

Contents

[Azure Cache for Redis Documentation](#)

Overview

[About Azure Cache for Redis](#)

Quickstarts

[ASP.NET app](#)

[.NET Core app](#)

[.NET app](#)

[Java app](#)

[Node.js app](#)

[Python app](#)

Tutorials

[Use caching with leaderboards](#)

Samples

[Azure Cache for Redis samples](#)

[Azure PowerShell](#)

[Azure CLI](#)

Concepts

[Failover and patching](#)

[Best practices](#)

[Architecture best practices for caching](#)

How-to guides

Plan

[Choose a cache tier](#)

[Distribute your cache with clustering](#)

[Persist your cache with Redis data persistence](#)

[Secure your cache with a virtual network](#)

[Set up geo-replication](#)

[Save with reserved capacity](#)

[Migrate from Managed Cache Service](#)

Automate

Deploy and manage using Azure Powershell

Deploy and Manage using Azure CLI

Provision an Azure Cache for Redis

Provision a Web App with Azure Cache for Redis

Integrate with application

ASP.NET

Use session state provider

Use output cache provider

Java Spring

Use Spring Boot Initializer

Manage

Configure in Azure portal

Import/Export data

Reboot

Schedule updates

Configure redis-cli.exe access

Remove TLS 1.0 and 1.1

Monitor

Monitor in Azure portal

Set alerts for exceptions

Diagnose and troubleshoot

Troubleshoot Redis server

Troubleshoot Redis client

Troubleshoot timeouts

Troubleshoot data loss

Scale

Update to a different size and tier

Use Redis cluster to scale in/out

Reference

Code samples

Azure PowerShell

[Azure CLI](#)

[.NET](#)

[Java](#)

[Redis clients](#)

[Redis commands](#)

[REST](#)

[Resource Manager template](#)

Resources

[Build your skills with Microsoft Learn](#)

[Azure Roadmap](#)

[Pricing](#)

[Azure Cache for Redis FAQ](#)

[Videos](#)

[MSDN forum](#)

Azure Cache for Redis description

12/13/2019 • 4 minutes to read • [Edit Online](#)

Azure Cache for Redis is based on the popular software [Redis](#). It is typically used as a cache to improve the performance and scalability of systems that rely heavily on backend data-stores. Performance is improved by temporarily copying frequently accessed data to fast storage located close to the application. With [Azure Cache for Redis](#), this fast storage is located in-memory with Azure Cache for Redis instead of being loaded from disk by a database.

Azure Cache for Redis can also be used as an in-memory data structure store, a distributed non-relational database, and a message broker. Application performance is improved by taking advantage of the low-latency, high-throughput performance of the Redis engine.

Azure Cache for Redis provides you access to a secure, dedicated Redis cache. Azure Cache for Redis is managed by Microsoft, hosted within Azure, and accessible to any application within or outside of Azure.

Using Azure Cache for Redis

There are many common patterns where Azure Cache for Redis is used to support application architecture or to improve application performance. Some of the most common include the following:

PATTERN	DESCRIPTION
Cache-Aside	Since a database can be large, loading an entire database into a cache is not a recommended approach. It is common to use the cache-aside pattern to load data items into the cache only as needed. When the system makes changes to the backend data, it can at that time also update the cache, which is distributed with other clients. Additionally, the system can set an expiration on data items, or use an eviction policy to cause data updates to be reloaded into the cache.
Content Caching	Most web pages are generated from templates with headers, footers, toolbars, menus, etc. They don't actually change often and should not be generated dynamically. Using an in-memory cache, like Azure Cache for Redis, will give your web servers quick access to this type of static content compared to backend datastores. This pattern reduces processing time and server load that would be required to generate content dynamically. This allows web servers to be more responsive, and can allow you to reduce the number of servers needed to handle loads. Azure Cache for Redis provides the Redis Output Cache Provider to help support this pattern with ASP.NET.
User session caching	This pattern is commonly used with shopping carts and other user history type information that a web application may want to associate with user cookies. Storing too much in a cookie can have a negative impact on performance as the cookie size grows and is passed and validated with every request. A typical solution is to use the cookie as a key to query the data in a backend database. Using an in-memory cache, like Azure Cache for Redis, to associate information with a user is much faster than interacting with a full relational database.

PATTERN	DESCRIPTION
Job and message queuing	When applications receive requests, often the operations associated with the request take additional time to execute. It is a common pattern to defer longer running operations by adding them to a queue, which is processed later, and possibly by another server. This method of deferring work is called task queuing. There are many software components designed to support task queues. Azure Cache for Redis also serves this purpose well as a distributed queue.
Distributed transactions	It is a common requirement for applications to be able to execute a series of commands against a backend data-store as a single operation (atomic). All commands must succeed, or all must be rolled back to the initial state. Azure Cache for Redis supports executing a batch of commands as a single operation in the form of Transactions .

Azure Cache for Redis offerings

Azure Cache for Redis is available in the following tiers:

TIER	DESCRIPTION
Basic	A single node cache. This tier supports multiple memory sizes (250 MB - 53 GB). This is an ideal tier for development/test and non-critical workloads. The Basic tier has no service-level agreement (SLA)
Standard	A replicated cache in a two-node, primary/secondary, configuration managed by Microsoft, with a high-availability SLA (99.9%)
Premium	The Premium tier is the Enterprise-ready tier. Premium tier Caches support more features and have higher throughput with lower latencies. Caches in the Premium tier are deployed on more powerful hardware providing better performance compared to the Basic or Standard Tier. This advantage means the throughput for a cache of the same size will be higher in Premium compared to Standard tier.

TIP

For more information about size, throughput, and bandwidth with premium caches, see [Azure Cache for Redis FAQ](#).

You can scale your cache up to a higher tier after it has already been created. Scaling down to a lower tier is not supported. For step-by-step scaling instructions, see [How to Scale Azure Cache for Redis](#) and [How to automate a scaling operation](#).

Feature comparison

The [Azure Cache for Redis Pricing](#) page provides a detailed comparison of each tier. The following table helps describe some of the features supported by tier:

FEATURE DESCRIPTION	PREMIUM	STANDARD	BASIC
Service Level Agreement (SLA)	✓	✓	-
Redis data persistence	✓	-	-
Redis cluster	✓	-	-
Security via Firewall rules	✓	✓	✓
Encryption in transit	✓	✓	✓
Enhanced security and isolation with VNet	✓	-	-
Import/Export	✓	-	-
Scheduled updates	✓	✓	✓
Geo-replication	✓	-	-
Reboot	✓	✓	✓

Next steps

- [ASP.NET Web App Quickstart](#) Create a simple ASP.NET web app that uses an Azure Cache for Redis.
- [.NET Quickstart](#) Create a .NET app that uses an Azure Cache for Redis.
- [.NET Core Quickstart](#) Create a .NET Core app that uses an Azure Cache for Redis.
- [Node.js Quickstart](#) Create a simple Node.js app that uses an Azure Cache for Redis.
- [Java Quickstart](#) Create a simple Java app that uses an Azure Cache for Redis.
- [Python Quickstart](#) Create a Python app that uses an Azure Cache for Redis.

Quickstart: Use Azure Cache for Redis with an ASP.NET web app

2/6/2020 • 10 minutes to read • [Edit Online](#)

In this quickstart, you use Visual Studio 2019 to create an ASP.NET web application that connects to Azure Cache for Redis to store and retrieve data from the cache. You then deploy the app to Azure App Service.

