**Chapter 4. Monitor, troubleshoot, and optimize Azure solutions**

Providing a good experience to your users is one of the key factors for the success of your application. Several factors affect the user’s experience, such as a good user interface design, ease of use, good performance, and low failure rate. You can ensure that your application will perform well by assigning more resources to your application, but if there are not enough users using your application, you might be wasting resources and money.

To ensure that your application is working correctly, you need to deploy a monitoring mechanism that helps you to get information about your application’s behavior. This is especially important during peak usage periods or failures. Azure provides several tools that help you to monitor, troubleshoot, and improve the performance of your application.

**Skills covered in this chapter:**

* [Skill 4.1: Integrate caching and content delivery within solutions](https://learning.oreilly.com/library/view/exam-ref-az-204/9780136798255/ch04.xhtml#ch04lev1sec1)
* [Skill 4.2: Instrument solutions to support monitoring and logging](https://learning.oreilly.com/library/view/exam-ref-az-204/9780136798255/ch04.xhtml#ch04lev1sec2)

**Skill 4.1: Integrate caching and content delivery within solutions**

Any web application that you implement delivers two types of content—dynamic and static.

* Dynamic content is the type of content that changes depending on user interaction. An example of dynamic content is a dashboard with several graphs or a list of user movements in a banking application.
* Static content is the same for all application users. Images and PDFs are examples of static content (as long as they are not dynamically generated) that users can download from your application.

If the users of your application access it from several locations across the globe, you can improve the performance of the application by delivering the content from the location nearest to the user. For static content, you can improve the performance by copying the content to different cache servers distributed across the globe. Using this technique, users can retrieve the static content from the nearest location with lower latency, which improves the performance of your application.

For dynamic content, you can use cache software to store the most accessed data. This means your application returns the information from the cache, which is faster than reprocessing the data or getting it from the storage system.

**This skill covers how to**

* [Develop code to implement CDNs in solutions](https://learning.oreilly.com/library/view/exam-ref-az-204/9780136798255/ch04.xhtml#ch04lev2sec1)
* [Configure cache and expiration policies for FrontDoor, CDNs, and Redis caches](https://learning.oreilly.com/library/view/exam-ref-az-204/9780136798255/ch04.xhtml#ch04lev2sec2)
* [Store and retrieve data in Azure Redis Cache](https://learning.oreilly.com/library/view/exam-ref-az-204/9780136798255/ch04.xhtml#ch04lev2sec3)

**Develop code to implement CDNs in solutions**

A Content Delivery Network (CDN) is a group of servers distributed in different locations across the globe that can deliver web content to users. Because the CDN has servers distributed in several locations, when a user makes a request to the CDN, the CDN delivers the content from the nearest server to the user.

The main advantage of using Azure CDN with your application is that Azure CDN caches your application’s static content. When a user makes a request to your application, the CDN stores the static content, such as images, documents, and stylesheet files. When a second user from the same location as the first user accesses your application, the CDN delivers the cached content, relieving your web server from delivering the static content. You can use third-party CDN solutions such as Verizon or Akamai with Azure CDN.

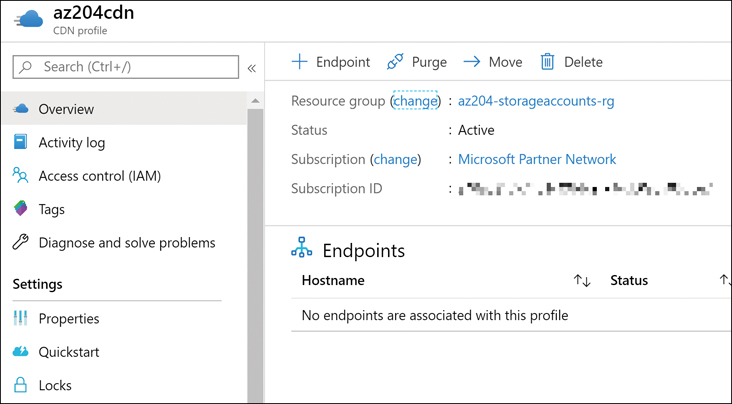
To use Azure CDN with your solution, you need to configure a profile. This profile contains the list of endpoints in your application that would be included in the CDN. The profile also configures the behavior of content delivery and access of each configured endpoint. When you configure an Azure CDN profile, you need to choose between using Microsoft’s CDN or using CDNs from Verizon or Akamai.

You can configure as many profiles as you need for grouping your endpoints based on different criteria, such as internet domain, web application, or any other criteria. Bear in mind that Azure CDN pricing tiers are applied at the profile level, so you can configure different profiles with different pricing characteristics. As with any CDN solution in the real world, you need a web application to run the procedures and demonstrations throughout this skill. The following procedure shows how to create a basic web application in Visual Studio and publish it in an Azure Web App. You can use this Azure Web App in all the examples in the rest of this skill:

1. Open Visual Studio 2019 on your computer.
2. In the Visual Studio 2019 home window, in the column named Get Started, click the Continue Without Code link at the bottom of the column.
3. Click the Tools menu and choose Get Tools And Features. Verify that the ASP.NET And Web Development In The Web & Cloud section is checked.
4. In the Visual Studio 2019 window, select File > New > Project to open the New Project window.
5. In the Create a New Project window, select C# in the drop-down menu below the Search For Templates text box at the top right of the window.
6. In the All Project Types drop-down menu, select Web.
7. In the list of templates on the right side of the window, select ASP.NET Core Web Application.
8. In the Configure Your New Project window, complete the following steps:
   1. Select a name for the project.
   2. Enter a path for the location of the solution.
   3. In the Solution drop-down menu, select Create A New Solution.
   4. Enter a name for the solution.
9. Click the Create button in the bottom-right corner of the Configure Your New Project window. This opens the Create A New ASP.NET Core Web Application window.
10. In the Create A New ASP.NET Core Web Application window, ensure that the following values are selected in the two drop-down menus at the top of the window:
    1. .NET Core
    2. ASP.NET Core 3.1
11. Select Web Application from the Project Templates area in the center of the window.
12. Uncheck the Configure For HTTPS option at the bottom right of the window.
13. Click the Create button in the bottom-right corner of the Create A New ASP.NET Core Web Application window.
14. On the right side of the Visual Studio window, in the Solution Explorer window, right-click the project’s name.
15. In the contextual menu, click Publish. This opens the Pick A Publish Target window.
16. In the Pick A Publish Target window, make sure that App Service is selected from the list of Available Targets on the left side of the window.
17. In the Azure App Service section, on the right side of the window, ensure that Create New Option is selected.
18. In the bottom-right corner of the window, click the Create Profile button, which opens the Create App Service window.
19. In the Create App Service window, add a new Azure account. This account needs to have enough privileges in the subscription for creating new resource groups, app services, and an App Service plan.
20. Once you have added a valid account, you can configure the settings for publishing your web application.
21. In the App Name text box, enter a name for the App Service. By default, this name matches the name that you gave to your project.
22. In the Subscription drop-down menu, select the subscription in which you want to create the App Service.
23. In the Resource Group drop-down menu, select the resource group in which you want to create the App Service and the App Service plan. If you need to create a new resource group, you can do so by clicking the New link on the right side of the drop-down menu.
24. To the right of the Hosting Plan drop-down menu, click the New link to open the Configure Hosting Plan window.
25. In the Configure Hosting Plan window, type a name for the App Service plan in the App Service Plan text box.
26. Select a region from the Location drop-down menu.
27. Select a virtual machine size from the Size drop-down menu.
28. Click the OK button in the bottom-right corner of the window. This closes the Configure Hosting Plan window.
29. At the bottom-right corner of the Create App Service window, click the Create button. This starts the creation of the needed resources and the upload of the code to the App Service.
30. Once the publishing process has finished, Visual Studio opens your default web browser with the URL of the newly deployed App Service. This URL will have the structure *https://<your\_app\_service\_name>.azurewebsites.net*.

Once you have created your testing Azure Web App, you can use the URL that you got on step 30 in the previous procedure with the rest of the procedures in this skill. The following procedure shows how to create an Azure CDN profile with one endpoint for caching content from a web application:

1. Open the Azure portal ([*https://portal.azure.com*](https://portal.azure.com/)).
2. Click the Create A Resource button in the Azure Services section.
3. On the New blade, in the Search The Marketplace text box, type CDN.
4. In the result list, click CDN.
5. On the CDN blade, click the Create button.
6. On the CDN profile blade, type a Name for the profile.
7. Select an existing Resource Group in the drop-down menu. Alternatively, you can create a new resource group by clicking the Create New link below the Resource Group drop-down menu.
8. In the Pricing Tier drop-down menu, select Standard Microsoft.
9. Click the Create button at the bottom of the CDN profile blade.
10. In the Search text box at the top of the Azure portal, type the name for your CDN profile.
11. In the result list, click the name of your CDN profile.
12. On the CDN profile blade, shown in [Figure 4-1](https://learning.oreilly.com/library/view/exam-ref-az-204/9780136798255/ch04.xhtml#ch04fig01), click the Endpoint button.



**Figure 4-1** CDN profile blade

1. In the Add An Endpoint panel, type a Name for the endpoint. Bear in mind that this name needs to be globally unique.
2. In the Origin Type drop-down menu, select Web App.
3. In the Origin Hostname drop-down menu, select the name of your web application.
4. In the Origin Path text box, type the path to the application you need to include in the CDN.
5. Leave the Origin Host header value as is. The Origin Host header value should match the Origin Hostname value.
6. Leave the other options as is they are.
7. Click the Add button.

The propagation of the content through the CDN depends on the type of CDN that you configured. For Standard Microsoft CDN, the propagation usually completes in 10 minutes. Once the propagation of the CDN completes, you can access your web application by using the endpoint that you configured in the previous procedure: *https://<your\_endpoint’s\_name>.azureedge.net*.

