "Application" -CollectErrors -CollectWarnings -Name "Example Application Event Log"

1. Retrieve the Log Analytics workspace secure key.

PowerShellCopy

Get-AzOperationalInsightsWorkspaceSharedKey `

-ResourceGroupName azwe-rg-devtest-logs-001 `

-Name azwe-devtest-logs-01

1. Map existing virtual machines with the Log Analytics workspace. The following query uses the wokspaceid and workspace-secret key of the log analytics workspace to install the Microsoft Enterprise Cloud Monitoring extension onto an existing VM.

PowerShellCopy

$PublicSettings = @{"workspaceId" = "<myWorkspaceId>"}

$ProtectedSettings = @{"workspaceKey" = "<myWorkspaceKey>"}

Set-AzVMExtension -ExtensionName "Microsoft.EnterpriseCloud.Monitoring" `

-ResourceGroupName "azwe-rg-devtest-logs-001" `

-VMName "azsu-d-sql01-01" `

-Publisher "Microsoft.EnterpriseCloud.Monitoring" `

-ExtensionType "MicrosoftMonitoringAgent" `

-TypeHandlerVersion 1.0 `

-Settings $PublicSettings `

-ProtectedSettings $ProtectedSettings `

-Location westeurope

1. Run the script to configure the below-listed performance counters to be collected from the virtual machine.

PowerShellCopy

#Login-AzureRmAccount

#Instance

##################################

$InstanceNameAll = "\*"

$InstanceNameTotal = '\_Total'

#Objects

##################################

$ObjectCache = "Cache"

$ObjectLogicalDisk = "LogicalDisk"

$ObjectMemory = "Memory"

$ObjectNetworkAdapter = "Network Adapter"

$ObjectNetworkInterface = "Network Interface"

$ObjectPagingFile = "Paging File"

$ObjectProcess = "Process"

$ObjectProcessorInformation = "Processor Information"

$ObjectProcessor = "Processor"

$ObjectSQLAgentAlerts = "SQLAgent:Alerts"

$ObjectSQLAgentJobs = "SQLAgent:Jobs"

$ObjectSQLAgentStatistics = "SQLAgent:Statistics"

$ObjectSQLServerAccessMethods = "SQLServer:Access Methods"

$ObjectSQLServerExecStatistics = "SQLServer:Exec Statistics"

$ObjectSQLServerLocks = "SQLServer:Locks"

$ObjectSQLServerSQLErrors = "SQLServer:SQL Errors"

$ObjectSystem = "System"

#Counters

#########################################################

$CounterCache = "Copy Read Hits %"

$CounterLogicalDisk =

"% Free Space" `

,"Avg. Disk sec/Read" `

,"Avg. Disk sec/Transfer" `

,"Avg. Disk sec/Write" `

,"Current Disk Queue Length" `

,"Disk Read Bytes/sec" `

,"Disk Reads/sec" `

,"Disk Transfers/sec" `

,"Disk Writes/sec"

$CounterMemory =

"% Committed Bytes In Use" `

,"Available MBytes" `

,"Page Faults/sec" `

,"Pages Input/sec" `

,"Pages Output/sec" `

,"Pool Nonpaged Bytes"

$CounterNetworkAdapter =

"Bytes Received/sec" `

,"Bytes Sent/sec"

$CounterNetworkInterface = "Bytes Total/sec"

$CounterPagingFile =

"% Usage" `

,"% Usage Peak"

$CounterProcess = "% Processor Time"

$CounterProcessorInformation =

"% Interrupt Time" `

,"Interrupts/sec"

$CounterProcessor = "% Processor Time"

$CounterProcessorTotal = "% Processor Time"

$CounterSQLAgentAlerts = "Activated alerts"

$CounterSQLAgentJobs = "Failed jobs"

$CounterSQLAgentStatistics = "SQL Server restarted"

$CounterSQLServerAccessMethods = "Table Lock Escalations/sec"

$CounterSQLServerExecStatistics = "Distributed Query"

$CounterSQLServerLocks = "Number of Deadlocks/sec"

$CounterSQLServerSQLErrors = "Errors/sec"

$CounterSystem = "Processor Queue Length"

#########################################################

$global:number = 1 #Name parameter needs to be unique that why we'll use number ++ in function

#########################################################

function AddPerfCounters ($PerfObject, $PerfCounters, $Instance)

{

ForEach ($Counter in $PerfCounters)