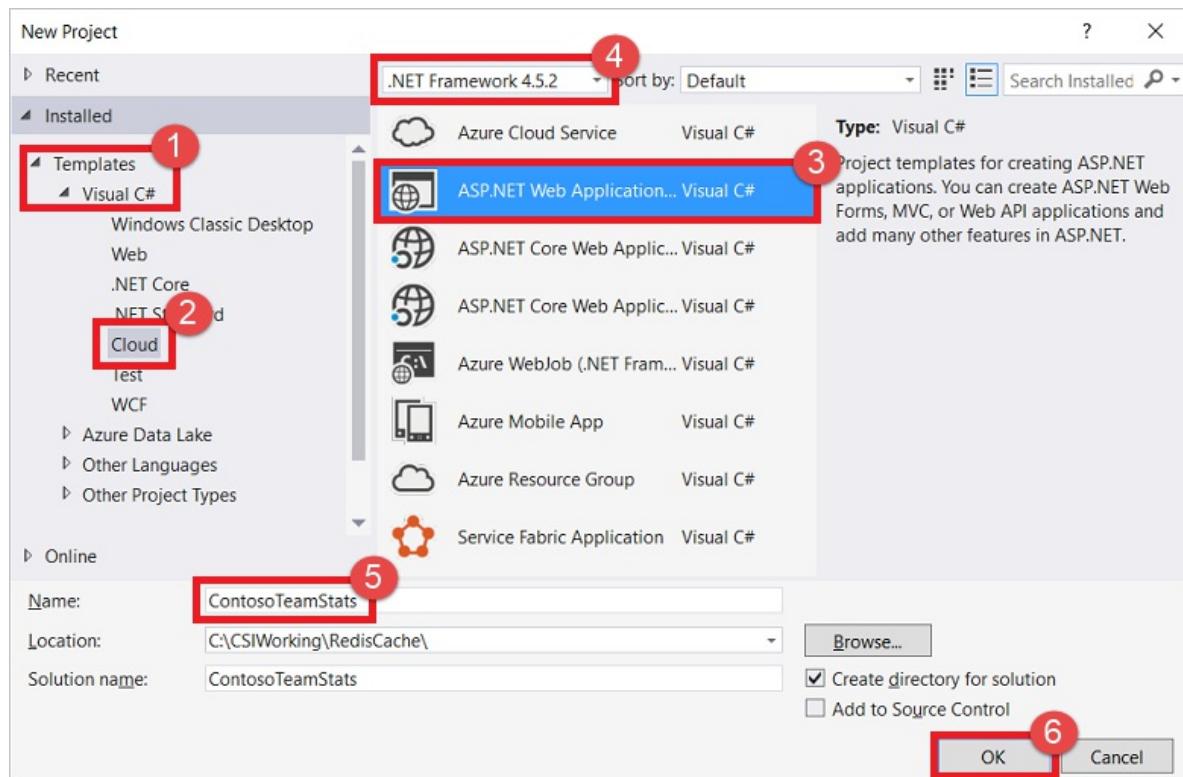
Prerequisites

- Azure subscription - [create one for free](#)
- [Visual Studio 2019](#) with the **ASP.NET and web development** and **Azure development** workloads.

Create the Visual Studio project

1. Open Visual Studio, and then select **File > New > Project**.

2. In the **New Project** dialog box, take the following steps:



- a. In the **Templates** list, expand the **Visual C#** node.
 - b. Select **Cloud**.
 - c. Select **ASP.NET Web Application**.
 - d. Verify that **.NET Framework 4.5.2** or higher is selected.
 - e. In the **Name** box, give the project a name. For this example, we used **ContosoTeamStats**.
 - f. Select **OK**.
3. Select **MVC** as the project type.

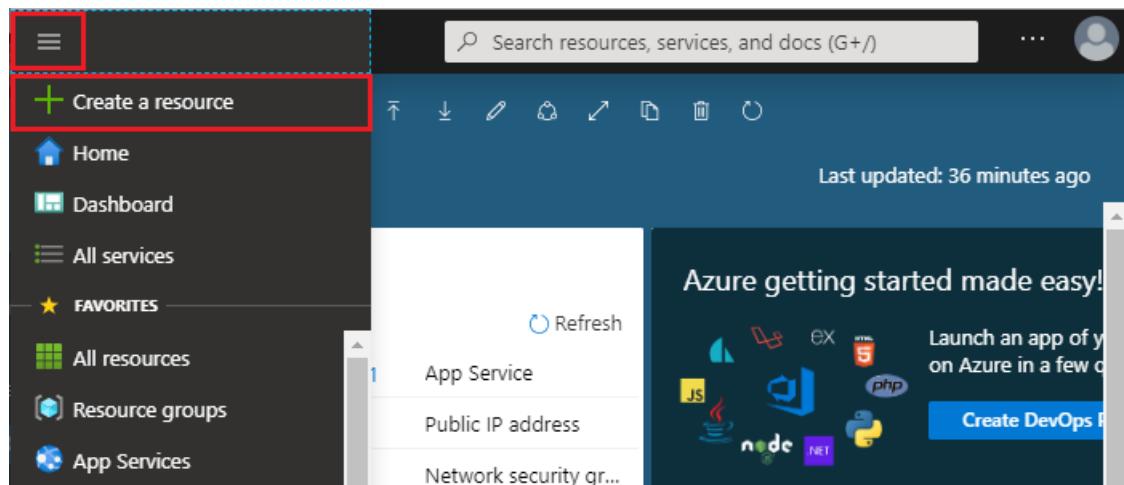
4. Make sure that **No Authentication** is specified for the **Authentication** settings. Depending on your version of Visual Studio, the default **Authentication** setting might be something else. To change it, select **Change Authentication** and then **No Authentication**.

5. Select **OK** to create the project.

Create a cache

Next, you create the cache for the app.

1. To create a cache, sign in to the [Azure portal](#) and select **Create a resource**.



2. On the **New** page, select **Databases** and then select **Azure Cache for Redis**.

New

Search the Marketplace

Azure Marketplace See all

Featured See all

Get started	 Azure SQL Managed Instance Quickstart tutorial
Recently created	 SQL Database Quickstart tutorial
AI + Machine Learning	 SQL Data Warehouse Quickstart tutorial
Analytics	 Azure Database for MariaDB Learn more
Blockchain	 Couchbase Enterprise Edition (Hourly Pricing) (preview) Learn more
Compute	 Azure Database for MySQL Quickstart tutorial
Containers	 Azure Database for PostgreSQL Quickstart tutorial
Databases	 Azure Cosmos DB Quickstart tutorial
Developer Tools	 SQL Server 2017 Enterprise Windows Server 2016 Learn more
DevOps	 Mixed Reality
Identity	 IT & Management Tools
Integration	 Networking
Internet of Things	 Software as a Service (SaaS)
Media	 Security
Mixed Reality	 Storage
IT & Management Tools	 Web
Networking	 Azure Cache for Redis Quickstart tutorial

- On the **New Redis Cache** page, configure the settings for your new cache.

SETTING	SUGGESTED VALUE	DESCRIPTION
DNS name	Enter a globally unique name.	The cache name must be a string between 1 and 63 characters that contains only numbers, letters, or hyphens. The name must start and end with a number or letter, and can't contain consecutive hyphens. Your cache instance's <i>host name</i> will be < <i>DNS name</i> >.redis.cache.windows.net.
Subscription	Drop down and select your subscription.	The subscription under which to create this new Azure Cache for Redis instance.

SETTING	SUGGESTED VALUE	DESCRIPTION
Resource group	Drop down and select a resource group, or select Create new and enter a new resource group name.	Name for the resource group in which to create your cache and other resources. By putting all your app resources in one resource group, you can easily manage or delete them together.
Location	Drop down and select a location.	Select a region near other services that will use your cache.
Pricing tier	Drop down and select a Pricing tier .	The pricing tier determines the size, performance, and features that are available for the cache. For more information, see Azure Cache for Redis Overview .

4. Select **Create**.

New Redis Cache □ X

DNS name *****
contoso-tmp ✓
.redis.cache.windows.net

Subscription *****
A Subscription

Resource group ***** ⓘ
(New) contoso-rg-tmp ▼
[Create new](#)

Location *****
Central US ▼

Pricing tier ([View full pricing details](#)) *****
Basic C0 (250 MB Cache) ▼

(PREVIEW) Availability zone
Requires Premium tier ▼

Redis Cluster ⓘ >
Requires Premium tier

Data persistence ⓘ >
Requires Premium tier

Virtual Network ⓘ >
Requires Premium tier

Unblock port 6379 (not SSL encrypted)

Create [Automation options](#)

It takes a while for the cache to create. You can monitor progress on the Azure Cache for Redis [Overview](#) page. When **Status** shows as **Running**, the cache is ready to use.