Once you have configured the endpoint, you can apply some advanced options to adjust the CDN to your needs:

* **Custom DNS domain** By default, when using the CDN, your users access your application by using the URL *https://<your\_endpoint’s\_name>.azureedge.net*. This URL would not be appropriate for your application. You can assign more appropriate DNS domains to the CDN endpoint, such as [*https://app.contoso.com*](https://app.contoso.com/), which allows your users to access your web application using a URL related to your business and your DNS domain name.
* **Compression** You can configure the CDN endpoint to compress some MIME types. This compression is made on the fly by the CDN when the content is delivered from the cache. Compressing the content allows you to deliver smaller files, improving the overall performance of the application.
* **Caching rules** You can control how the content is stored in the cache by setting different rules for different paths or content types. By configuring a cache rule, you can modify the cache expiration time, depending on the conditions you configure. Caching rules are only available for profiles from Verizon’s Azure CDN Standard and Akamai’s Azure CDN Standard.
* **Geo-filtering** You can block or allow a web application’s content to specific countries across the globe.
* **Optimization** You can configure the CDN for optimizing the delivery of different types of content. Depending on the type of profile, you can optimize your endpoint for
  + General web delivery
  + Dynamic site acceleration
  + General media streaming
  + Video-on-demand media streaming
  + Large file downloads

***NOTE* DYNAMIC SITE ACCELERATION**

Although Dynamic Site Acceleration is part of the features provided by the Azure CDN, this is not strictly a cache solution. If you need to use Dynamic Site Acceleration with Microsoft Azure services, you should use Azure Front Door Service instead of Azure CDN .

If you need to dynamically create new CDN profiles and endpoints, Microsoft provides the Azure CDN Library for .NET and Azure CDN Library for Node.js. Using these libraries, you can automate most of the operations reviewed in this section.

***NEED MORE REVIEW?* HOW CACHING WORKS**

Caching web content involves working with HTTP headers, setting the appropriate expiration times, or deciding which files should be included in the cache. You can review the details of how caching works by reading the article at [*https://docs.microsoft.com/en-us/azure/cdn/cdn-how-caching-works*](https://docs.microsoft.com/en-us/azure/cdn/cdn-how-caching-works).

**Images *EXAM TIP***

Content Delivery Networks (CDN) are appropriate for caching static content that changes infrequently. Although Azure CDN from Akamai and Azure CDN from Verizon include Dynamic Site Acceleration (DSA), this feature is not the same as a cache system. You should not confuse Azure CDN DSA optimization with Azure CDN cache.

**Configure cache and expiration policies for FrontDoor, CDNs, and Redis caches**

When you work with cached content, you need to control the lifetime or validity of that content. Although static content usually has a low rate of change, this kind of content can change. For example, if you are caching the logo of your company and the logo is changed, your users won’t see the change in the application until the new logo is loaded in the cache. In this scenario, you can simply purge or remove the old logo from the cache, and the new image will be loaded into the cache as soon as the first user accesses the application.

This mechanism of manually purging the cache could be appropriate for a very specific scenario. Still, in general terms, you should consider using an automatic mechanism for having the freshest content in your cache system. When you add content to a CDN cache, the system automatically assigns a TimeToLive (TTL) value to the content file instead of continuously comparing the file in the cache with the original content on the web server. The cache system checks whether the TTL is lower than the current time. If the TTL is lower than the current time, the CDN considers the content to be fresh and keeps the content in the cache. If the TTL expires, the CDN marks the content as stale or invalid. When the next user tries to access the invalid content file, the CDN compares the cached file with the content in the web server. If both files match, the CDN updates the version of the cached file and makes the file valid again by resetting the expiration time. If the files in the cache and the web server don’t match, the CDN removes the file from the cache and updates the content with the freshest content file on the web server.

The cached content can become invalid by deleting the content from the cache or by reaching the expiration time. You can configure the default TTL associated with a site by using the Cache-Control HTTP Header. You set the value for this header in different ways:

* **Default CDN configuration** If you don’t configure any value for the TTL, the Azure CDN automatically configures a default value of seven days.
* **Caching rules** You can configure TTL values globally or by using custom matching rules. Global caching rules affect all content in the CDN. Custom caching rules control the TTL for different paths or files in your web application. You can even disable the caching for some parts of your web application.
* **Web.config files** You use the web.config file to set the expiration time of the folder. You can even configure web.config files for different folders by setting different TTL values. Use the following XML code to set the TTL:

[Click here to view code image](https://learning.oreilly.com/library/view/exam-ref-az-204/9780136798255/ch04_images.xhtml#pg207a)

<configuration>

<system.webServer>

<staticContent>

<clientCache cacheControlMode="UseMaxAge" cacheControlMaxAge=

"3.00:00:00" />

</staticContent>

</system.webServer>

</configuration>

* **Programmatically** If you work with ASP.NET, you can control the CDN caching behavior by setting the HttpResponse.Cache property. You can use the following code to set the expiration time of the content to five hours:

[Click here to view code image](https://learning.oreilly.com/library/view/exam-ref-az-204/9780136798255/ch04_images.xhtml#pg208a)

// Set the caching parameters.

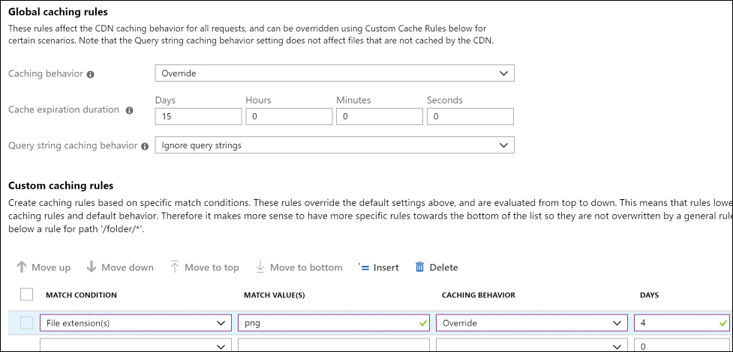
Response.Cache.SetExpires(DateTime.Now.AddHours(5));

Response.Cache.SetCacheability(HttpCacheability.Public);

Response.Cache.SetLastModified(DateTime.Now);

Use the following procedure to create caching rules in your Azure CDN. Bear in mind that you can configure caching rules only for Azure CDN for Verizon and Azure CDN for Akamai profiles:

1. Open the Azure portal ([*https://portal.azure.com*](https://portal.azure.com/)).
2. Click the Create A Resource button.
3. On the New blade, in the Search The Marketplace text box, type CDN.
4. In the result list, click CDN.
5. On the CDN blade, click the Create button.
6. On the CDN profile blade, type a Name for the profile.
7. Select an existing Resource Group from the drop-down menu. Alternatively, you can create a new resource group by clicking the Create New link below the Resource Group drop-down menu.
8. On the Pricing Tier drop-down menu, select Standard Akamai.
9. Check the Create A New CDN Endpoint Now check box.
10. Type a name for the endpoint in the CDN Endpoint Name text box. Beware that this name cannot be the same as the CDN Profile name.
11. In the Origin Type drop-down menu, select Web App.
12. In the Origin Hostname drop-down menu, select the name of your web application.
13. Click the Create button at the bottom of the CDN Profile blade.
14. In the Search text box at the top of the Azure portal, type the name of your CDN profile.
15. In the result list, click your CDN profile’s name.
16. On the Overview panel, on the CDN profile blade, in the Endpoints list, click the existing endpoint.
17. On the Endpoint blade, click Caching Rules in the Settings section of the navigation menu.
18. On the Caching Rules panel, shown in [Figure 4-2](https://learning.oreilly.com/library/view/exam-ref-az-204/9780136798255/ch04.xhtml#ch04fig02), set the Caching Behavior drop-down menu to Override in the Global Caching Rules section.



**Figure 4-2** Configuring Caching Rules

1. Set the Cache Expiration Duration to 15 days.
2. On the Custom Caching Rules list, create a new custom rule. Set the Match Condition drop-down menu to File Extension(s).
3. In the Match Value(s) text box, type **png**.
4. In the Caching Behavior drop-down menu, select Override.
5. In the Days column, type **4**.
6. In the top-left corner of the panel, click the Save button.

When you work with Azure Cache for Redis, you can also set the TTL for the different values stored in the in-memory database. If you don’t set a TTL for the key/value pair, the entry in the cache won’t expire. When you create a new entry in the in-memory database, you set the TTL value as a parameter of the StringSet() method. The following code snippet shows how to set a TTL of 5 hours to a String value:

[Click here to view code image](https://learning.oreilly.com/library/view/exam-ref-az-204/9780136798255/ch04_images.xhtml#pg209a)

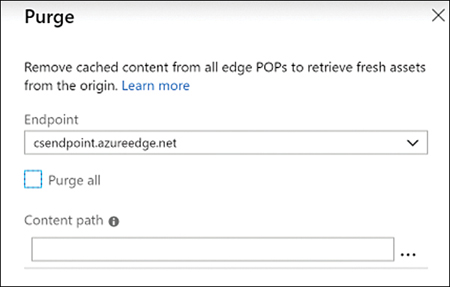
\_cache.StringSet(key, Serialize(value), new TimeSpan(5, 0, 0));

Apart from invalidating the content of the cache by the expiration of the content, you can manually invalidate the content by removing it directly from the CDN or Redis Cache. You can remove a key from the Azure Cache for Redis in-memory database. You can use the following methods:

* **KeyDelete() method** Use this method for removing a single key from the database. You need to use this method with a database instance.
* **FlushAllDatabases()** method Use this method to remove all keys from all databases in the Azure Cache for Redis.

For Azure CDN, you can invalidate part or the entire content of the CDN profile by using the Purge option available in the Azure portal. Use the following procedure for purging content from your Azure CDN profile:

1. Open the Azure portal ([*https://portal.azure.com*](https://portal.azure.com/)).
2. In the Search text box at the top of the Azure portal, type the name of your CDN profile.
3. On the Overview panel, in your CDN profile blade, click the Purge button.
4. On the Purge panel, shown in [Figure 4-3](https://learning.oreilly.com/library/view/exam-ref-az-204/9780136798255/ch04.xhtml#ch04fig03), select the Endpoint you want to purge from the drop-down menu control.



**Figure 4-3** Purging content from the cache

1. In the Content Path text box, type the path that you want to purge from the cache. If you want to purge all the content from the cache, you need to check the Purge All check box.