{

New-AzOperationalInsightsWindowsPerformanceCounterDataSource `

-ResourceGroupName 'azwe-rg-devtest-logs-001' `

-WorkspaceName 'azwe-devtest-logs-01' `

-ObjectName $PerfObject `

-InstanceName $Instance `

-CounterName $Counter `

-IntervalSeconds 10 `

-Name "Windows Performance Counter $global:number"

$global:number ++

}

}

AddPerfCounters -PerfObject $ObjectLogicalDisk -PerfCounter $CounterLogicalDisk -Instance $InstanceNameAll

AddPerfCounters -PerfObject $ObjectNetworkAdapter -PerfCounter $CounterNetworkAdapter -Instance $InstanceNameAll

AddPerfCounters -PerfObject $ObjectNetworkInterface -PerfCounter $CounterNetworkInterface -Instance $InstanceNameAll

AddPerfCounters -PerfObject $ObjectPagingFile -PerfCounter $CounterPagingFile -Instance $InstanceNameAll

AddPerfCounters -PerfObject $ObjectProcess -PerfCounter $CounterProcess -Instance $InstanceNameAll

AddPerfCounters -PerfObject $ObjectProcessorInformation -PerfCounter $CounterProcessorInformation -Instance $InstanceNameAll

AddPerfCounters -PerfObject $ObjectProcessor -PerfCounter $CounterProcessor -Instance $InstanceNameAll

AddPerfCounters -PerfObject $ObjectProcessor -PerfCounter $CounterProcessorTotal -Instance $InstanceNameTotal

AddPerfCounters -PerfObject $ObjectSQLAgentAlerts -PerfCounter $CounterSQLAgentAlerts -Instance $InstanceNameAll

AddPerfCounters -PerfObject $ObjectSQLAgentJobs -PerfCounter $CounterSQLAgentJobs -Instance $InstanceNameAll

AddPerfCounters -PerfObject $ObjectSQLAgentStatistics -PerfCounter $CounterSQLAgentStatistics -Instance $InstanceNameAll

AddPerfCounters -PerfObject $ObjectSQLServerAccessMethods -PerfCounter $CounterSQLServerAccessMethods -Instance $InstanceNameAll

AddPerfCounters -PerfObject $ObjectSQLServerExecStatistics -PerfCounter $CounterSQLServerExecStatistics -Instance $InstanceNameAll

AddPerfCounters -PerfObject $ObjectSQLServerLocks -PerfCounter $CounterSQLServerLocks -Instance $InstanceNameAll

AddPerfCounters -PerfObject $ObjectSQLServerSQLErrors -PerfCounter $CounterSQLServerSQLErrors -Instance $InstanceNameAll

AddPerfCounters -PerfObject $ObjectSystem -PerfCounter $CounterSystem -Instance $InstanceNameAll

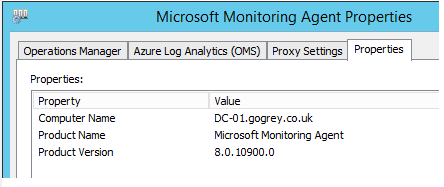
AddPerfCounters -PerfObject $ObjectMemory -PerfCounter $CounterMemory -Instance $InstanceNameAll

AddPerfCounters -PerfObject $ObjectCache -PerfCounter $CounterCache -Instance $InstanceNameAll

1. To generate some interesting performance statistics. Download the [HeavyLoad utility](https://www.jam-software.com/heavyload/) (a free load testing utility) and run it on the virtual machine to simulate high CPU, Memory, and IOPS consumption.

How it works

1. Log Analytics works by running the Microsoft Monitoring Agent service on the machine. The service locally captures and buffers the events and pushes them securely out to the Log Analytics workspace in Azure.
2. Log in to the virtual machine, navigate to the C:\Program Files\Microsoft Monitoring Agent\MMA, and open the control panel. It shows you the details of the log analytics workspace connected. You also can add multiple log analytics workspaces to publish the log data into various workspaces.



Summary

So far, we've created a log analytics workspace in a resource group.

The log analytics workspace has been configured to collect performance counters, event logs, and IIS Logs.

The Microsoft Enterprise cloud monitoring extension has mapped a virtual machine to the log analytics workspace.

HeavyLoad has been used to simulate high CPU, memory, and IOPS on the virtual machine.