The screenshot shows the Azure portal interface for an Azure Cache for Redis instance named 'contoso-tmp'. The left sidebar lists navigation options: Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Settings, and Access keys. The 'Access keys' option is highlighted with a red box. The main content area displays the following details:

Resource group (change)	Host name
contoso-rg-tmp	contoso-tmp.redis.cache.windows.net
Status	Ports
Running - Basic 250 MB	Non-SSL port (6379) disabled
Location	Keys
Central US	Show access keys...
Subscription (change)	*Best practices*
A Subscription	https://aka.ms/redis/p/bestpractices
Subscription ID	*New features*
12345678-1234-1234-1234-123456781234	https://aka.ms/newfeatures

Retrieve host name, ports, and access keys from the Azure portal

To connect to an Azure Cache for Redis instance, cache clients need the host name, ports, and a key for the cache. Some clients might refer to these items by slightly different names. You can get the host name, ports, and keys from the [Azure portal](#).

- To get the access keys, from your cache left navigation, select **Access keys**.

The screenshot shows the 'Access keys' page for the 'contoso-tmp' cache. The left sidebar lists navigation options: Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Settings, and Access keys. The 'Access keys' option is highlighted with a red box. The main content area displays the following information:

Primary	Copy to clipboard
d2Y2ba56GvC2kEM3NmOrKuctu+emE1A1A1A1A1A1A1A1A=	
Secondary	
5MLRS9d56Lgt+61AjWU9r6KJsFwS6691A1A1A1A1A=	
Primary connection string (StackExchange.Redis)	
contoso-tmp.redis.cache.windows.net:6380,password=d2Y2ba56GvC2kEM3NmOrKuctu+emE...	
Secondary connection string (StackExchange.Redis)	
contoso-tmp.redis.cache.windows.net:6380,password=5MLRS9d56Lgt+61AjWU9r6KJsFwS669...	

For information on other clients see:
<http://aka.ms/redisclients>

- To get the host name and ports, from your cache left navigation, select **Properties**. The host name is of the form <DNS name>.redis.cache.windows.net.

The screenshot shows the Azure Cache for Redis Properties page for a cache named 'contoso-tmp'. The left sidebar lists various settings: Access keys, Advanced settings, Scale, Cluster size, Data persistence, Schedule updates, Geo-replication, Virtual Network, Firewall, and Properties. The 'Properties' item is highlighted with a red box. The main pane displays the status as 'Running', host name as 'contoso-tmp.redis.cache.windows.net' (with a 'Copy to clipboard' button), non-SSL port as 'Disabled', SSL port as '6380', and location as 'Central US'.

To edit the `CacheSecrets.config` file

1. Create a file on your computer named `CacheSecrets.config`. Put it in a location where it won't be checked in with the source code of your sample application. For this quickstart, the `CacheSecrets.config` file is located at `C:\AppSecrets\CacheSecrets.config`.
2. Edit the `CacheSecrets.config` file. Then add the following content:

```
<appSettings>
    <add key="CacheConnection" value="<cache-name>.redis.cache.windows.net,abortConnect=false,ssl=true,password=<access-key>" />
</appSettings>
```

3. Replace `<cache-name>` with your cache host name.

4. Replace `<access-key>` with the primary key for your cache.

TIP

You can use the secondary access key during key rotation as an alternate key while you regenerate the primary access key.

5. Save the file.

Update the MVC application

In this section, you update the application to support a new view that displays a simple test against Azure Cache for Redis.

- [Update the web.config file with an app setting for the cache](#)
- Configure the application to use the StackExchange.Redis client
- Update the HomeController and Layout
- Add a new RedisCache view

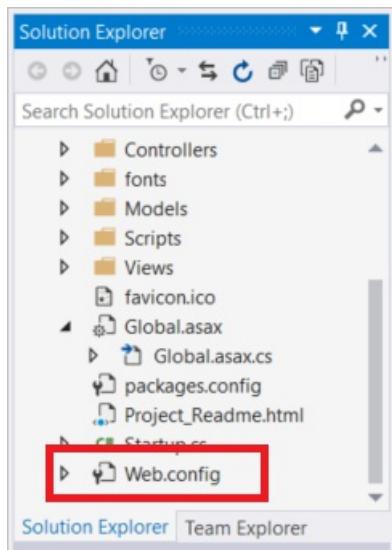
Update the web.config file with an app setting for the cache

When you run the application locally, the information in `CacheSecrets.config` is used to connect to your Azure Cache for Redis instance. Later you deploy this application to Azure. At that time, you configure an app setting in Azure that the application uses to retrieve the cache connection information instead of this file.

Because the file `CacheSecrets.config` isn't deployed to Azure with your application, you only use it while testing the application locally. Keep this information as secure as possible to prevent malicious access to your cache data.

To update the `web.config` file

1. In **Solution Explorer**, double-click the `web.config` file to open it.



2. In the `web.config` file, find the `<appSetting>` element. Then add the following `file` attribute. If you used a different file name or location, substitute those values for the ones that are shown in the example.

- Before: `<appSettings>`
- After: `<appSettings file="C:\AppSecrets\CacheSecrets.config">`

The ASP.NET runtime merges the contents of the external file with the markup in the `<appSettings>` element. The runtime ignores the file attribute if the specified file can't be found. Your secrets (the connection string to your cache) aren't included as part of the source code for the application. When you deploy your web app to Azure, the `CacheSecrets.config` file isn't deployed.

To configure the application to use StackExchange.Redis

1. To configure the app to use the `StackExchange.Redis` NuGet package for Visual Studio, select **Tools > NuGet Package Manager > Package Manager Console**.
2. Run the following command from the `Package Manager Console` window:

```
Install-Package StackExchange.Redis
```

3. The NuGet package downloads and adds the required assembly references for your client application to access Azure Cache for Redis with the `StackExchange.Azure` Cache for Redis client. If you prefer to use a strong-named version of the `StackExchange.Redis` client library, install the `StackExchange.Redis.StrongName` package.

To update the `HomeController` and Layout

1. In **Solution Explorer**, expand the **Controllers** folder, and then open the `HomeController.cs` file.
2. Add the following two `using` statements at the top of the file to support the cache client and app settings.

```
using System.Configuration;
using StackExchange.Redis;
```

3. Add the following method to the `HomeController` class to support a new `RedisCache` action that runs some commands against the new cache.

```

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);
    });

    // Connection refers to a property that returns a ConnectionMultiplexer
    // as shown in the previous example.
    IDatabase cache = lazyConnection.Value.GetDatabase();

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

    // Simple PING 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();

    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();

    // Demonstrate "SET Message" executed as expected...
    ViewBag.command4 = "GET Message";
    ViewBag.command4Result = cache.StringGet("Message").ToString();

    // Get the client list, useful to see if connection list is growing...
    ViewBag.command5 = "CLIENT LIST";
    ViewBag.command5Result = cache.Execute("CLIENT", "LIST").ToString().Replace(" id=", "\rid=");

    lazyConnection.Value.Dispose();

    return View();
}

```

4. In **Solution Explorer**, expand the **Views > Shared** folder. Then open the *_Layout.cshtml* file.

Replace:

```
@Html.ActionLink("Application name", "Index", "Home", new { area = "" }, new { @class = "navbar-brand" })
```

with:

```
@Html.ActionLink("Azure Cache for Redis Test", "RedisCache", "Home", new { area = "" }, new { @class = "navbar-brand" })
```

To add a new RedisCache view

1. In **Solution Explorer**, expand the **Views** folder, and then right-click the **Home** folder. Choose **Add > View....**
2. In the **Add View** dialog box, enter **RedisCache** for the View Name. Then select **Add**.
3. Replace the code in the *RedisCache.cshtml* file with the following code:

```

@{
    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>

```

Run the app locally

By default, the project is configured to host the app locally in [IIS Express](#) for testing and debugging.

To run the app locally

1. In Visual Studio, select **Debug > Start Debugging** to build and start the app locally for testing and debugging.
2. In the browser, select **Azure Cache for Redis Test** on the navigation bar.
3. In the following example, the `Message` key previously had a cached value, which was set by using the Azure Cache for Redis console in the portal. The app updated that cached value. The app also executed the `PING` and `CLIENT LIST` commands.

Azure Redis Cache Test.

A simple example with Azure Redis Cache on ASP.NET.