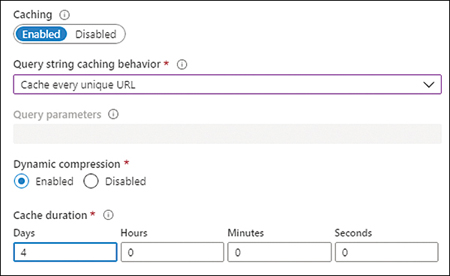
***NOTE* PURGE ALL AND WILDCARDS IN AZURE CDN FOR AKAMAI**

At the time of this writing, the Purge All and Wildcard options are not available for Akamai CDNs.

Azure CDN is not the only service that Microsoft provides for caching content. The Azure Front Door service allows you to route the traffic efficiently to the closest location to the user. As part of the features offered by the Azure Front Door service, it also allows you to cache content by providing a CDN. As with Azure CDN, you can configure the cache and expiration time for the elements in the cache.

The cache configuration is performed at routing rule level. Using the Azure Front Door service, you can route the traffic for different paths in your URL to different back-end pools hosting your application. A routing rule defines each of these routes. With this structure in mind, you can configure caching for some parts of your application, whereas others remain uncached. The following procedure shows how to enable caching in a routing rule. This procedure assumes that you have already deployed an Azure Front Door. Because we didn’t review how to work with Azure Front Door previously in this chapter, you can deploy a demo Front Door by using the quick start guide at [*https://docs.microsoft.com/en-us/azure/frontdoor/quickstart-create-front-door*](https://docs.microsoft.com/en-us/azure/frontdoor/quickstart-create-front-door)*.*

1. Open the Azure portal ([*https://portal.azure.com*](https://portal.azure.com/)).
2. In the Search Resources, Services, And Docs text box, type the name of your Azure Front Door instance.
3. Click the name of your Azure Front Door instance in the result list.
4. In your Azure Front Door blade, click Front Door Designer in the Settings section on the navigation menu on the left side of the blade.
5. On the Front Door Designer blade, click one of the routing rules inside the green rectangle with the title Routing Rules.
6. On the Update Routing Rule panel, scroll down to the bottom of the panel.
7. Change the Caching switch control from Disabled to Enabled.
8. In the cache settings shown in [Figure 4-4](https://learning.oreilly.com/library/view/exam-ref-az-204/9780136798255/ch04.xhtml#ch04fig04), change the value of Cache Duration from 0 Days to 4 Days.



**Figure 4-4** Configuring Azure Front Door cache

1. Click the Update button.
2. Click the Save button on the top-left corner of the Front Door Designer blade.

You can also control the cache expiration of an individual item by setting the appropriate cache headers. The following HTTP headers control the cache and expiration of an item in the Azure Front Door cache:

* **Cache-Control: max-age** Expressed in seconds, this header controls how long the item is valid in the cache. For example, if you set this value to 3600, the item can be used up to 60 minutes before the Azure Front Door service makes a request to the back-end pool for a fresh version of the item.
* **Cache-Control: s-maxage** Expressed in seconds, this directive is similar to the previous one but is meaningful only on CDN environments. This directive has precedence over max-age and expires directives.
* **Expires** Expressed using an HTTP-date timestamp, this directive sets the datetime until the item is valid in the cache. The max-age and s-maxage directives take precedence over this directive.

Purging the content of the cache is as simple as in the Azure CDN services. The following steps show how to purge the content of your Azure Front Door:

1. Open the Azure portal ([*https://portal.azure.com*](https://portal.azure.com/)).
2. In the Search Resources, Services, And Docs text box, type the name of your Azure Front Door instance.
3. Click the name of your Azure Front Door instance in the result list.
4. In your Azure Front Door blade, click Front Door Designer in the Settings section on the navigation menu on the left side of the blade.
5. On the Front Door Designer blade, click the Purge button on the top left side of the blade.
6. On the Purge panel, mark the Purge All check box. Alternatively, if you want to purge only part of the cached content, type the path of the content that you want to purge in the Content Path text box below the Purge All check box.
7. Click the Purge button at the bottom of the panel.

***NEED MORE REVIEW?* AZURE FRONT DOOR CACHING**

Azure Front Door is an advanced routing and caching system. This service allows you to cache big files and compress data on the fly. You can review more details about how Azure Front Door caching works by reading the article at [*https://docs.microsoft.com/en-us/azure/frontdoor/front-door-caching*](https://docs.microsoft.com/en-us/azure/frontdoor/front-door-caching).

**Store and retrieve data in Azure Redis Cache**

Redis is an open-source cache system that allows you to work *like in* an in-memory data structure store, database cache, or message broker. The Azure Redis Cache or Azure Cache for Redis is a Redis implementation managed by Microsoft. Azure Redis Cache has three pricing layers that provide you with different levels of features:

* **Basic** This is the tier with the fewest features and less throughput and higher latency. You should use this tier only for development or testing purposes. There is no Service Level Agreement (SLA) associated with the Basic tier.
* **Standard** This tier offers a two-node, primary-secondary replicated Redis cache that is managed by Microsoft. This tier has associated a high-availability SLA of 99.9 percent.
* **Premium** This is an enterprise-grade Redis cluster managed by Microsoft. This tier offers the complete group of features with the highest throughput and lower latencies. The Redis cluster is also deployed on more powerful hardware. This tier has a high-availability SLA of 99.9 percent.

***NOTE* SCALING THE AZURE REDIS CACHE SERVICE**

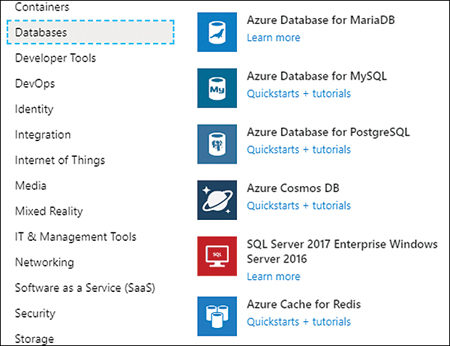
You can scale up your existing Azure Redis cache service to a higher tier, but you cannot scale down your current tier to a lower one .

When you are working with Azure Cache for Redis, you can use different implementation patterns that solve different issues, depending on the architecture of your application:

* **Cache-Aside** In most situations, your application stores the data that it manages in a database. Accessing data in a database is a relatively slow operation because it depends on the time to access the disk storage system. A solution would be to load the database in memory, but this approach is costly; in most cases, the database simply doesn’t fit on the available memory. One solution to improve the performance of your application in these scenarios is to store the most-accessed data in the cache. When the back-end system changes the data in the database, the same system can also update the data in the cache, which makes the change available to all clients.
* **Content caching** Most web applications use web page templates that use common elements, such as headers, footers, toolbars, menus, stylesheets, images, and so on. These template elements are static elements (or at least don’t change often). Storing these elements in Azure Cache for Redis relieves your web servers from serving these elements and improves the time your servers need to generate dynamic content.
* **User session caching** This pattern is a good idea if your application needs to register too much information about the user history or data that you need to associate with cookies. Storing too much information in a session cookie hurts the performance of your application. You can save part of that information in your database and store a pointer or index in the session cookie that points that user to the information in the database. If you use an in-memory database, such as Azure Cache for Redis, instead of a traditional database, your application benefits from the faster access times to the data stored in memory.
* **Job and message queuing** You can use Azure Cache for Redis to implement a distributed queue that executes long-lasting tasks that may negatively affect the performance of your application.
* **Distributed transactions** A transaction is a group of commands that need to complete or fail together. Any transaction needs to ensure that the data is always in a stable state. If your application needs to execute transactions, you can use Azure Cache for Redis for implementing these transactions.

You can work with Azure Cache for Redis using different languages, such as ASP.NET, .NET, .NET Core, Node.js, Java, or Python. Before you can add caching features to your code using Azure Redis Cache, you need to create your Azure Cache for Redis database using the following procedure:

1. Open the Azure portal ([*https://portal.azure.com*](https://portal.azure.com/)).
2. Click Create A Resource in the Azure Services section.
3. On the New blade, click Databases on the navigation menu on the left side of the blade.
4. In the list of Database services, shown in [Figure 4-5](https://learning.oreilly.com/library/view/exam-ref-az-204/9780136798255/ch04.xhtml#ch04fig05), click the Azure Cache For Redis item.



**Figure 4-5** Creating a new Azure Cache for Redis resource

1. On the New Redis Cache blade, type a DNS Name for your Redis resource.
2. Select the Subscription, Resource Group, and Location from the appropriate drop-down menu that best fits your needs.
3. In the Pricing tier drop-down menu, select the Basic C0 tier.
4. Click the Create button at the bottom of the New Redis Cache blade.

The deployment of your new Azure Cache for Redis takes a few minutes to complete. Once the deployment is complete, you need to get the access keys for your instance of the Azure Cache for Redis. You use this information in your code to connect the Redis service in Azure.

If you are using any of the .NET languages, you can use the StackExchange.Redis client for accessing your Azure Cache for Redis resource. You can also use this Redis client for accessing other Redis implementations. When reading or writing values in the Azure Cache for Redis, you need to create a ConnectionMultiplexer object. This object creates a connection to your Redis server. The ConnectionMultiplexer class is designed to be reused as much as possible.

For this reason, you should store this object and reuse it across all your code, whenever it is possible to reuse. Creating a connection is a costly operation. For this reason, you should not create a ConnectionMultiplexer object for each read or write operation to the Redis cache. Once you have created your ConnectionMultiplexer object, you can use any of the available operations in the StackExchange.Redis package. Following are the basic operations that you can use with Redis:

* **Use Redis as a database** You get a database from Redis, using the GetDatabase() method, for writing and reading values from the database. You use the StringSet() or StringGet() methods for writing and reading.
* **Use Redis as a messaging queue** You get a subscriber object from the Redis client, using the GetSubscriber() method. Then you can publish messages to a queue, using the Publish() method, and read messages from a queue, using the Subscribe() method. Queues in Redis are known as “channels.”