Next unit: Examine Kusto Query Language (KQL)

**Examine Kusto Query Language (KQL)**

Completed100 XP

* 5 minutes

Kusto is the primary way to query Log Analytics. It provides both a query language and a set of control commands.

Kusto can be used directly within Azure Data Explorer.

Azure Data Studio also offers a Kusto query experience and supports the creation of Jupiter-style notebooks for Kusto queries.

See [Getting Started with Kusto Queries.](https://learn.microsoft.com/en-us/azure/data-explorer/kusto/concepts/)

**Walkthrough**

Note: This walkthrough continues the previous lesson on Azure Log Analytics, and the walkthrough started within it.

1. Log in to the [Azure portal](https://portal.azure.com/) and navigate to the log analytics workspace. From the left blade in the log analytics workspace, click Logs. It will open the Logs window, ready for you to start exploring all the data points captured into the workspace.
2. We'll need to use the Kusto Query Language to query the logs. Run the following query to list the last heartbeat of each machine connected to the log analytics workspace.

C#Copy

// Last heartbeat of each computer

// Show the last heartbeat sent by each computer

Heartbeat

| summarize arg\_max(TimeGenerated, \*) by Computer

1. Show a list of all distinct counters being captured.

C#Copy

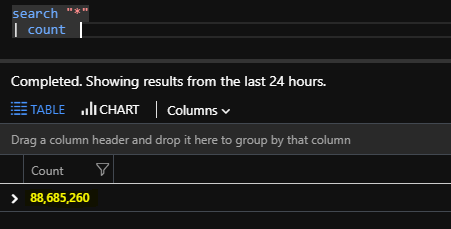
// What data is being collected?

// List the collected performance counters and object types (Process, Memory, Processor.)

Perf

| summarize by ObjectName, CounterName

1. Show a count of the data points collected in the last 24 hours. The result shows that we have 88M data points. We can query against them in near real-time to analyze and correlate insights.



1. Run the following query to generate the max CPU Utilization trend over the last 24 hours, aggregated at a granularity of 1 min. Render the data as a time chart.

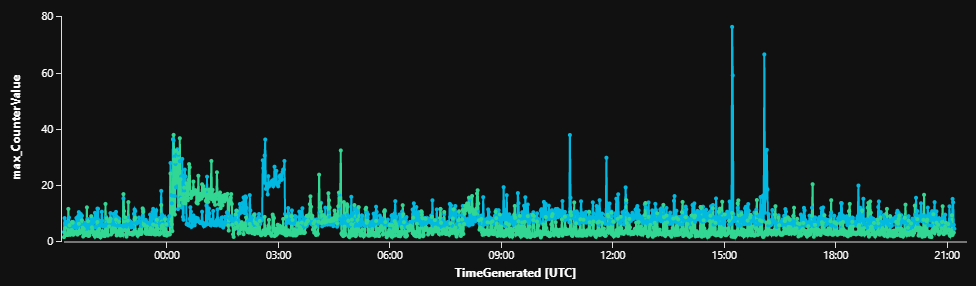
C#Copy

Perf

| where ObjectName == "Processor" and InstanceName == "\_Total"

| summarize max(CounterValue) by Computer, bin(TimeGenerated, 1m)

| render timechart



1. Run the following query to see all the processes running on that machine contributing to the CPU Utilization. Render the data in a pie chart.

C#Copy

Perf

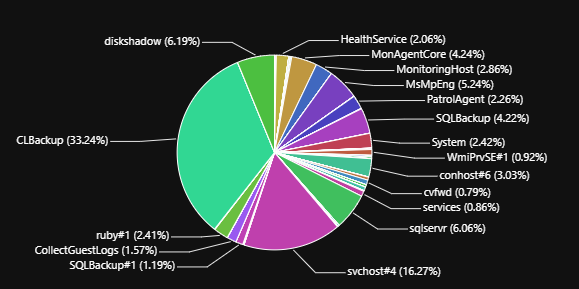
| where ObjectName contains "process"

and InstanceName !in ("\_Total", "Idle")

and CounterName == "% Processor Time"

| summarize avg(CounterValue) by InstanceName, CounterName, bin(TimeGenerated, 1m)

| render piechart



**There's more**

This unit has introduced the basic concepts of Log Analytics and how to get started with the basics.

We've only scratched the surface of what is possible with Log Analytics.

We would encourage you to try out the advanced tutorials available for Log Analytics on [Microsoft Learn](https://learn.microsoft.com/en-us/azure/azure-monitor/).