Command	Result
PING	PONG
GET Message	Hello! The cache is working from the Redis Console!
SET Message "Hello! The cache is working from ASP.NET!"	True
GET Message	Hello! The cache is working from ASP.NET!
CLIENT LIST	<pre>id=288406 addr=127.0.0.1:23426 fd=23 name=PORTAL_CONSOLE age=5 idle=5 flags=N db=0 sub=0 psub=0 multi=-1 qbuf=0 qbuf-free=0 obl=0 oll=0 omem=0 ovw=0 owmem=0 events=r cmd=set numops=4 id=288407 addr=... fd=14 name=ZION age=0 idle=0 flags=N db=0 sub=0 psub=0 multi=-1 qbuf=0 qbuf-free=3 2768 obl=0 oll=0 omem=0 ovw=0 owmem=0 events=r cmd=client numops=17 id=288408 addr=... fd=16 name=ZION age=0 idle=0 flags=N db=0 sub=1 psub=0 multi=-1 qbuf=0 qbuf-free=0 obl=0 oll=0 omem=0 ovw=0 owmem=0 events=r cmd=subscribe numops=5</pre>

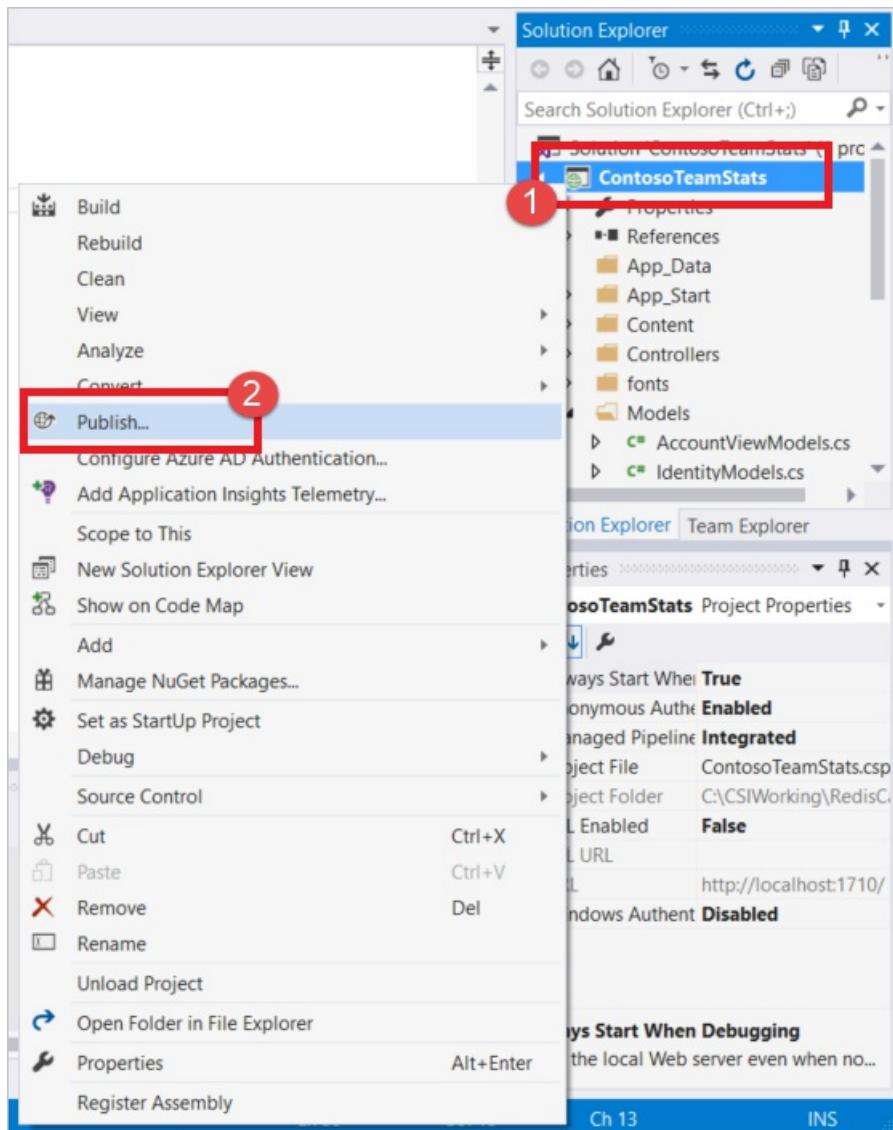
© 2018 - My ASP.NET Application

Publish and run in Azure

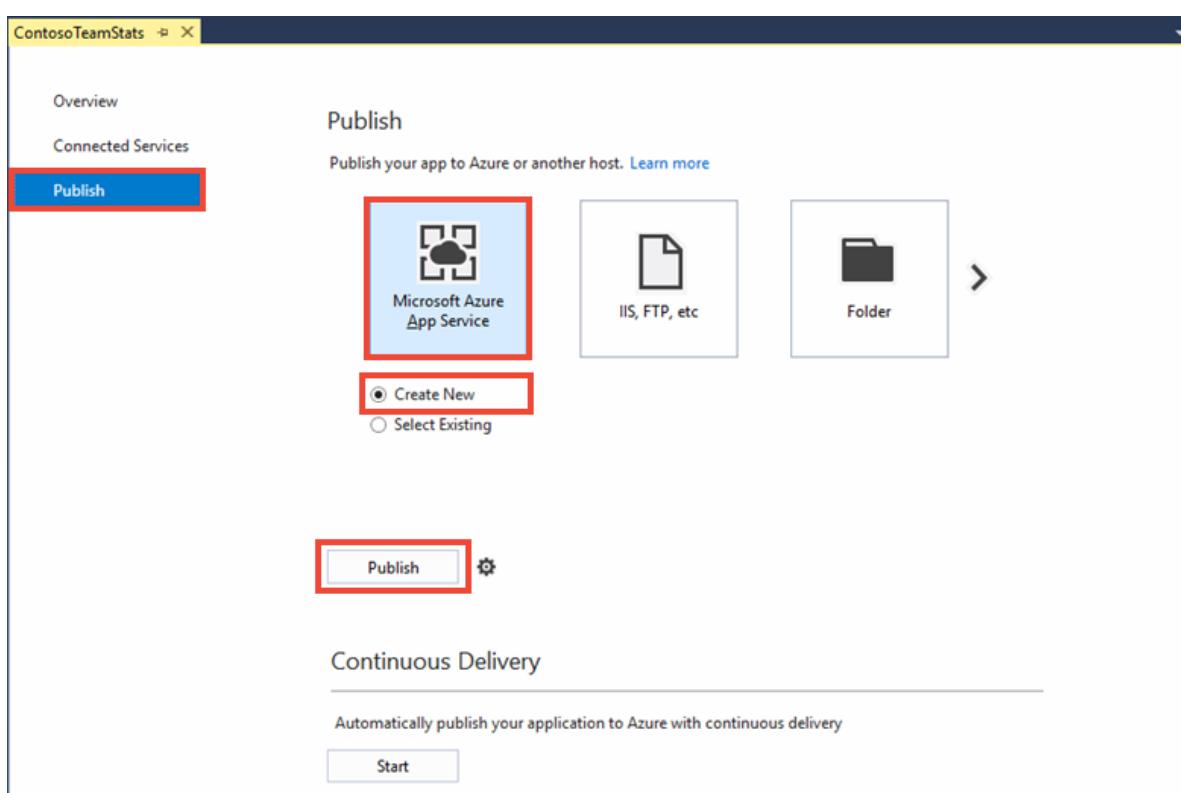
After you successfully test the app locally, you can deploy the app to Azure and run it in the cloud.

To publish the app to Azure

1. In Visual Studio, right-click the project node in Solution Explorer. Then select **Publish**.



2. Select **Microsoft Azure App Service**, select **Create New**, and then select **Publish**.



3. In the **Create App Service** dialog box, make the following changes:

SETTING	RECOMMENDED VALUE	DESCRIPTION
App name	Use the default.	The app name is the host name for the app when it's deployed to Azure. The name might have a timestamp suffix added to it to make it unique if necessary.
Subscription	Choose your Azure subscription.	This subscription is charged for any related hosting costs. If you have multiple Azure subscriptions, verify that the subscription that you want is selected.
Resource group	Use the same resource group where you created the cache (for example, <i>TestResourceGroup</i>).	The resource group helps you manage all resources as a group. Later, when you want to delete the app, you can just delete the group.
App Service plan	Select New , and then create a new App Service plan named <i>TestingPlan</i> . Use the same Location you used when creating your cache. Choose Free for the size.	An App Service plan defines a set of compute resources for a web app to run with.