The following procedure shows how to connect to an Azure Cache for Redis database and read and write data to and from the database using an ASP.NET application:

1. Open Visual Studio 2019.
2. In the welcome window of Visual Studio 2019, on the Get Started column, click Create A New Project.
3. On the Create A New Project window, on the All Languages drop-down menu, select C#.
4. In the Search For Templates text box type **asp.net**.
5. In the result list, click ASP.NET Web Application (.NET Framework).
6. Click the Next button at the bottom right of the window.
7. On the Configure Your New Project, type a Project Name, a Location, and a Solution Name for your project.
8. Click the Create button at the bottom right of the window.
9. In the Create A New ASP.NET Web Application window, select the MVC template on the template list in the middle left side of the window. MVC is for Model-View-Controller.
10. On the right side of the Create A New ASP.NET Web Application window, in the Authentication section, ensure the Authentication is set to No Authentication.
11. Click the Create button at the bottom-right corner of the window.
12. In the Visual Studio window, select Tools > NuGet Package Manager > Manage NuGet Packages For Solution.
13. On the NuGet Package Manager tab, click Browse.
14. Type **StackExchange.Redis** and press Enter.
15. Click the StackExchange.Redis package.
16. On the right side of the NuGet Manager tab, click the check box next to your project.
17. Click the Install button.
18. In the Preview Changes window, click OK.
19. In the License Acceptance window, click the I Accept button.
20. Open the Azure portal ([*https://portal.azure.com*](https://portal.azure.com/)).
21. In the search text box in the top-middle of the portal, type the name of your Azure Cache for Redis that you created in the previous example.
22. Click your Azure Cache for Redis in the results list.
23. On the Azure Cache for Redis blade, click Access Keys in the Settings section in the navigation menu on the left side of the blade.
24. On the Access Keys blade, copy the value of the Primary Connection String (StackExchange.Redis). You need this value on the next steps.
25. In the Visual Studio window, open the Web.config file.
26. In the <appSettings> section, add the following code:

[Click here to view code image](https://learning.oreilly.com/library/view/exam-ref-az-204/9780136798255/ch04_images.xhtml#pg215a)

<add key="CacheConnection " value="<value\_copied\_in\_step\_24>"/>

***NOTE* SECURITY BEST PRACTICE**

In real-world development, you should avoid putting connection strings and secrets on files that could be checked with the rest of your code. To avoid this, you can put the <appSettings> section with the keys containing the sensible secrets or connection strings in a separate file outside the source code control folder. Then add the file parameter to the <appSettings> tag pointing to the external appSettings file path. You can also use the Azure App Configuration in conjunction with the Azure Key Vault for storing your connection strings.

1. Open the HomeController.cs file in the Controllers folder.
2. Add the following using statements to the HomeController.cs file:
3. using System.Configuration;

using StackExchange.Redis;

1. Add the code in [Listing 4-1](https://learning.oreilly.com/library/view/exam-ref-az-204/9780136798255/ch04.xhtml#ch04ex01) to the HomeController class.

**Listing 4-1** HomeController RedisCache method

[Click here to view code image](https://learning.oreilly.com/library/view/exam-ref-az-204/9780136798255/ch04_images.xhtml#lis4-1a)

**// C#. ASP.NET.**

public ActionResult RedisCache()

{

ViewBag.Message = "A simple example with Azure Cache for Redis on ASP.NET.";

var lazyConnection = new Lazy<ConnectionMultiplexer>(() =>

{

string cacheConnection = ConfigurationManager.AppSettings["CacheConnection"]

.ToString();

return ConnectionMultiplexer.Connect(cacheConnection);

});

// You need to create a ConnectionMultiplexer object for accessing the Redis

// cache.

// Then you can get an instance of a database.

IDatabase cache = lazyConnection.Value.GetDatabase();

// Perform cache operations using the cache object...

// Run a simple Redis command

ViewBag.command1 = "PING";

ViewBag.command1Result = cache.Execute(ViewBag.command1).ToString();

// Simple get and put of integral data types into the cache

ViewBag.command2 = "GET Message";

ViewBag.command2Result = cache.StringGet("Message").ToString();

// Write a new value to the database.

ViewBag.command3 = "SET Message \"Hello! The cache is working from ASP.NET!\"";

ViewBag.command3Result = cache.StringSet("Message", "Hello! The cache is working

from ASP.NET!").ToString();

// Get the message that we wrote on the previous step

ViewBag.command4 = "GET Message";

ViewBag.command4Result = cache.StringGet("Message").ToString();

// Get the client list, useful to see if the connection list is growing...

ViewBag.command5 = "CLIENT LIST";

ViewBag.command5Result = cache.Execute("CLIENT", "LIST").ToString().Replace(

"id=", "\rid=");

lazyConnection.Value.Dispose();

return View();

}

1. In the Solution Explorer, right-click Views > Home folder and select Add > View on the contextual menu.
2. In the Add View window, type **RedisCache** for the View Name.
3. Click the Add button.
4. Open the RedisCache.cshtml file.
5. Replace the content of the RedisCache.cshtml file with the content of [Listing 4-2](https://learning.oreilly.com/library/view/exam-ref-az-204/9780136798255/ch04.xhtml#ch04ex02).

**Listing 4-2** RedisCache View

[Click here to view code image](https://learning.oreilly.com/library/view/exam-ref-az-204/9780136798255/ch04_images.xhtml#lis4-2a)

**// C#. ASP.NET.**

@{

ViewBag.Title = "Azure Cache for Redis Test";

}

<h2>@ViewBag.Title.</h2>

<h3>@ViewBag.Message</h3>

<br /><br />

<table border="1" cellpadding="10">

<tr>

<th>Command</th>

<th>Result</th>

</tr>

<tr>

<td>@ViewBag.command1</td>

<td><pre>@ViewBag.command1Result</pre></td>

</tr>

<tr>

<td>@ViewBag.command2</td>

<td><pre>@ViewBag.command2Result</pre></td>

</tr>

<tr>

<td>@ViewBag.command3</td>

<td><pre>@ViewBag.command3Result</pre></td>

</tr>

<tr>

<td>@ViewBag.command4</td>

<td><pre>@ViewBag.command4Result</pre></td>

</tr>

<tr>

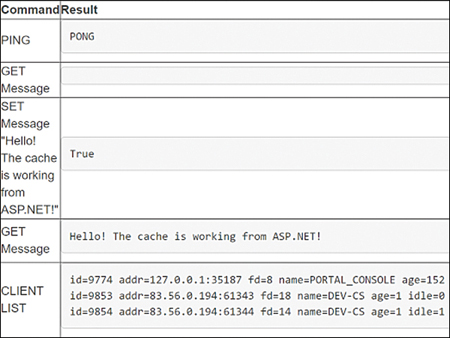
<td>@ViewBag.command5</td>

<td><pre>@ViewBag.command5Result</pre></td>

</tr>

</table>

1. Press F5 to run your project locally.
2. In the web browser running your project, append the */Home/RedisCache* URI to the URL. Your result should look like [Figure 4-6](https://learning.oreilly.com/library/view/exam-ref-az-204/9780136798255/ch04.xhtml#ch04fig06).



**Figure 4-6** Example results

**Images *EXAM TIP***

You can use Azure Cache for Redis for static content and the most-accessed dynamic data. You can use it for in-memory databases or message queues using a publication/subscription pattern.

***NEED MORE REVIEW?* MORE DETAILS ABOUT REDIS**

You can review features, patterns, and transactions of the Redis cache system by reading the following articles:

* [*https://stackexchange.github.io/StackExchange.Redis/Basics*](https://stackexchange.github.io/StackExchange.Redis/Basics)
* [*https://stackexchange.github.io/StackExchange.Redis/Transactions*](https://stackexchange.github.io/StackExchange.Redis/Transactions)
* [*https://stackexchange.github.io/StackExchange.Redis/KeysValues*](https://stackexchange.github.io/StackExchange.Redis/KeysValues)

**Skill 4.2: Instrument solutions to support monitoring and logging**

Knowing how your application behaves during regular operation is essential, especially for production environments. You need to get information about the number of users, resource consumption, transactions, and other metrics that can help you to troubleshoot your application if an error happens. Adding custom metrics to your application is also important when creating alerts that warn you when your application is not behaving as expected.

Azure provides features for monitoring the consumption of resources assigned to your application. Also, you can monitor the transactions and any other metrics that you may need, which allows you to fully understand how your application behaves under conditions that are usually difficult to simulate or test. You can also use these metrics for efficiently creating autoscale rules to improve the performance of your application.

**This skill covers how to**

* [Configure instrumentation in an app or service by using Application Insights](https://learning.oreilly.com/library/view/exam-ref-az-204/9780136798255/ch04.xhtml#ch04lev2sec4)
* [Analyze log data and troubleshoot solutions by using Azure Monitor](https://learning.oreilly.com/library/view/exam-ref-az-204/9780136798255/ch04.xhtml#ch04lev2sec5)
* [Implement Application Insights Web Test and Alerts](https://learning.oreilly.com/library/view/exam-ref-az-204/9780136798255/ch04.xhtml#ch04lev2sec6)
* [Implement code that handles transient fault](https://learning.oreilly.com/library/view/exam-ref-az-204/9780136798255/ch04.xhtml#ch04lev2sec7)

**Configure instrumentation in an app or service by using Application Insights**

Microsoft provides you with the ability to monitor your application while it is running by using Application Insights. This tool integrates with your code, allowing you to monitor what is happening inside your code while it is executing in a cloud, on-premises, or hybrid environment. You can also enable Application Insights for applications that are already deployed in Azure without modifying the already deployed code.