**Next unit: Explore Application Insights**

**Explore Application Insights**

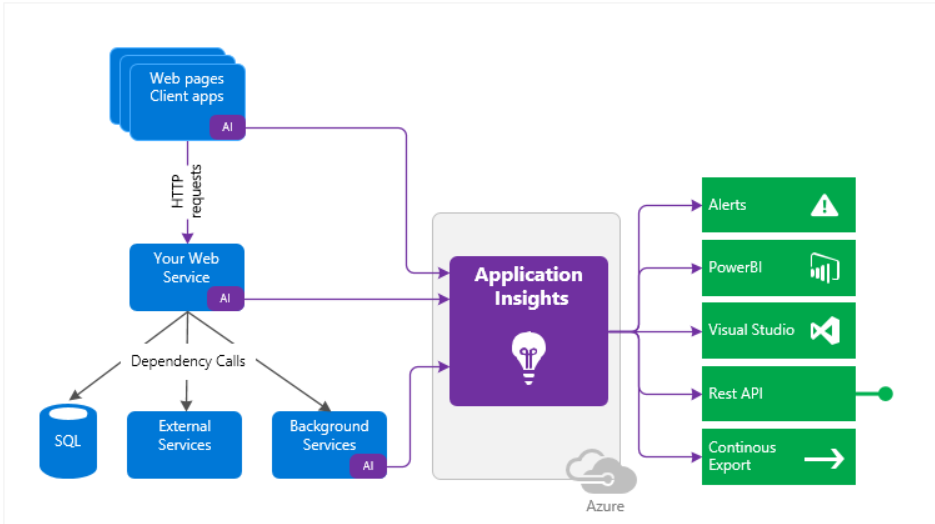
Completed100 XP

* 5 minutes

You install a small instrumentation package in your application and set up an Application Insights resource in the Microsoft Azure portal.

The instrumentation monitors your app and sends telemetry data to the portal. (The application can run anywhere - it doesn't have to be hosted in Azure.)

You can instrument the web service application, background components, and JavaScript in the web pages.



Also, you can pull in telemetry from the host environments such as performance counters, Azure diagnostics, or Docker logs.

You can also set up web tests periodically, sending synthetic requests to your web service.

All these telemetry streams are integrated into the Azure portal, where you can apply powerful analytic and search tools to the raw data.

**What's the overhead?**

The impact on your app's performance is minimal. Tracking calls are non-blocking and are batched and sent in a separate thread.

**What do Application Insights monitor?**

Application Insights is aimed at the development team to help you understand how your app is doing and being used. It monitors:

* Request rates, response times, and failure rates - Find out which pages are most popular, at what times of day, and where your users are. See which pages do best. If your response times and failure rates increase with more requests, perhaps you 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, pick specific instances, and drill into the stack trace and related requests. Both server and browser exceptions are reported.
* Pageviews and load performance - reported by your users' browsers.
* AJAX calls from web pages - rates, response times, and failure rates.
* User and session count.
* Performance counters from your Windows or Linux server machines include CPU, memory, and network usage.
* Host diagnostics from Docker or Azure.
* Diagnostic trace logs from your app - so you can correlate trace events with requests.
* Custom events and metrics that you write yourself in the client or server code to track business events such as items sold or games won.

**Where do I see my telemetry?**

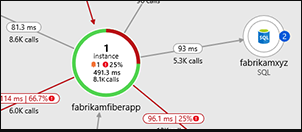
There are plenty of ways to explore your data. Check out this article for more information - [Smart detection and manual alerts](https://learn.microsoft.com/en-us/azure/application-insights/app-insights-proactive-diagnostics).

Automatic alerts adapt to your app's usual patterns of telemetry and trigger when there's something outside the usual pattern. You can also [set alerts](https://learn.microsoft.com/en-us/azure/azure-monitor/app/alerts) on levels of custom or standard metrics.



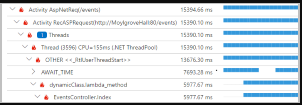
**Application map**

The components of your app, with key metrics and alerts.



**Profiler**

Inspect the execution profiles of sampled requests.



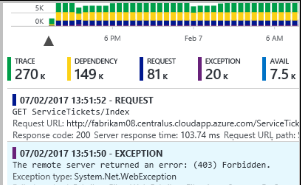
**Usage analysis**

Analyze user segmentation and retention.