Create App Service
Host your web and mobile applications, REST APIs, and more in Azure

Hosting Services

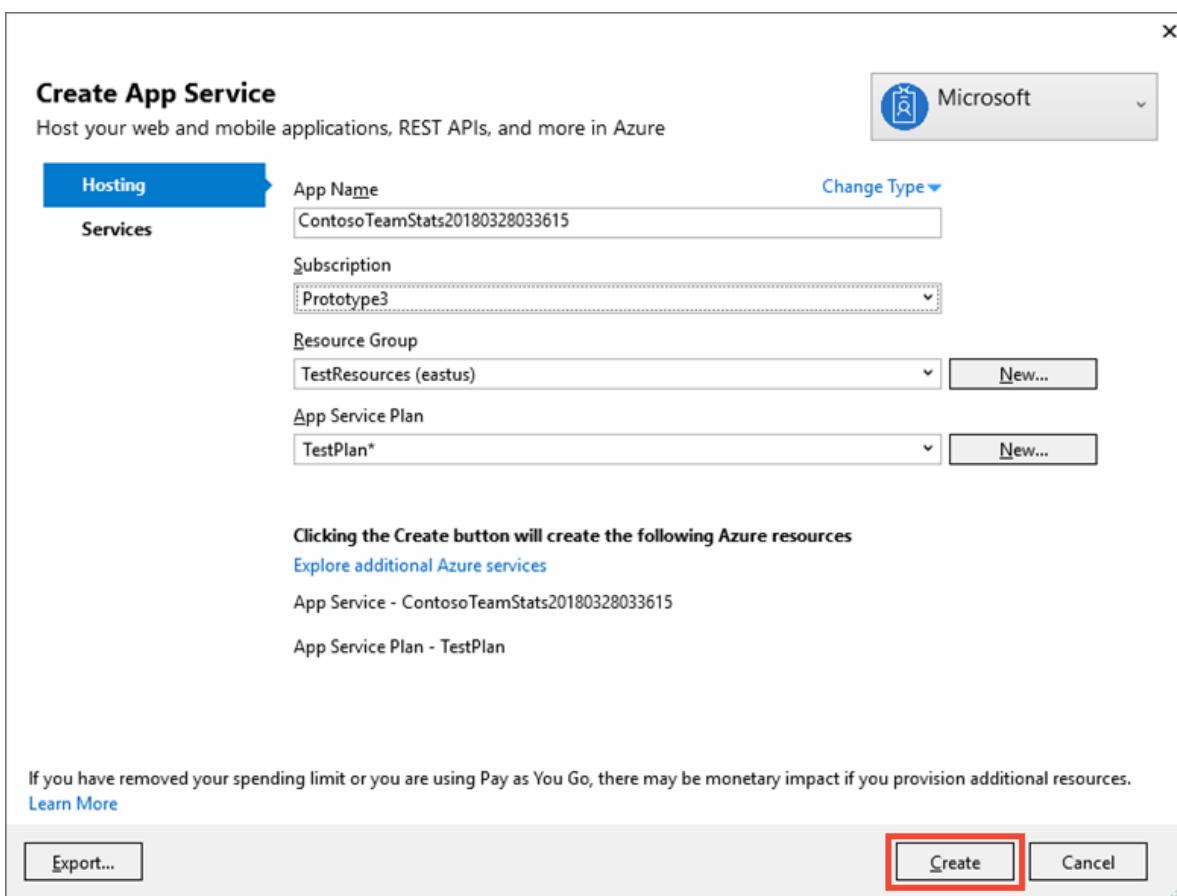
App Name	<input type="text" value="ContosoTeamStats20180328033615"/> Change Type ▾
Subscription	<input type="text" value="Prototype3"/>
Resource Group	<input type="text" value="TestResources (eastus)"/> New...
App Service Plan	<input type="text" value="TestPlan*"/> New...

Clicking the Create button will create the following Azure resources
[Explore additional Azure services](#)

App Service - ContosoTeamStats20180328033615
App Service Plan - TestPlan

If you have removed your spending limit or you are using Pay as You Go, there may be monetary impact if you provision additional resources.
[Learn More](#)

Export... **Create** **Cancel**



4. After you configure the App Service hosting settings, select **Create**.
5. Monitor the **Output** window in Visual Studio to see the publishing status. After the app has been published, the URL for the app is logged:

```

Output
Show output from: Build
2>Adding file (ContosoTeamStats20180328033615\Views\Home\RedisCache.cshtml).
2>Adding file (ContosoTeamStats20180328033615\Views\Shared\Error.cshtml).
2>Adding file (ContosoTeamStats20180328033615\Views\Shared\_Layout.cshtml).
2>Adding file (ContosoTeamStats20180328033615\Web.config).
2>Adding file (ContosoTeamStats20180328033615\Views\_ViewStart.cshtml).
2>Adding ACLs for path (ContosoTeamStats20180328033615)
2>Adding ACLs for path (ContosoTeamStats20180328033615)
2>Publish Succeeded.
2>Web App was published successfully http://contosoteamstats20180328033615.azurewebsites.net/
===== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped =====
===== Publish: 1 succeeded, 0 failed, 0 skipped =====

```

Web Publish Activity Data Tools Operations Error List Output Package Manager Console

Add the app setting for the cache

After the new app has been published, add a new app setting. This setting is used to store the cache connection information.

To add the app setting

- Type the app name in the search bar at the top of the Azure portal to find the new app you created.

The screenshot shows the Microsoft Azure portal's search interface. A search bar at the top contains the text "ContosoTeam". Below the search bar, a "RESOURCES" section lists "ContosoTeamStats20180328033615" under "APP SERVICE". Other sections like "RESOURCE GROUPS", "SERVICES", and "MARKETPLACE" show 0 results. A "DOCUMENTATION" section links to "How to create a Web App with Redis Cache | Microsoft Docs". The search results are described as "Searching 1 of 7 subscriptions".

- Add a new app setting named **CacheConnection** for the app to use to connect to the cache. Use the same value you configured for `CacheConnection` in your `CacheSecrets.config` file. The value contains the cache host name and access key.

The screenshot shows the "Application settings" page for the "ContosoTeamStats20180328033615" app service. On the left, a sidebar lists "Create a resource", "All services", "Dashboard", "Resource groups", "All resources", "Recent", "App Services", and "SQL databases". The main area shows "Application settings" with a "SETTINGS" section containing "Application settings". A red box highlights this section. To the right, "Connection strings" are listed. At the top right, there are "Save" and "Discard" buttons, with "Save" being highlighted by a red box. A red box also highlights the "CacheConnection" entry in the "Application settings" list, which has the value "YourCacheName.redis.cache.windows.net,ab...".

Run the app in Azure

In your browser, go to the URL for the app. The URL appears in the results of the publishing operation in the Visual Studio output window. It's also provided in the Azure portal on the overview page of the app you created.

Select **Azure Cache for Redis Test** on the navigation bar to test cache access.

Command	Result
PING	PONG
GET Message	Hello! Test Message set from the Redis Console in the Portal
SET Message "Hello! The cache is working from ASP.NET!"	True
GET Message	Hello! The cache is working from ASP.NET!
CLIENT LIST	<pre>id=296037 addr=127.0.0.1:38029 fd=22 name=PORTAL_CONSOLE age=204 idle=4 flags=N db=0 sub=0 psub=0 multi=-1 qbuf=0 qbuf-free=0 obl=0 oll=0 omem=0 ow=0 oumem=0 events=r cmd=set numops=5 id=296137 addr=[REDACTED] fd=17 name=RD00155D8A496E age=0 idle=0 flags=N db=0 sub=1 psub=0 multi=-1 qbuf=0 qbuf-free=32768 obl=0 oll=0 omem=0 ow=0 oumem=0 events=r cmd=subscribe numops=5 id=296138 addr=[REDACTED] fd=20 name=RD00155D8A496E age=0 idle=0 flags=N db=0 sub=0 psub=0 multi=-1 qbuf=0 qbuf-free=32768 obl=0 oll=0 omem=0 ow=0 oumem=0 events=r cmd=client numops=17</pre>

© 2018 - My ASP.NET Application

Clean up resources

If you're continuing to the next tutorial, you can keep the resources that you created in this quickstart and reuse them.