By adding a small instrumentation package, you can measure several aspects of your application. These measures, known as telemetry, are automatically sent to the Application Insight component deployed in Azure. Based on the information sent from the telemetry streams from your application to the Azure portal, you can analyze your application’s performance and create alerts and dashboards, which help you better understand how your application is behaving. Although Application Insights needs to be deployed in the Azure portal, your application can be executed in Azure, in other public clouds, or in your on-premises infrastructure. When you deploy the Application Insights instrumentation in your application, it monitors the following points:

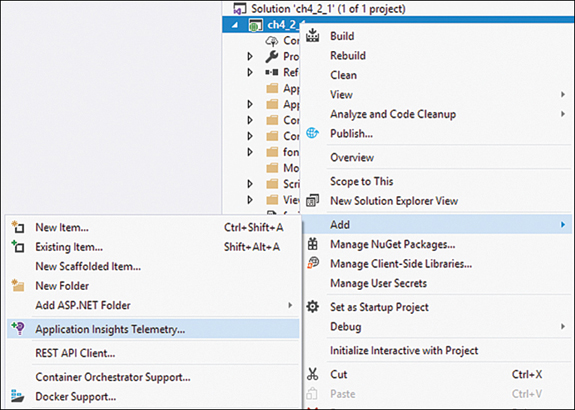
* **Request rates, response times, and failure rates** You can view which pages your users request more frequently, distributed across time. You may find that your users tend to visit specific pages at the beginning of the day, whereas other pages are more visited at the end of the day. You can also monitor the time that your server takes for delivering the requested page or even if there were failures when delivering the page. You should monitor the failure rates and response times to ensure that your application is performing correctly and your users have a pleasant experience.
* **Dependency rates, response times, and failure rates** If your application depends on external services (such as Azure Storage Accounts), Google or Twitter security services for authenticating your users, or any other external service, you can monitor how these external services are performing and how they are affecting your application.
* **Exceptions** The instrumentation keeps track of the exceptions raised by servers and browsers while your application is executing. You can review the details of the stack trace for each exception via the Azure portal. You can also view statistics about exceptions that arise during your application’s execution.
* **Page views and load performance** Measuring the performance of your server’s page delivery is only part of the equation. Using Application Insights, you can also get information about the page views and load performance reported from the browser’s side.
* **AJAX calls** This measures the time taken by AJAX calls made from your application’s web pages. It also measures the failure rates and response time.
* **User and session counts** You can keep track of the number of users who are connected to your application. Just as the same user can initiate multiple sessions, you can track the number of sessions connected to your application. This allows you to clearly measure the threshold of concurrent users supported by your application.
* **Performance counters** You can get information about the performance counters of the server machine (CPU, memory, and network usage) from which your code is executing.
* **Hosts diagnostics** Hosts diagnostics can get information from your application if it is deployed in a Docker or Azure environment.
* **Diagnostic trace logs** Trace log messages can be used to correlate trace events with the requests made to the application by your users.
* **Custom events and metrics** Although the out-of-the-box instrumentation offered by Application Insights offers much information, some metrics are too specific to your application to be generalized and included in the general telemetry. For those cases, you can create custom metrics to monitor your server and client code. This allows you to monitor user actions, such as shopping cart checkouts or game scoring.

Application Insights are not limited to .NET languages. There are instrumentation libraries available for other languages, such as Java, JavaScript, or Node.js. There are also libraries available for other platforms like Android or iOS. You can use the following procedure to add Application Insight instrumentation to your ASP.NET application. To run this example, you need to meet these prerequisites:

* An Azure Subscription.
* Visual Studio 2017/2019. If you don’t have Visual Studio, you can download the Community edition for free from [*https://visualstudio.microsoft.com/free-developer-offers/*](https://visualstudio.microsoft.com/free-developer-offers/).
* Install the following workloads in Visual Studio:
  + ASP.NET and web development, including the optional components.
  + Azure development.

In this example, you are going to create a new MVC application from a template and then add the Application Insights instrumentation. You can use the same procedure to add instrumentation to any of your existing ASP.NET applications:

1. Open Visual Studio 2019.
2. In the home window in Visual Studio, click the Create A New Project button in the section Get Started on the right side of the window.
3. In the Create A New Project window, in the Search box, type **MVC**.
4. Select the ASP.NET Web Application (.NET Framework) template.
5. Click the Next button in the bottom-right corner of the window.
6. Type a name for your project and solution in the Project Name and Solution Name boxes, respectively.
7. Select the Location where your project will be stored.
8. Click the Create button at the bottom-right corner of the window.
9. On the Create A New ASP.NET Web Application window, select the MVC template.
10. Click the Create button at the bottom-right corner of the window.
11. In the Solution Explorer window, right-click the name of your project.
12. In the contextual menu, shown in [Figure 4-7](https://learning.oreilly.com/library/view/exam-ref-az-204/9780136798255/ch04.xhtml#ch04fig07), select Add > Application Insights Telemetry.



**Figure 4-7** Adding Application Insights Telemetry

1. On the Application Insights Configuration page, click the Get Started button at the bottom of the page.
2. On the Register Your App With Application Insights page, ensure that the correct Azure Account and Azure Subscription are selected in the drop-down menus.
3. Click the Configure Settings link below the Resource drop-down menu.
4. In the Application Insights Configuration dialog box, select the Resource Group and Location where you want to create the new Application Insight resource.
5. Click the Register button.
6. On the Application Insights Configuration tab, click the Collect Traces From System.Diagnostics button at the bottom of the tab. Enabling this option allows you to send a log message directly to Application Insights.

At this point, Visual Studio starts adding the needed packages and dependencies to your project. Visual Studio also automatically configures the Instrumentation Key, which allows your application to connect to the Application Insights resource created in Azure. Now your project is connected with the instance of the Application Insights deployed in Azure. As soon as you run your project, the Application Insights instrumentation starts sending information to Azure. You can review this information in the Azure portal or your Visual Studio. Use the following steps to access Application Insights from Visual Studio and Azure portal:

1. From the Visual Studio window, in the Solution Explorer window, navigate to your project’s name and select Connected Services > Application Insights.
2. Right-click Application Insights.
3. On the contextual menu, click Search Live Telemetry. The Application Insights Search tab appears in Visual Studio.
4. In the Solution Explorer, right-click Application Insights to open the Azure Portal Application Insights from Visual Studio.
5. On the contextual menu, click Open Application Insights Portal.

Apart from the standard metrics that come out of the box with the default Application Insights instrumentation, you can also add your custom events and metrics to your code. Using custom events and metrics, you can analyze and troubleshoot logic and workflows that are specific to your application. The following example shows how to modify the MVC application that you created on the previous example for adding custom events and metrics:

1. Open the project that you created in the previous example.
2. Open the HomeController.cs file.
3. Add the following using statements at the beginning of the file:

[Click here to view code image](https://learning.oreilly.com/library/view/exam-ref-az-204/9780136798255/ch04_images.xhtml#pg223a)

using Microsoft.ApplicationInsights;

using System.Diagnostics;

1. Replace the content of the HomeController class in the HomeController.cs file with the content in [Listing 4-3](https://learning.oreilly.com/library/view/exam-ref-az-204/9780136798255/ch04.xhtml#ch04ex03).

**Listing 4-3** HomeController class

[Click here to view code image](https://learning.oreilly.com/library/view/exam-ref-az-204/9780136798255/ch04_images.xhtml#lis4-3a)

**// C#. ASP.NET.**

public class HomeController : Controller

{

private TelemetryClient telemetry;

private double indexLoadCounter;

public HomeController()

{

//Create a TelemetryClient that can be used during the life of the

// Controller.

telemetry = new TelemetryClient();

//Initialize some counters for the custom metrics.

//This is a fake metric just for demo purposes.

indexLoadCounter = new Random().Next(1000);

}

public ActionResult Index()

{

//This example is trivial as ApplicationInsights already registered the

// load of the page.

//You can use this example for tracking different events in the

// application.

telemetry.TrackEvent("Loading the Index page");

//Before you can submit a custom metric, you need to use the GetMetric

//method.

telemetry.GetMetric("CountOfIndexPageLoads").TrackValue(indexLoadCounter);

//This trivial example shows how to track exceptions using Application

//Insights.

//You can also send trace message to Application Insights.

try

{

Trace.TraceInformation("Raising a trivial exception");

throw new System.Exception(@"Trivial Exception for testing Tracking

Exception feature in Application Insights");

}

catch (System.Exception ex)

{

Trace.TraceError("Capturing and managing the trivial exception");

telemetry.TrackException(ex);

}

//You need to instruct the TelemetryClient to send all in-memory data to

// the ApplicationInsights.

telemetry.Flush();

return View();

}

public ActionResult About()

{

ViewBag.Message = "Your application description page.";

//This example is trivial as ApplicationInsights already registers the

//load of the page.

//You can use this example for tracking different events in the

// application.

telemetry.TrackEvent("Loading the About page");

return View();

}

public ActionResult Contact()

{

ViewBag.Message = "Your contact page.";

//This example is trivial as ApplicationInsights already registers the load

//of the page.

//You can use this example for tracking different events in the

// application.

telemetry.TrackEvent("Loading the Contact page");

return View();

}

}

1. In the Solution Explorer, open the ApplicationInsights.config file.
2. In the <Add Type="Microsoft.ApplicationInsights.Extensibility.PerfCounterCollector.PerformanceCollectorModule, Microsoft.AI.PerfCounterCollector"> XML item, add the following child XML item:

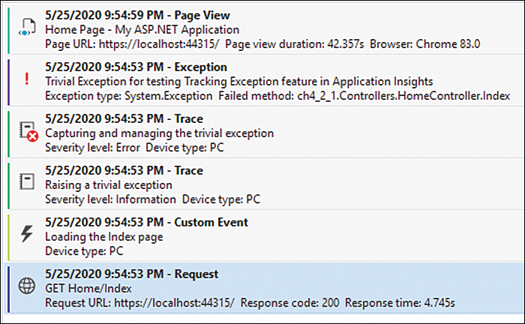
[Click here to view code image](https://learning.oreilly.com/library/view/exam-ref-az-204/9780136798255/ch04_images.xhtml#pg225a)

<EnableIISExpressPerformanceCounters>true</EnableIISExpressPerformanceCounters>

***NOTE* CONTROLLERS CONSTRUCTORS**

In the previous example, we used a private property in the constructor for creating and initializing a TelemetryClient object. In a real-world application, you should use dependency injection techniques for properly initializing the Controller class. There are several frameworks, like Unity, Autofac, or Ninject, that can help you in implementing the dependency injection pattern in your code .

At this point, you can press F5 and run your project to see how your application is sending information to Application Insights. If you review the Application Insights Search tab, you can see the messages, shown in [Figure 4-8](https://learning.oreilly.com/library/view/exam-ref-az-204/9780136798255/ch04.xhtml#ch04fig08), that your application is sending to Application Insights.