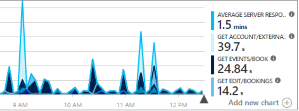
**Diagnostic search, for instance, data.**

Search and filter events such as requests, exceptions, dependency calls, log traces, and page views.



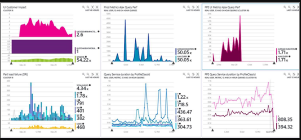
**Metrics Explorer for aggregated data**

Explore, filter, and segment aggregated data such as rates of requests, failures, exceptions, response times, and page load times.



**Dashboards**

Mash up data from multiple resources and share it with others. Great for multi-component applications and continuous display in the team room.



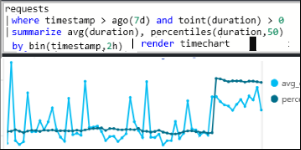
**Live Metrics Stream**

When you deploy a new build, watch these near-real-time performance indicators to ensure everything works as expected.



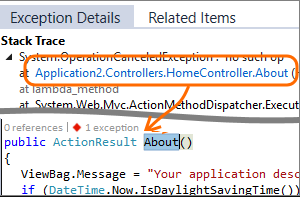
**Analytics**

Answer challenging questions about your app's performance and usage by using this powerful query language.



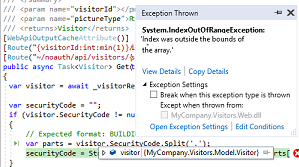
**Visual Studio**

See performance data in the code. Go to code from stack traces.



**Snapshot debugger**

Debug snapshots sampled from live operations, with parameter values.



**Power BI**

Integrate usage metrics with other business intelligence.



**REST API**

Write code to run queries over your metrics and raw data.



**Continuous export**

Bulk export of raw data to storage as soon as it arrives.



**Next unit: Implement Application Insights**

**Implement Application Insights**

Completed100 XP

* 3 minutes

**Monitor**

Install Application Insights in your app, set up [availability web tests](https://learn.microsoft.com/en-us/azure/application-insights/app-insights-monitor-web-app-availability), and:

* Set up a [dashboard](https://learn.microsoft.com/en-us/azure/azure-monitor/app/app-insights-dashboards) for your team room to keep an eye on load, responsiveness, and the performance of your dependencies, page loads, and AJAX calls.
* Discover which are the slowest and most-failing requests.
* Watch [Live Stream](https://learn.microsoft.com/en-us/azure/application-insights/app-insights-live-stream) when you deploy a new release to know immediately about any degradation.

**Detect, Diagnose**

If you receive an alert or discover a problem:

* Assess how many users are affected.
* Correlate failures with exceptions, dependency calls, and traces.
* Examine profiler, snapshots, stack dumps, and trace logs.

**Build, Measure, Learn**

[Measure the effectiveness](https://learn.microsoft.com/en-us/azure/application-insights/app-insights-usage-overview) of each new feature that you deploy.

* Plan to measure how customers use new UX or business features.
* Write custom telemetry into your code.
* Base the next development cycle on hard evidence from your telemetry.

**Get started**

Application Insights is one of the many services hosted within Microsoft Azure, and telemetry is sent there for analysis and presentation.

So, before you do anything else, you'll need a subscription to [Microsoft Azure](https://azure.com/).

It's free to sign up, and if you choose the basic [pricing plan](https://azure.microsoft.com/pricing/details/application-insights/) of Application Insights, there's no charge until your application has grown to have large usage.

If your organization already has a subscription, they could add your Microsoft account to it.

There are several ways to get started. Begin with whichever works best for you. You can add the others later.

**At run time**

Instrument your web app on the server. Avoids any update to the code. You need admin access to your server.

* [IIS on-premises or on a VM](https://learn.microsoft.com/en-us/azure/application-insights/app-insights-monitor-performance-live-website-now)
* [Azure web app or VM](https://learn.microsoft.com/en-us/azure/azure-monitor/app/azure-vm-vmss-apps)
* [J2EE](https://learn.microsoft.com/en-us/azure/application-insights/app-insights-java-live)

**At development time**

Add Application Insights to your code. Allows you to write custom telemetry and to instrument back-end and desktop apps.