Otherwise, if you're finished with the quickstart sample application, you can delete the Azure resources that you created in this quickstart to avoid charges.

IMPORTANT

Deleting a resource group is irreversible. When you delete a resource group, all the resources in it are permanently deleted. Make sure that you do not accidentally delete the wrong resource group or resources. If you created the resources for hosting this sample inside an existing resource group that contains resources you want to keep, you can delete each resource individually from their respective blades instead of deleting the resource group.

To delete a resource group

1. Sign in to the [Azure portal](#), and then select **Resource groups**.
2. In the **Filter by name...** box, type the name of your resource group. The instructions for this article used a resource group named *TestResources*. On your resource group, in the results list, select ..., and then select **Delete resource group**.

The screenshot shows the Microsoft Azure portal interface. On the left, the navigation bar includes 'Create a resource', 'All services', 'FAVORITES' (with 'Dashboard' and 'Resource groups' selected), 'All resources', and 'Recent'. The main content area is titled 'Resource groups' and shows a table with one item: 'TestResources' under 'NAME'. A red box highlights the 'Resource groups' link in the sidebar and the 'TestResources' row in the table. Another red box highlights the 'Delete resource group' button in the table's header.

You're asked to confirm the deletion of the resource group. Type the name of your resource group to confirm, and then select **Delete**.

After a few moments, the resource group and all of its resources are deleted.

Next steps

In the next tutorial, you use Azure Cache for Redis in a more realistic scenario to improve performance of an app. You update this application to cache leaderboard results using the cache-aside pattern with ASP.NET and a database.

[Create a cache-aside leaderboard on ASP.NET](#)

Quickstart: Use Azure Cache for Redis with a .NET Core app

12/10/2019 • 9 minutes to read • [Edit Online](#)

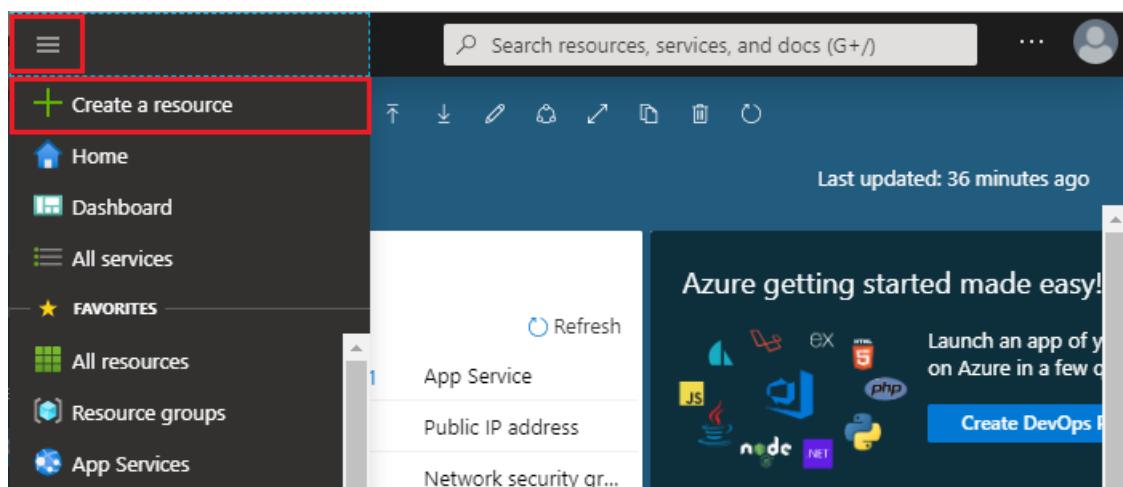
In this quickstart, you incorporate Azure Cache for Redis into a .NET Core app to have access to a secure, dedicated cache that is accessible from any application within Azure. You specifically use the [StackExchange.Redis](#) client with C# code in a .NET Core console app.

Prerequisites

- Azure subscription - [create one for free](#)
- [.NET Core SDK](#)
- [.NET Framework 4 or higher](#), which is required by the StackEdchage.Redis client.

Create a cache

1. To create a cache, sign in to the [Azure portal](#) and select **Create a resource**.



2. On the **New** page, select **Databases** and then select **Azure Cache for Redis**.

New

Search the Marketplace

Azure Marketplace See all

Featured See all

- Get started
- Recently created
- AI + Machine Learning
- Analytics
- Blockchain
- Compute
- Containers
- Databases
- Developer Tools
- DevOps
- Identity
- Integration
- Internet of Things
- Media
- Mixed Reality
- IT & Management Tools
- Networking
- Software as a Service (SaaS)
- Security
- Storage
- Web

Azure SQL Managed Instance
[Quickstart tutorial](#)

SQL Database
[Quickstart tutorial](#)

SQL Data Warehouse
[Quickstart tutorial](#)

Azure Database for MariaDB
[Learn more](#)

Couchbase Enterprise Edition (Hourly Pricing) (preview)
PREVIEW
[Learn more](#)

Azure Database for MySQL
[Quickstart tutorial](#)

Azure Database for PostgreSQL
[Quickstart tutorial](#)

Azure Cosmos DB
[Quickstart tutorial](#)

SQL Server 2017 Enterprise Windows Server 2016
[Learn more](#)

Azure Cache for Redis
[Quickstart tutorial](#)

- On the **New Redis Cache** page, configure the settings for your new cache.

SETTING	SUGGESTED VALUE	DESCRIPTION
DNS name	Enter a globally unique name.	The cache name must be a string between 1 and 63 characters that contains only numbers, letters, or hyphens. The name must start and end with a number or letter, and can't contain consecutive hyphens. Your cache instance's <i>host name</i> will be < <i>DNS name</i> >.redis.cache.windows.net.
Subscription	Drop down and select your subscription.	The subscription under which to create this new Azure Cache for Redis instance.

SETTING	SUGGESTED VALUE	DESCRIPTION
Resource group	Drop down and select a resource group, or select Create new and enter a new resource group name.	Name for the resource group in which to create your cache and other resources. By putting all your app resources in one resource group, you can easily manage or delete them together.
Location	Drop down and select a location.	Select a region near other services that will use your cache.
Pricing tier	Drop down and select a Pricing tier .	The pricing tier determines the size, performance, and features that are available for the cache. For more information, see Azure Cache for Redis Overview .

4. Select **Create**.

New Redis Cache

DNS name *

contoso-tmp .redis.cache.windows.net

Subscription *

A Subscription

Resource group * ⓘ

(New) contoso-rg-tmp

[Create new](#)

Location *

Central US

Pricing tier ([View full pricing details](#)) *

Basic C0 (250 MB Cache)

(PREVIEW) Availability zone

Requires Premium tier

Redis Cluster ⓘ

Requires Premium tier

Data persistence ⓘ

Requires Premium tier

Virtual Network ⓘ

Requires Premium tier

Unblock port 6379 (not SSL encrypted)

Create [Automation options](#)

It takes a while for the cache to create. You can monitor progress on the Azure Cache for Redis [Overview](#) page. When **Status** shows as **Running**, the cache is ready to use.

The screenshot shows the Azure portal interface for the 'contoso-tmp' resource. The left sidebar contains navigation links: Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Settings, and Access keys. The main content area displays resource details for 'contoso-tmp'. At the top are buttons for Console, Move, and Delete. Below are sections for Resource group, Host name, Activity log, Ports, Access control, Keys, Tags, and Diagnose and solve problems. A red box highlights the 'Status' section, which shows 'Running - Basic 250 MB'. Other status details include Location (Central US), Subscription (A Subscription), and Subscription ID (12345678-1234-1234-1234-123456781234). The 'Keys' section includes links to 'Show access keys...' and 'Best practices'.

Resource group (change) contoso-rg-tmp	
Status	Host name contoso-tmp.redis.cache.windows.net
Running - Basic 250 MB	Ports Non-SSL port (6379) disabled
Location Central US	Keys Show access keys...
Subscription (change) A Subscription	*Best practices* https://aka.ms/redis/p/bestpractices
Subscription ID 12345678-1234-1234-1234-123456781234	*New features* https://aka.ms/newfeatures

Retrieve host name, ports, and access keys from the Azure portal

To connect to an Azure Cache for Redis instance, cache clients need the host name, ports, and a key for the cache. Some clients might refer to these items by slightly different names. You can get the host name, ports, and keys from the [Azure portal](#).