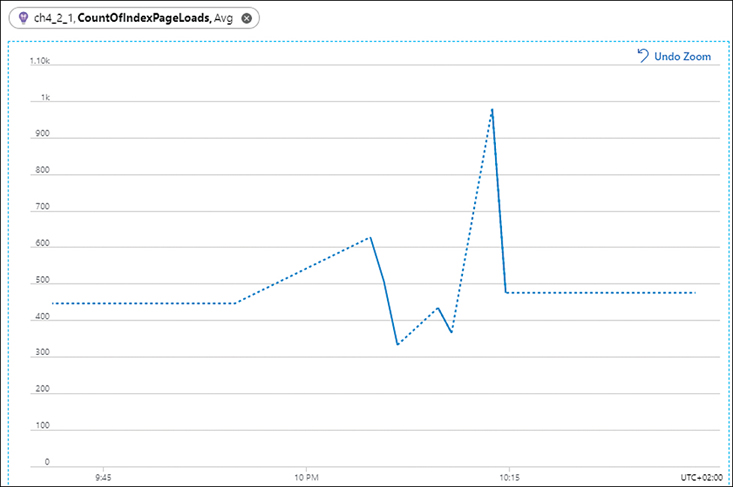


**Figure 4-8** Application Insights messages

You send messages to Application Insights by using the TelemetryClass class. This class provides you with the appropriate methods for sending the different types of messages to Application Insights. You can send custom events by using the TrackEvent() method. You use this method for tracking meaningful events to your application, such as when the user creates a new shopping cart in an eCommerce web application or the user wins a game in a mobile app.

If you need to keep track of the value of certain variables or properties in your code, you can use the combination of GetMetric() and TrackValue() methods. The GetMetric() method retrieves a metric from the azure.applicationinsight namespace. If the metric doesn’t exist on the namespace, the Application Insights library automatically creates a new one. Once you have a reference to the correct metric, you can use the TrackValue() method to add a value to that metric. You can use these custom metrics for setting alerts or autoscale rules. Use the following steps for viewing the custom metrics in the Azure portal:

1. From the Visual Studio window, in the Solution Explorer window, navigate to your project’s name and select Connected Services > Application Insights.
2. Right-click Application Insights.
3. In the contextual menu, click Open Application Insights Portal.
4. On the Application Insights blade, click Metrics in the Monitoring section of the navigation menu on the left side of the blade.
5. On the Metrics blade, on the toolbar above the empty graph, on the Metric Namespace drop-down menu, select azure.applicationsight.
6. On the Metric drop-down menu, select CountOfIndexPageLoad. This is the custom metric that you defined in the previous example.
7. On the Aggregation drop-down menu, select Count. The values for your graph will be different but should look similar to [Figure 4-9](https://learning.oreilly.com/library/view/exam-ref-az-204/9780136798255/ch04.xhtml#ch04fig09).



**Figure 4-9** Custom metric graph

You can also send log messages to Application Insights by using the integration between System.Diagnostics and Application Insights. Any message sent to the diagnostics system using the Trace class appears in Application Insights as a Trace message. In this same line, use the TraceException() method for sending the stack trace and the exception to Application Insights. The advantage of doing this is that you can easily correlate exceptions with the operations that were performing your code when the exception happened.

**Images *EXAM TIP***

Remember that Application Insights is a solution for monitoring the behavior of an application on different platforms, written in different languages. You can use Application Insights with web applications and native applications or mobile applications written in .NET, Java, JavaScript, or Node.js. There is no requirement to run your application in Azure. You only need to use Azure for deploying the Application Insights resource that you use for analyzing the information sent by your application.

***NEED MORE REVIEW?* CREATING CUSTOM EVENTS AND METRICS**

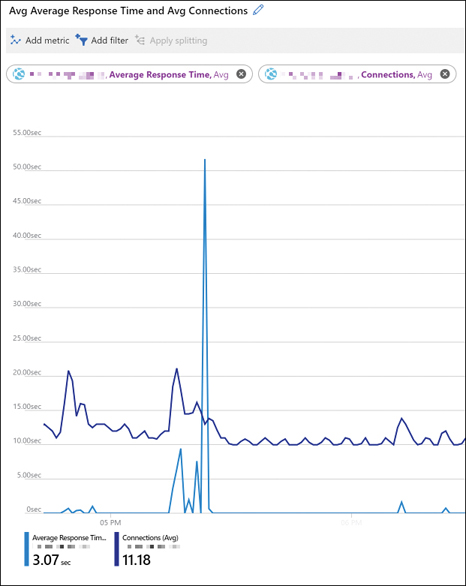
You can create more complex metrics and events than the one that we reviewed here. For complex operations, you can track all the actions inside an operation for correctly correlating all the messages generated during the execution of the operation. You can learn more about how to create custom events and metrics by reading the article at [*https://docs.microsoft.com/en-us/azure/azure-monitor/app/api-custom-events-metrics*](https://docs.microsoft.com/en-us/azure/azure-monitor/app/api-custom-events-metrics).

**Analyze log data and troubleshoot solutions by using Azure Monitor**

Azure Monitor is a tool composed of several elements that help you monitor and better understand the behavior of your solutions. Application Insights is a tool for collecting information from your solutions. Once you have the collected information, you can use the Analyze tools for reviewing the data and troubleshooting your application. Depending on the information that you need to analyze, you can use Metric Analytics or Log Analytics.

You can use Metric Analytics for reviewing the standard and custom metrics sent from your application. A metric is a numeric value that is related to some aspect at a particular point in time of your solution. CPU usage, free memory, and the number of requests are all examples of metrics; also, you can create your own custom metrics. Because metrics are lightweight, you can use them to monitor scenarios in near real-time. You analyze metric data by representing the values of the metrics in a time interval using different types of graphs. Use the following steps for reviewing graphs:

1. Open the Azure portal ([*https://portal.azure.com*](https://portal.azure.com/)).
2. On the Search Resources, Services, And Docs text box on the top side of the Azure portal, type monitor.
3. Click Monitor in the Services section in the result list.
4. On the Monitor blade, click Metrics on the navigation menu on the left side of the blade.
5. On the Metrics blade, the Select A Scope panel should appear automatically.
6. On the Select A Scope panel, in the scope tree, select the subscription or resource groups that contain the Azure App Service containing the metrics you want to add to the graph.
7. In the Resource Type drop-down menu, below the scope tree, select only the App Services resource type.
8. In the App Service drop-down, select one of your App Services.
9. Click the Apply button at the bottom of the panel.
10. On the Metrics blade, select the Average Response Time metric in the Metric drop-down menu.
11. Click the Add Metric button at the top of the graph. You can add several metrics to the same graph, which means you can analyze different metrics that are related between them.
12. Repeat step 10 for adding the Connections metric. [Figure 4-10](https://learning.oreilly.com/library/view/exam-ref-az-204/9780136798255/ch04.xhtml#ch04fig10) shows the metrics added to the graph.

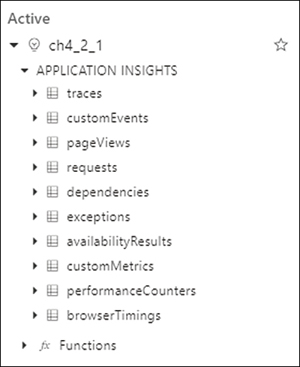


**Figure 4-10** Configuring metrics for a graph

You use Log Analytics for analyzing the trace, logs, events, exceptions, and any other message sent from your application. Log messages are more complex than metrics because they can contain much more information than a simple numeric value. You can analyze log messages by using queries for retrieving, consolidating, and analyzing the collected data. Log Analytics for Azure Monitor uses a version of the Kusto query language. You can construct your queries to get information from the data stored in Azure Monitor. To do so, complete the following steps:

1. Open the Azure portal ([*https://portal.azure.com*](https://portal.azure.com/)).
2. In the Search Resources, Services, And Docs text box at the top of the Azure portal, type **monitor**.
3. Click Monitor in the Services section in the result list.
4. On the Monitor blade, click Logs in the navigation menu on the left side of the blade.
5. On the Logs blade, click the Get Started button.
6. On the Logs blade, the Select A Scope panel should appear automatically.
7. On the Select A Scope panel, in the scope tree, navigate to the resources containing the logs you want to query. Click the check box next to the resource. You can select only resources of the same type. For this example, the resource type should be Application Insights.
8. Click the Apply button at the bottom of the panel.
9. On the Logs blade, type **traces** in the text area.
10. Click the Run button.
11. You can review the result of your query in the section below the query text area.

This simple query returns all the traces error events stored in your Application Insights workspace. You can use more complex queries to get more information about your solution. The available fields for the queries depend on the data loaded in the workspace. The data schema manages these fields. [Figure 4-11](https://learning.oreilly.com/library/view/exam-ref-az-204/9780136798255/ch04.xhtml#ch04fig11) shows the schema associated with a workplace that stores data from Application Insights.



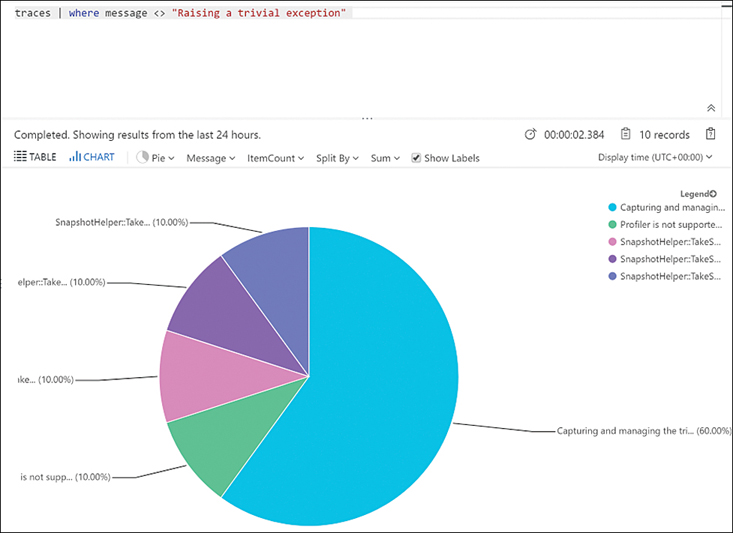
**Figure 4-11** Workspace schema

Once you get the results from a query, you can easily refine the results of the query by adding where clauses to the query. The easiest way to add new filtering criteria is to expand one of the records in the table view in the results section below the query text area. If you move your mouse over each of the fields in a record, you can see three small dots before the field of the record. If you click the three dots icon, a contextual menu appears for including or excluding the value of the field in the where clause. Based on the example in the previous section, the following query would get all traces sent from the application except those with the message Raising a trivial exception.