* [Visual Studio](https://learn.microsoft.com/en-us/azure/azure-monitor/app/asp-net) 2013 update two or later.
* [Java](https://learn.microsoft.com/en-us/azure/application-insights/app-insights-java-get-started)
* Node.js
* [Other platforms](https://learn.microsoft.com/en-us/azure/application-insights/app-insights-platforms)
* [Instrument your web pages](https://learn.microsoft.com/en-us/azure/application-insights/app-insights-javascript) for page view, and another client-side telemetry.
* [Analyze mobile app usage](https://learn.microsoft.com/en-us/azure/application-insights/app-insights-mobile-center-quickstart) by integrating with Visual Studio App Center.
* [Availability tests](https://learn.microsoft.com/en-us/azure/application-insights/app-insights-monitor-web-app-availability) - ping your website regularly from our servers.

**Next unit: Design and implement metrics and queries**

**Design and implement metrics and queries**

Completed100 XP

* 3 minutes

Collecting metrics associated with project lifecycle allows organizations to gain insights into their software development practices and streamline their DevOps processes. By enabling telemetry data collection from project activities, teams can track pipeline runs, work item updates, and other DevOps-related events in real-time. This approach is important for enhancing visibility, monitoring performance, and optimizing the software development lifecycle. It empowers teams to make data-driven decisions, identify bottlenecks in the project workflow, and continuously improve their development practices.

Both GitHub and Azure DevOps include support for collecting and analyzing project-related telemetry. GitHub implements a few built-in metrics that reflect repository activity, pull request performance, and community engagement through its Insights feature, but its project tracking and analytics capabilities are limited. Azure DevOps, on the other hand, offers a comprehensive set of analytics and reporting features specifically designed to support project monitoring and management. It provides customizable dashboards, rich reporting capabilities, and a wide range of predefined metrics and queries that facilitate tracking work progress throughout the software development lifecycle.

In addition to the telemetry functionality built into GitHub and Azure DevOps, you can extend the scope of monitoring by taking advantage of Azure services such as Azure Monitor and Application Insights. In this case, Azure DevOps also offers richer and more streamlined integration than GitHub.

**Using GitHub built-in project tracking**

In GitHub projects, tracking progress involves monitoring metrics such as issue throughput, issue aging, code review turnaround time, pull request merge time, and contributor activity. These metrics offer insights into project health, team efficiency, and code quality, helping with identifying areas for improvement and project execution.

The simplest way to review metrics in GitHub projects is to use the Insights functionality, which supports visualizing repository activity, pull request performance, and community engagement.

In addition, it is possible to create custom GitHub Actions workflows that automate the collection and processing of project metrics. Such workflows can be triggered based on events such as pull requests, issues, or code pushes, and include actions to collect and analyze metrics.

Another option is to leverage GitHub REST API to programmatically access project data, including metrics related to issues, pull requests, and contributors. Such data can serve as the basis for building custom dashboards or reports using other analytics tools.

**Using Azure DevOps built-in project tracking**

Azure DevOps offers a large number of built-in metrics that reflect various aspects of project management, such as cycle time, lead time, workflow efficiency, and backlog health. In general, these metrics can be grouped into several categories:

* Work Item Metrics:
  + Count of Work Items: Monitor the total number of work items created, completed, or in progress over time.
  + Work Item Age: Track the age of open work items to identify overdue tasks.
  + Velocity: Measure the rate at which work items are completed over iterations or sprints.
* Build and Release Metrics:
  + Build Success Rate: Monitor the percentage of successful builds versus failed builds.
  + Release Deployment Frequency: Track how frequently releases are deployed to production or other environments.
  + Deployment Success Rate: Measure the percentage of successful deployments to identify any deployment issues.
* Test Metrics:
  + Test Case Pass Rate: Monitor the percentage of passed test cases versus failed test cases.
  + Test Case Execution Time: Track the time taken to execute test cases to identify performance bottlenecks.
  + Test Case Failure Trends: Analyze trends in test case failures to identify recurring issues.
* Code Metrics:
  + Code Churn: Measure the amount of code added, modified, or deleted over time.
  + Code Quality Metrics: Monitor code quality metrics such as code coverage, cyclomatic complexity, and maintainability index.
* Team Performance Metrics:
  + Sprint Burndown: Track the progress of sprint goals by monitoring the remaining work versus time.
  + Team Velocity: Measure the average amount of work completed by the team in each sprint.
  + Lead Time: Monitor the time taken from the creation of a work item to its completion.