[Click here to view code image](https://learning.oreilly.com/library/view/exam-ref-az-204/9780136798255/ch04_images.xhtml#pg229a)

traces | where message <> "Raising a trivial exception"

You can review the results of this query in both table and chart formats. Using the different visualization formats, you can get a different insight into the data. [Figure 4-12](https://learning.oreilly.com/library/view/exam-ref-az-204/9780136798255/ch04.xhtml#ch04fig12) shows how the results from the previous query are plotted into a pie chart.



**Figure 4-12** Rendering query results

***NEED MORE REVIEW?* CREATING LOG QUERIES**

Creating the appropriate query for your needs greatly depends on the details of your solution. You can review the details about the Kusto query language and how to create complex queries by reviewing the following articles:

* Kusto Query Language: [*https://docs.microsoft.com/en-us/azure/kusto/query/*](https://docs.microsoft.com/en-us/azure/kusto/query/)
* Azure Monitor log queries: [*https://docs.microsoft.com/en-us/azure/azure-monitor/log-query/query-language*](https://docs.microsoft.com/en-us/azure/azure-monitor/log-query/query-language)

**Images *EXAM TIP***

When you try to query logs from the Azure Monitor, remember that you need to enable the diagnostics logs for the Azure App Services. If you get the message, *We didn’t find any logs* when you try to query the logs for your Azure App Service, that could mean that you need to configure the diagnostic settings in your App Service.

**Implement Application Insights Web Test and Alerts**

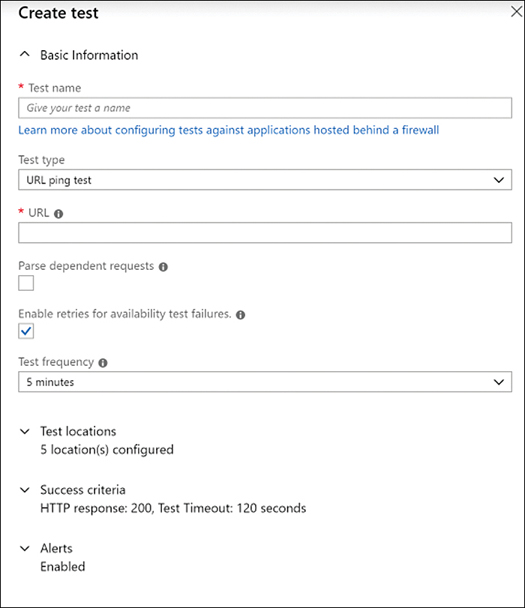
As a result of analyzing the data sent from your application to the Azure Monitor using Application Insights, you may find some situations that you need to monitor more carefully. Using Azure Monitor, you can set alerts based on the value of different metrics or logs. For example, you can create an alert to receive a notification when your application generates an HTTP return code 502.

You can also configure Application Insights for monitoring the availability of your web application. You can configure different types of tests for checking the availability of your web application:

* **URL ping test** This is a simple test for checking whether your application is available by making a request to a single URL for your application.
* **Multi-step web test** Using Visual Studio Enterprise, you can record the steps that you want to use as the verification for your application. You use this type of test for checking complex scenarios. The process of recording the steps in a web application generates a file with the recorded steps. Using this generated file, you can create a web test in Application Insights; then you upload the recording file.
* **Custom Track Availability Test** You can create your own availability test in your code using the TrackAvailability() method.

When creating a URL ping test, you can check not only the HTTP response code but also the content returned by the server. This way, you can minimize the possibility of false positives. These false positives can happen if the server returns a valid HTTP response code, but the content is different due to configuration errors. Use the following procedure for creating an URL ping test on your Application Insights that checks the availability of your web application:

1. Open the Azure portal ([*https://portal.azure.com*](https://portal.azure.com/)).
2. In the Search Resources, Services, And Docs text box at the top of the Azure portal, type **monitor**.
3. Click Monitor in the Services section in the result list.
4. On the Monitor blade, click Applications in the Insights section.
5. On the Applications blade, click the name of the Application Insights resource where you want to configure the alert.
6. On the Applications Insights blade, click Availability in the Investigate section of the navigation menu on the left side of the blade.
7. On the Availability blade, click Add Test at the top left of the blade.
8. On the Create Test blade, shown in [Figure 4-13](https://learning.oreilly.com/library/view/exam-ref-az-204/9780136798255/ch04.xhtml#ch04fig13), type a name for the test in the Test Name text box.



**Figure 4-13** Creating a URL test

1. Ensure that URL Ping Test is selected in the Test Type drop-down menu.
2. In the URL text box, type the URL of the application you want to test.
3. Expand the Test Location section. Select the locations from which you want to perform the URL ping test.
4. Leave the other options as they are.
5. Click the Create button at the bottom of the panel.

When you configure the URL ping test, you cannot configure the Alert directly during the creation process. You need to finish the creation of the test, and then you can edit the Alert for defining the actions that you want to perform when the alert fires. Use the following procedure for configuring an alert associated with the URL ping test that you configured previously:

1. On the Availability blade, click the ellipsis beside the newly created alert.
2. In the contextual menu, click Open Rules (Alerts) Page.
3. On your alert Rules Management blade, in the Condition section, ensure that there is a default condition with the name *whenever the average failed locations is greater than or equal to 2 count*.
4. On the Action Group section, click the Select Action Group link.
5. On the Configured Actions panel, click the Create Action Group button.
6. On the Select An Action Group To Attach To This Alert Rule panel, click Create Action Group.
7. On the Create Action Group panel, select a resource group to save this action group. Alternatively, you can create a new resource group by clicking the Create New link below the Resource Group drop-down menu.
8. Type a name in the Action Group Name text box. This name needs to be unique in the resource group that you selected in the previous step.
9. Click the Next: Notifications button at the bottom of the panel.
10. In the Notifications section, in the Notification Type drop-down menu, select Email/SMS Message/Push/Voice.
11. On the Email/SMS/Push/Voice panel, select the Email check box.
12. Type an email address in the text box below the Email check box.
13. Click the OK button at the bottom of the panel.
14. Type a name in the Name text box, next to the Notification Type drop-down menu.
15. Click the Next: Actions at the bottom of the panel.
16. Leave the Actions section as is. You can use this section for configuring actions like calling an Azure Function, creating a ticket in an ITSM system, or start an Azure Automation Runbook.
17. Click the Review & Create button.
18. Click the Create button.
19. On your alert Rules Management blade, ensure that the newly created Action Group has been correctly added to the list of Action Groups attached to the alert.
20. Click the Save button on the top-left corner of your alert Rules Management blade.

Now you can test whether the URL ping test is working correctly by temporarily shutting down your testing application. After five minutes, you should receive an email message at the email address you configured in the alert action associated with the URL ping test.

**Images *EXAM TIP***

Remember that you need a Visual Studio Enterprise license for creating multistep web tests. You use the Visual Studio Enterprise for the definition of the steps that are part of the test, and then you upload the test definition to Azure Application Insights.

***NEED MORE REVIEW?* AZURE MONITOR ALERTS**

Apart from creating alerts when a web test fails, you can also create alerts based on other conditions that depend on the events information stored in the Application Insights. You can review the details about how to create these alerts by reviewing the article at [*https://docs.microsoft.com/en-us/azure/azure-monitor/platform/alerts-log*](https://docs.microsoft.com/en-us/azure/azure-monitor/platform/alerts-log).

**Implement code that handles transient faults**

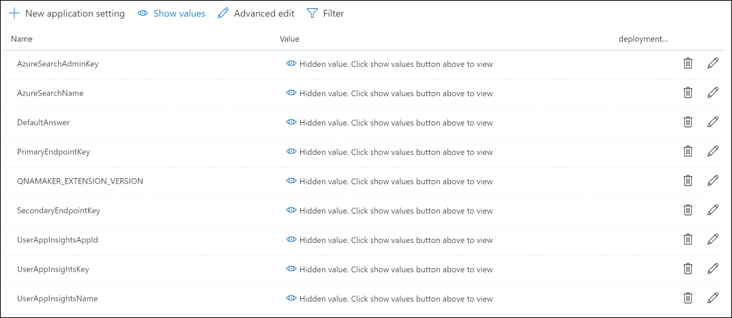
Developing applications for the cloud means that your application depends on the resources in the cloud to run your code. As we already reviewed in previous chapters, these resources provide out-of-the-box high availability and fault-tolerant features that make your application more resilient. Azure Cloud Services use redundant hardware and load balancers. Although you are guaranteed not to suffer big breakdowns, there can be situations that can temporarily affect your application, such as performing automatic failovers or load balancing operations. Usually, recovery from that kind of transient situation is as simple as retrying the operation your application was performing. For example, if your application was reading a record from a database and you get a timeout error because of a temporary overload of the database, you can retry the read operation to get the needed information.

Dealing with these transient faults leads you to deal with some interesting challenges. Your application needs to respond to these challenges to ensure that it offers a reliable experience to your users. These challenges are

* **Detect and classify faults** Not all the faults that may happen during the application execution are transient. Your application needs to identify whether the fault is transient, long-lasting, or a terminal failure. Even the term “long-lasting failure” is dependent on the logic of your application because the amount of time that you consider “long-lasting” depends on the type of operations your application performs. Your application also needs to deal with the different responses that come from different services types. An error occurring while reading data from a storage system is different from an error occurring while writing data.
* **Retry the operation when appropriate** Once your application determines that it’s dealing with a transient fault, the application needs to retry the operation. It also needs to keep track of the number of retries of the faulting operation.
* **Implement an appropriate retry strategy** Indefinitely retrying the operation could lead to other problems, such as performance degradation or blocking the resources that your application is using. To avoid those performance problems, your application needs to set a retry strategy that defines the number of retries, sets the delay between each retry, and sets the actions that your application should take after a failed attempt. Setting the correct number of retries and the delay between them is a complex task that depends on factors such as the type of resources, the operating conditions, and the application itself.

You can use the following guidelines when implementing a suitable transient fault mechanism in your application:

* **Use existing built-in retry mechanism** When working with SDKs for specific services, the SDK usually provides a built-in retry mechanism. Before thinking of implementing your retry mechanism, you should review the SDK that you are using to access the services on which your application depends and use the built-in retry mechanism. These built-in retry mechanisms are tailored to the specific features and requirements of the target service. If you still need to implement your retry mechanism for a service—such as a storage service or a service bus—you should carefully review the requirements of each service to ensure that you correctly manage the faulting responses.
* **Determine whether the operation is suitable for retrying** When an error is raised, it usually indicates the nature of the error. You can use this information to determine whether the error is a transient fault. Once you determine your application is dealing with a transient fault, you need to determine whether retrying the operation can succeed. You should not retry operations that indicate an invalid operation, such as a service that suffered a fatal error or continuing to look for an item after receiving an error indicating the item does not exist in the database. You should implement operation retries if the following conditions are met:
  + You can determine the full effect of the operation.
  + You fully understand the conditions of the retry.
  + You can validate these conditions.
* **Use the appropriate retry count and interval** Setting the wrong retry count could lead your application to fail or could lock resources that can affect the health of the application. If you set the retry count too low, your application may not have enough time to recover from the transient fault and will fail. If you set the retry count to a value that is too high or too short, you can lock resources that your application is using, such as threads, connections, or memory. This high-resource consumption can affect the health of your application. When choosing the appropriate retry count and interval, you need to consider the type of operation that suffered the transient fault. For example, if the transient fault happens during an operation that is part of user interaction, you should use a short retry interval and count, which avoids having your user wait too long for your application to recover from the transient fault. On the other hand, if the fault happens during an operation that is part of a critical workflow, setting a longer retry count and interval makes sense if restarting the workflow is time-consuming or expensive. Following are some of the most common strategies for choosing the retry interval:
  + **Exponential back-off** You use a short time interval for the first retry, and then you exponentially increase the interval time for subsequent retries. For example, you set the initial interval to 3 seconds and then use 9, 27, 81 for the subsequent retries.
  + **Incremental intervals** You set a short time interval for the first retry, then you incrementally increase the interval time for the subsequent retries. For example, you set the initial interval to 3 seconds and then use 5, 8, 13, 21 for the subsequent retries.
  + **Regular intervals** You use the same time interval for each retry. This strategy is not appropriate in most cases. You should avoid using this strategy when accessing services or resources in Azure. In those cases, you should use the exponential back-off strategy with a circuit breaker pattern.
  + **Immediate retry** You retry as soon as the transient fault happens. You should not use this type of retry more than once. The immediate retries are suitable for peak faults, such as network package collisions or spikes in hardware components. If the immediate retry doesn’t recover from the transient fault, you should switch to another retry strategy.
  + **Randomization** If your application executes several retries in parallel—regardless of the retry strategy—using the same retry values for all the retries can negatively affect your application. In general, you should use random starting retry interval values with any of the previous strategies. This allows you to minimize the probability that two different application threads start the retry mechanism at the same time in the event of a transient fault.
* **Avoid anti-patterns** When implementing your retry mechanism, there are some patterns you should avoid:
  + Avoid implementing duplicated layers of retries. If your operation is made of several requests to several services, you should avoid implementing retries on every stage of the operation.
  + Never implement endless retry mechanisms. If your application never stops retrying in the event of a transient fault, the application can cause resource exhaustion or connection throttling. You should use the circuit breaker pattern or a finite number of retries.
  + Never use immediate retry more than once.
* **Test the retry strategy and implementation** Because of the difficulties when selecting the correct retry count and interval values, you should thoroughly test your retry strategy and implementation. You should pay special attention to heavy load and high-concurrency scenarios. You should test this by injecting transient and nontransient faults into your application.
* **Manage retry policy configuration** When you are implementing your retry mechanism, you should not hardcode the values for the retry count and intervals. Instead, you can define a retry policy that contains the retry count and interval as well as the mechanism that determines whether a fault is transient or nontransient. You should store this retry policy in configuration files so that you can fine-tune the policy. You should also implement this retry policy configuration so that your application stores the values in memory instead of continuously rereading the configuration file. If you are using Azure App Service, you should consider using the service configuration shown in [Figure 4-14](https://learning.oreilly.com/library/view/exam-ref-az-204/9780136798255/ch04.xhtml#ch04fig14).



**Figure 4-14** Azure App Service application settings

* **Log transient and non-transient faults** You should include a logging mechanism in your application every time a transient or nontransient fault happens. A single transient fault doesn’t indicate an error in your application. If the number of transient faults is increasing, this can be an indicator of a more significant potential failure or that you should increase the resources assigned to the faulting service. You should log transient faults as warning messages instead of errors. Using the Error Log Level could lead to triggering false alerts in your monitoring system. You should also consider measuring and logging the overall time taken by your retry mechanism when recovering a faulty operation. This allows you to measure the overall impact of transient faults on user response times, process latency, and efficiency of the application.

***NEED MORE REVIEW?* MANAGING TRANSIENT FAULTS**

You can review some general guidelines for implementing a transient fault-handling mechanism by reviewing the following articles:

* [*https://docs.microsoft.com/en-us/azure/architecture/best-practices/transient-faults*](https://docs.microsoft.com/en-us/azure/architecture/best-practices/transient-faults)
* [*https://docs.microsoft.com/en-us/aspnet/aspnet/overview/developing-apps-with-windows-azure/building-real-world-cloud-apps-with-windows-azure/transient-fault-handling*](https://docs.microsoft.com/en-us/aspnet/aspnet/overview/developing-apps-with-windows-azure/building-real-world-cloud-apps-with-windows-azure/transient-fault-handling)

***NEED MORE REVIEW?* USEFUL PATTERNS**

When implementing your retry mechanism, you can use the following patterns:

* Retry pattern You can review the details and examples of how to implement the pattern by reading the article at [*https://docs.microsoft.com/en-us/azure/architecture/patterns/retry*](https://docs.microsoft.com/en-us/azure/architecture/patterns/retry).
* Circuit pattern You can review the details and examples of how to implement the pattern by reading the article at [*https://docs.microsoft.com/en-us/azure/architecture/patterns/circuit-breaker*](https://docs.microsoft.com/en-us/azure/architecture/patterns/circuit-breaker)*.*

**Images *EXAM TIP***

Remember to test your retry strategy carefully. Using a wrong retry strategy could lead your application to exhaust the resources needed for executing your code. A wrong retry strategy can potentially lead to infinite loops if you don’t use circuit breakers.

**Chapter summary**

* Your application needs to be able to manage transient faults.
* You need to determine the type of fault before retrying the operation.
* You should not use immediate retry more than once.
* You should use random starting values for the retry periods.
* You should use the built-in SDK mechanism when available.
* You should test your retry count and interval strategy.
* You should log transient and nontransient faults.
* You can improve the performance of your application by adding a cache to your application.
* Azure Cache for Redis allows the caching of dynamic content.
* Using Azure Cache for Redis, you can create in-memory databases to cache the most-used values.
* Azure Cache for Redis allows you to use messaging queue patterns.
* Content Delivery Networks (CDNs) store and distribute static content in servers distributed across the globe.
* CDNs reduce the latency by serving the content from the server nearest to the user.
* You can invalidate the content of the cache by setting a low TTL (Time-To-Live).
* You can invalidate the content of the cache by removing all or part of the content from the cache.
* Application Insights gets information from your application and sends it to Azure.
* You can use Application Insights with different platforms and languages.
* Application Insights is part of the Azure Monitor service.
* Application Insights generates two types of information: metrics and logs.
* Application Insights allows you to create web tests to monitor the availability of your application.
* You can configure alerts and trigger different actions associated with web tests.

**Thought experiment**

In this thought experiment, demonstrate your skills and knowledge of the topics covered in this chapter. You can find answers to this thought experiment in the next section.

Your company has a Line-of-Business (LOB) application that has been developed by your team. This LOB application is an eCommerce application that has more usage during holiday periods. The LOB application needs to get some information from external systems. You are receiving some complaints about the stability and the performance of the application. Answer the following questions about the troubleshooting and the performance of the application:

[**1.**](https://learning.oreilly.com/library/view/exam-ref-az-204/9780136798255/ch04.xhtml#ch04que1a) After reviewing the metrics of your application in the Azure Monitor, you find that you don’t have enough detail about the performance of the internal application workflows. What should you do to get information about the internal workflows?

[**2.**](https://learning.oreilly.com/library/view/exam-ref-az-204/9780136798255/ch04.xhtml#ch04que2a) After reviewing the metrics of your application in the Azure Monitor, you find that some of the stability issues are due to the external systems. You need to minimize the effect on the user experience. Which strategy should you use?

[**3.**](https://learning.oreilly.com/library/view/exam-ref-az-204/9780136798255/ch04.xhtml#ch04que3a) You need to ensure that the purchase process is working correctly. You decide to configure a web test in Application Insights. Which type of test should you configure?

**Thought experiment answers**

This section contains the solution to the thought experiment. Each answer explains why the answer choice is correct.

[**1.**](https://learning.oreilly.com/library/view/exam-ref-az-204/9780136798255/ch04.xhtml#ch04que1) You should integrate Application Insights instruments with your code. Once you integrate the Application Insights with your code, you can track custom events in your code. You can define operations inside your code to track complex operations compounded of several tasks. This allows you to get more information about the internal workflows executed in the application. Performing Application Insights agent-based monitoring doesn’t provide enough information.

[**2.**](https://learning.oreilly.com/library/view/exam-ref-az-204/9780136798255/ch04.xhtml#ch04que2) When you are dealing with the user experience, you should consider implementing a retry strategy consisting of a small number of retries with a short retry interval. Using this kind of strategy allows you to minimize the time that your users need to wait for your application to recover from a transient fault. You can also consider using an immediate retry as the first retry. If this first retry fails, then you should switch to another retry strategy. There is no one-fits-all strategy, so you need to test your strategy to ensure that you provide the best user experience.

[**3.**](https://learning.oreilly.com/library/view/exam-ref-az-204/9780136798255/ch04.xhtml#ch04que3) The process of a purchase in a web application is a complex testing scenario. In this scenario, you need to use a multistep web test. Using Visual Studio Enterprise, you need to record the steps needed for performing a purchase in your web application. Once you have generated the file with the recorded steps, you can create a web test in Application Insights to monitor the purchase process.