You can display and analyze these metrics by using dashboards accessible directly within Azure DevOps portal. For additional insights into project progress and performance, consider creating custom, query-based reports in Azure DevOps Analytics. Some of the more commonly used queries, groupings, and visualizations that provide meaningful insights into Azure DevOps Projects include:

* **Cycle Time Analysis**
  + Query to calculate the cycle time (time taken to complete a work item from start to finish) for different types of work items (e.g., user stories, bugs) over time.
  + Group the work items by their state (e.g., Active, Resolved, Closed).
  + Calculate the time difference between the creation date and the completion date of each work item.
  + Visualize the average cycle time per state over time to identify bottlenecks and improve process efficiency.
* **Lead Time Distribution**
  + Query to analyze the lead time (time taken from the creation of a work item to its completion) distribution for different teams or areas of the project.
  + Group the work items by their area path or team assignment.
  + Calculate the lead time for each work item and visualize the distribution using a histogram or box plot.
  + Identify outliers and investigate the reasons for longer lead times in specific areas to optimize workflow and reduce delivery time.
* **Cumulative Flow Diagram (CFD)**
  + Query to create a Cumulative Flow Diagram that tracks the number of work items in each state (e.g., To Do, In Progress, Done) over time.
  + Group the work items by their state and order them by their date of state change.
  + Calculate the cumulative count of work items in each state at regular intervals (e.g., daily, weekly).
  + Visualize the CFD to monitor the flow of work items through the project stages and identify areas of congestion or delays.
* **Work Item Aging Analysis**
  + Query to analyze the aging of work items in the backlog or specific stages of the project.
  + Group the work items by their age (e.g., days since creation) and their priority or type.
  + Calculate metrics such as the average age of work items, the percentage of overdue work items, and the distribution of aging across different categories.
  + Visualize the aging trends to prioritize backlog grooming activities and ensure timely delivery of high-priority items.

**Using Azure Monitor and Application Insights for project tracking**

There are numerous ways to integrate project tracking with Azure Monitor and Application Insights, but one of the more commonly used approaches involves combining continuous monitoring, release annotations, and work items. This optimizes the software development lifecycle by providing valuable information on performance bottlenecks, facilitating proactive issue resolution, and enabling informed decision-making in order to successfully deliver high-quality software.

With continuous monitoring, release pipelines can incorporate monitoring data from Application Insights. Effectively, an Application Insights alert can block or rollback a gated deployment until the alert is resolved. Conversely, if monitoring checks pass, deployments can proceed automatically, without the need for manual intervention.

Annotations designate events in the CI/CD pipeline. They capture correlation with between delivery of software releases and the resulting application performance. Azure DevOps support automatic generation of annotations by build pipelines. You can also create arbitrary annotations by using PowerShell.

Work items integration functionality facilitates embedding contextual Application Insights data when generating work items in GitHub and Azure DevOps. The supports a number of features that facilitate data capture and work item generation, including:

* Advanced fields like assignee, projects, or milestones.
* Multiple configurations for any number of repositories or work items.
* Pre-built & customizable Keyword Query Language (KQL) queries to add Application Insights data to work items.
* Customizable workbook templates.

**Next unit: Monitor application performance with Application Insights**

**Monitor application performance with Application Insights**

Completed100 XP

* 60 minutes

**Estimated time:** 60 minutes.

**Scenario**

Application Insights is an extensible Application Performance Management (APM) service for web developers on multiple platforms. You can use it to monitor your live web applications. It automatically detects performance anomalies, includes powerful analytics tools to help you diagnose issues, and helps you continuously improve performance and usability. It works for apps on various platforms, including .NET, Node.js, and Java EE, hosted on-premises, hybrid, or any public cloud. It integrates with your DevOps process with connection points available in various development tools. It also allows you to monitor and analyze telemetry from mobile apps through integration with Visual Studio App Center.

In this lab, you'll learn how to add Application Insights to an existing web application and monitor the application via the Azure portal.

**Objectives**

After completing this lab, you'll be able to:

* Deploy Azure App Service web apps.
* Generate and monitor Azure web app application traffic by using Application Insights.
* Investigate Azure web app performance by using Application Insights.
* Track Azure web app usage by using Application Insights.
* Create Azure web app alerts by using Application Insights.

**Requirements**

* This lab requires **Microsoft Edge** or an [Azure DevOps-supported browser](https://learn.microsoft.com/en-us/azure/devops/server/compatibility).
* **Set up an Azure DevOps organization:** If you don't already have an Azure DevOps organization that you can use for this lab, create one by following the instructions available at [Create an organization or project collection](https://learn.microsoft.com/en-us/azure/devops/organizations/accounts/create-organization).
* Identify an existing Azure subscription or create a new one.
* Verify that you have a Microsoft or Microsoft Entra ID account with the **Owner** role in the Azure subscription and the **Global Administrator** role in the Microsoft Entra ID tenant associated with the Azure subscription. For details, refer to [List Azure role assignments using the Azure portal](https://learn.microsoft.com/en-us/azure/active-directory/roles/manage-roles-portal) and [View and assign administrator roles in Microsoft Entra ID](https://learn.microsoft.com/en-us/azure/active-directory/roles/manage-roles-portal).

**Exercises**

During this lab, you'll complete the following exercises:

* Exercise 0: Configure the lab prerequisites.
* Exercise 1: Monitor an Azure App Service web app using Azure Application Insights.
* Exercise 2: Remove the Azure lab resources.

[Screenshot of a launch button, which will take you to the lab.](https://go.microsoft.com/fwlink/?linkid=2270116)

**Next unit: Knowledge check**

**Knowledge check**

Completed200 XP

* **Module assessment**
* 5 minutes

 Answer 100% of questions correctly in order to pass. [**Retake**](https://learn.microsoft.com/en-us/training/modules/implement-tools-track-usage-flow/10-knowledge-check)

Dismiss alert

Choose the best response for each question.

**Check your knowledge**

Top of Form

**1.**

**Which of the following choices Azure Monitor lets you create custom dashboards based on?**

Alerts, Action Groups.

Metrics, Logs.

**Correct. Azure Monitor lets you create custom dashboards based on Metrics and Logs.**

Action Groups, Workflows.

**Incorrect. Azure Monitor lets you create custom dashboards based on Metrics and Logs.**

**2.**

**Which of the following query languages can you use to query Azure Log Analytics?**

Kusto.

**Correct. Kusto is the primary way to query Log Analytics. It provides both a query language and a set of control commands.**

T-SQL.

Xpath.

**3.**

**Which of the following choices is the unique key configured by Application Insights used by the Application Insights SDK?**

ApplicationInsights Key.

AppInsightsSDK Key.

**Incorrect. Application Insights configures a unique key called AppInsights Key in your application. The Application Insights SDK uses this key to identify the Azure App Insights workspace the telemetry data needs to be uploaded.**

AppInsights Key.

**Correct. Application Insights configures a unique key called AppInsights Key in your application. The Application Insights SDK uses this key to identify the Azure App Insights workspace the telemetry data needs to be uploaded.**

Bottom of Form

**Next unit: Summary**

[**Previous**](https://learn.microsoft.com/en-us/training/modules/implement-tools-track-usage-flow/9-monitor-application-performance-app-insights/)

**Summary**

Completed100 XP

* 1 minute

This module introduced you to continuous feedback practices and tools to track usage and flow, such as Azure Logs Analytics, Kusto Query Language (KQL), and Application Insights.

You learned how to describe the benefits and usage of:

* Implement tools to track feedback.
* Plan for continuous monitoring.
* Implement Application Insights.
* Use Kusto Query Language (KQL).

**Learn more**

* [Give feedback with Test & Feedback extension - Azure Test Plans | Microsoft Learn](https://learn.microsoft.com/en-us/azure/devops/test/provide-stakeholder-feedback).
* [Request stakeholder feedback - Azure Test Plans | Microsoft Learn](https://learn.microsoft.com/en-us/azure/devops/test/request-stakeholder-feedback).
* [Continuous monitoring with Azure Monitor - Azure Monitor | Microsoft Learn](https://learn.microsoft.com/en-us/azure/azure-monitor/continuous-monitoring).
* [Continuous monitoring of your DevOps release pipeline with Azure Pipelines and Azure Application Insights - Azure Monitor | Microsoft Learn](https://learn.microsoft.com/en-us/azure/azure-monitor/app/continuous-monitoring).
* [What is Azure Application Insights? - Azure Monitor | Microsoft Learn](https://learn.microsoft.com/en-us/azure/azure-monitor/app/app-insights-overview).
* [KQL quick reference | Microsoft Learn](https://learn.microsoft.com/en-us/azure/data-explorer/kql-quick-reference).

“”” <https://learn.microsoft.com/en-us/training/modules/implement-tools-track-usage-flow/11-summary#completion> “