- To get the access keys, from your cache left navigation, select **Access keys**.

- To get the host name and ports, from your cache left navigation, select **Properties**. The host name is of the form <DNS name>.redis.cache.windows.net.

The screenshot shows the Azure Cache for Redis properties page for a resource named 'contoso-tmp'. On the left, there's a sidebar with various settings like Access keys, Advanced settings, Scale, Cluster size, Data persistence, Schedule updates, Geo-replication, Virtual Network, Firewall, and Properties. The 'Properties' item is highlighted with a red box. The main pane displays the status as 'Running', host name as 'contoso-tmp.redis.cache.windows.net' (with a 'Copy to clipboard' button), non-SSL port as 'Disabled', SSL port as '6380', and location as 'Central US'.

Make a note of the **HOST NAME** and the **Primary** access key. You will use these values later to construct the `CacheConnection` secret.

Create a console app

Open a new command window and execute the following command to create a new .NET Core console app:

```
dotnet new console -o Redistest
```

In your command window, change to the new `Redistest` project directory.

Add Secret Manager to the project

In this section, you will add the [Secret Manager tool](#) to your project. The Secret Manager tool stores sensitive data for development work outside of your project tree. This approach helps prevent the accidental sharing of app secrets within source code.

Open your `Redistest.csproj` file. Add a `DotNetCliToolReference` element to include `Microsoft.Extensions.SecretManager.Tools`. Also add a `UserSecretsId` element as shown below, and save the file.

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>netcoreapp2.0</TargetFramework>
    <UserSecretsId>Redistest</UserSecretsId>
  </PropertyGroup>
  <ItemGroup>
    <DotNetCliToolReference Include="Microsoft.Extensions.SecretManager.Tools" Version="2.0.0" />
  </ItemGroup>
</Project>
```

Execute the following command to add the `Microsoft.Extensions.Configuration.UserSecrets` package to the project:

```
dotnet add package Microsoft.Extensions.Configuration.UserSecrets
```

Execute the following command to restore your packages:

```
dotnet restore
```

In your command window, execute the following command to store a new secret named *CacheConnection*, after replacing the placeholders (including angle brackets) for your cache name and primary access key:

```
dotnet user-secrets set CacheConnection "<cache  
name>.redis.cache.windows.net,abortConnect=false,ssl=true,password=<primary-access-key>"
```

Add the following `using` statement to *Program.cs*:

```
using Microsoft.Extensions.Configuration;
```

Add the following members to the `Program` class in *Program.cs*. This code initializes a configuration to access the user secret for the Azure Cache for Redis connection string.

```
private static IConfigurationRoot Configuration { get; set; }  
const string SecretName = "CacheConnection";  
  
private static void InitializeConfiguration()  
{  
    var builder = new ConfigurationBuilder()  
        .AddUserSecrets<Program>();  
  
    Configuration = builder.Build();  
}
```

Configure the cache client

In this section, you will configure the console application to use the [StackExchange.Redis](#) client for .NET.

In your command window, execute the following command in the *Redistest* project directory:

```
dotnet add package StackExchange.Redis
```

Once the installation is completed, the *StackExchange.Redis* cache client is available to use with your project.

Connect to the cache

Add the following `using` statement to *Program.cs*:

```
using StackExchange.Redis;
```

The connection to the Azure Cache for Redis is managed by the `ConnectionMultiplexer` class. This class should be shared and reused throughout your client application. Do not create a new connection for each operation.

In *Program.cs*, add the following members to the `Program` class of your console application:

```
private static Lazy<ConnectionMultiplexer> lazyConnection = new Lazy<ConnectionMultiplexer>(() =>
{
    string cacheConnection = Configuration[SecretName];
    return ConnectionMultiplexer.Connect(cacheConnection);
});

public static ConnectionMultiplexer Connection
{
    get
    {
        return lazyConnection.Value;
    }
}
```

This approach to sharing a `ConnectionMultiplexer` instance in your application uses a static property that returns a connected instance. The code provides a thread-safe way to initialize only a single connected `ConnectionMultiplexer` instance. `abortConnect` is set to false, which means that the call succeeds even if a connection to the Azure Cache for Redis is not established. One key feature of `ConnectionMultiplexer` is that it automatically restores connectivity to the cache once the network issue or other causes are resolved.

The value of the `CacheConnection` secret is accessed using the Secret Manager configuration provider and used as the password parameter.

Executing cache commands

In `Program.cs`, add the following code for the `Main` procedure of the `Program` class for your console application:

```

static void Main(string[] args)
{
    InitializeConfiguration();

    // Connection refers to a property that returns a ConnectionMultiplexer
    // as shown in the previous example.
    IDatabase cache = lazyConnection.Value.GetDatabase();

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

    // Simple PING command
    string cacheCommand = "PING";
    Console.WriteLine("\nCache command : " + cacheCommand);
    Console.WriteLine("Cache response : " + cache.Execute(cacheCommand).ToString());

    // Simple get and put of integral data types into the cache
    cacheCommand = "GET Message";
    Console.WriteLine("\nCache command : " + cacheCommand + " or StringGet()");
    Console.WriteLine("Cache response : " + cache.StringGet("Message").ToString());

    cacheCommand = "SET Message \"Hello! The cache is working from a .NET Core console app!\"";
    Console.WriteLine("\nCache command : " + cacheCommand + " or StringSet()");
    Console.WriteLine("Cache response : " + cache.StringSet("Message", "Hello! The cache is working
from a .NET Core console app!").ToString());

    // Demonstrate "SET Message" executed as expected...
    cacheCommand = "GET Message";
    Console.WriteLine("\nCache command : " + cacheCommand + " or StringGet()");
    Console.WriteLine("Cache response : " + cache.StringGet("Message").ToString());

    // Get the client list, useful to see if connection list is growing...
    cacheCommand = "CLIENT LIST";
    Console.WriteLine("\nCache command : " + cacheCommand);
    Console.WriteLine("Cache response : \n" + cache.Execute("CLIENT",
"LIST").ToString().Replace("id=", "id="));

    lazyConnection.Value.Dispose();
}

```

Save *Program.cs*.

Azure Cache for Redis has a configurable number of databases (default of 16) that can be used to logically separate the data within an Azure Cache for Redis. The code connects to the default database, DB 0. For more information, see [What are Redis databases?](#) and [Default Redis server configuration](#).

Cache items can be stored and retrieved by using the `StringSet` and `StringGet` methods.

Redis stores most data as Redis strings, but these strings can contain many types of data, including serialized binary data, which can be used when storing .NET objects in the cache.

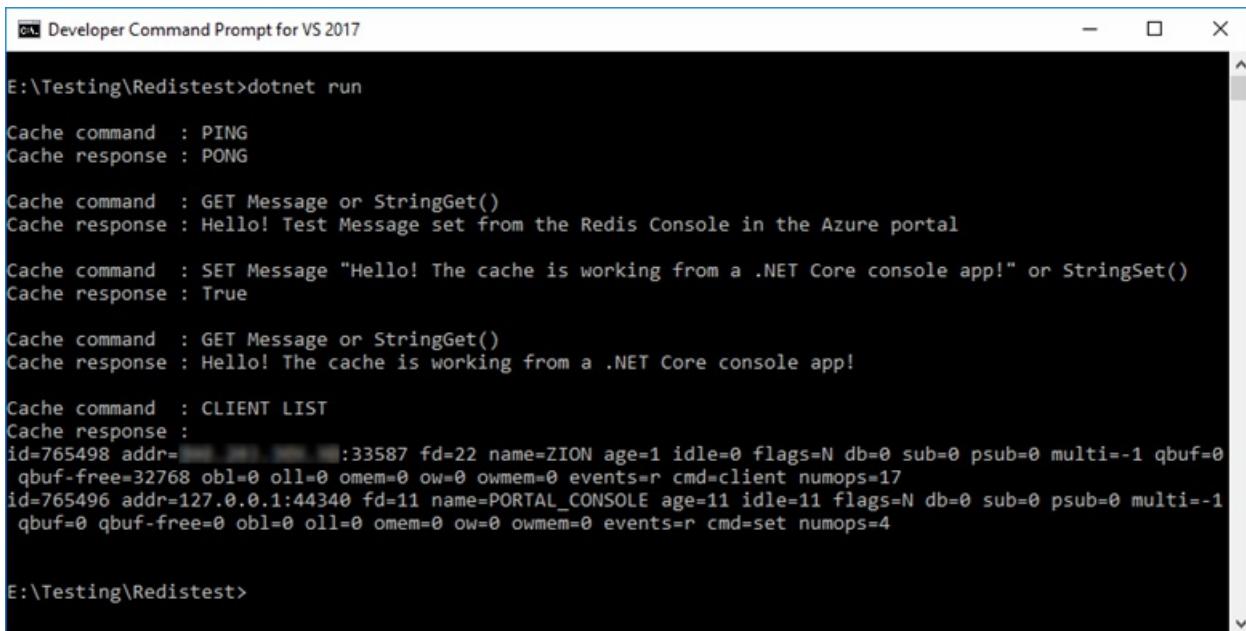
Execute the following command in your command window to build the app:

```
dotnet build
```

Then run the app with the following command:

```
dotnet run
```

In the example below, you can see the `Message` key previously had a cached value, which was set using the Redis Console in the Azure portal. The app updated that cached value. The app also executed the `PING` and `CLIENT LIST` commands.



E:\Testing\Redistest>dotnet run

Cache command : PING
Cache response : PONG

Cache command : GET Message or StringGet()
Cache response : Hello! Test Message set from the Redis Console in the Azure portal

Cache command : SET Message "Hello! The cache is working from a .NET Core console app!" or StringSet()
Cache response : True

Cache command : GET Message or StringGet()
Cache response : Hello! The cache is working from a .NET Core console app!

Cache command : CLIENT LIST
Cache response :

```
id=765498 addr=[REDACTED]:33587 fd=22 name=ZION age=1 idle=0 flags=N db=0 sub=0 psub=0 multi=-1 qbuf=0
qbuf-free=32768 obl=0 oll=0 omem=0 ow=0 owmem=0 events=r cmd=client numops=17
id=765496 addr=127.0.0.1:44340 fd=11 name=PORTAL_CONSOLE age=11 idle=11 flags=N db=0 sub=0 psub=0 multi=-1
qbuf=0 qbuf-free=0 obl=0 oll=0 omem=0 ow=0 owmem=0 events=r cmd=set numops=4
```

E:\Testing\Redistest>

Work with .NET objects in the cache

Azure Cache for Redis can cache both .NET objects and primitive data types, but before a .NET object can be cached it must be serialized. This .NET object serialization is the responsibility of the application developer, and gives the developer flexibility in the choice of the serializer.

One simple way to serialize objects is to use the `JsonConvert` serialization methods in [Newtonsoft.Json](#) and serialize to and from JSON. In this section, you will add a .NET object to the cache.

Execute the following command to add the [Newtonsoft.json](#) package to the app:

```
dotnet add package Newtonsoft.json
```

Add the following `using` statement to the top of *Program.cs*:

```
using Newtonsoft.Json;
```

Add the following `Employee` class definition to *Program.cs*:

```
class Employee
{
    public string Id { get; set; }
    public string Name { get; set; }
    public int Age { get; set; }

    public Employee(string EmployeeId, string Name, int Age)
    {
        this.Id = EmployeeId;
        this.Name = Name;
        this.Age = Age;
    }
}
```

At the bottom of `Main()` procedure in *Program.cs*, and before the call to `Dispose()`, add the following lines of code to cache and retrieve a serialized .NET object:

```

// Store .NET object to cache
Employee e007 = new Employee("007", "Davide Columbo", 100);
Console.WriteLine("Cache response from storing Employee .NET object : " +
    cache.StringSet("e007", JsonConvert.SerializeObject(e007)));

// Retrieve .NET object from cache
Employee e007FromCache = JsonConvert.DeserializeObject<Employee>(cache.StringGet("e007"));
Console.WriteLine("Deserialized Employee .NET object :\n");
Console.WriteLine("\tEmployee.Name : " + e007FromCache.Name);
Console.WriteLine("\tEmployee.Id : " + e007FromCache.Id);
Console.WriteLine("\tEmployee.Age : " + e007FromCache.Age + "\n");

```

Save *Program.cs* and rebuild the app with the following command:

```
dotnet build
```

Run the app with the following command to test serialization of .NET objects:

```
dotnet run
```

```

Developer Command Prompt for VS 2017
E:\Testing\Redistest>dotnet run

Cache command : PING
Cache response : PONG

Cache command : GET Message or StringGet()
Cache response : Hello! Test Message set from the Redis Console in the Azure portal

Cache command : SET Message "Hello! The cache is working from a .NET Core console app!" or StringSet()
Cache response : True

Cache command : GET Message or StringGet()
Cache response : Hello! The cache is working from a .NET Core console app!

Cache command : CLIENT LIST
Cache response :
id=765496 addr=127.0.0.1:44340 fd=11 name=PORTAL_CONSOLE age=377 idle=25 flags=N db=0 sub=0 psub=0 multi=-1
qbuf=0 qbuf-free=0 obl=0 oll=0 omem=0 ow=0 owmem=0 events=r cmd=set numops=5
id=765684 addr=127.0.0.1:33615 fd=14 name=ZION age=1 idle=0 flags=N db=0 sub=0 psub=0 multi=-1 qbuf=0
qbuf-free=32768 obl=0 oll=0 omem=0 ow=0 owmem=0 events=r cmd=client numops=17

Cache response from storing Employee .NET object : True
Deserialized Employee .NET object :

    Employee.Name : Davide Columbo
    Employee.Id   : 007
    Employee.Age  : 100

E:\Testing\Redistest>

```

Clean up resources

If you will be continuing to the next tutorial, you can keep the resources created in this quickstart and reuse them.

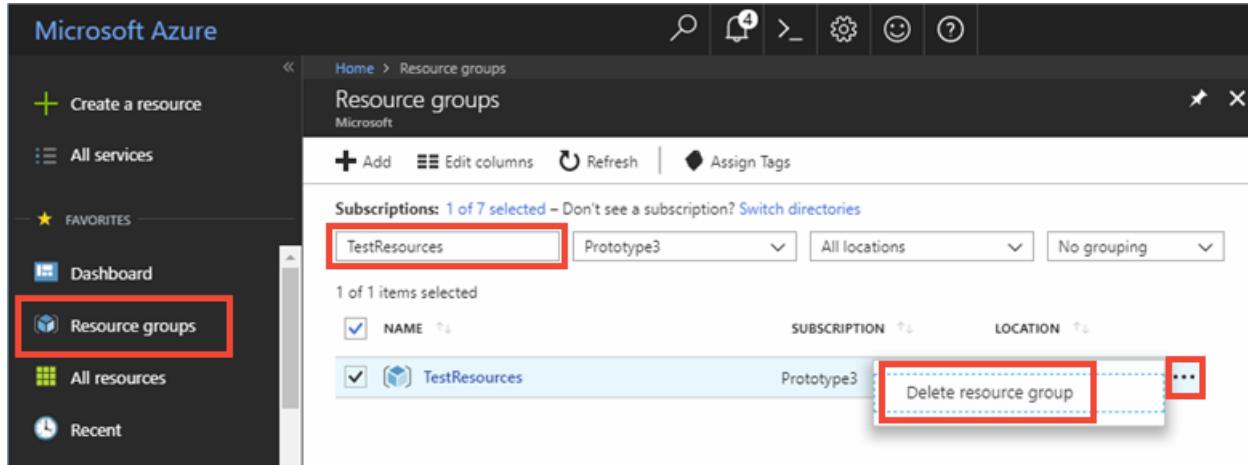
Otherwise, if you are finished with the quickstart sample application, you can delete the Azure resources created in this quickstart to avoid charges.

IMPORTANT

Deleting a resource group is irreversible and that the resource group and all the resources in it are permanently deleted. Make sure that you do not accidentally delete the wrong resource group or resources. If you created the resources for hosting this sample inside an existing resource group that contains resources you want to keep, you can delete each resource individually from their respective blades instead of deleting the resource group.

Sign in to the [Azure portal](#) and click **Resource groups**.

In the **Filter by name...** textbox, type the name of your resource group. The instructions for this article used a resource group named *TestResources*. On your resource group in the result list, click ... then **Delete resource group**.



The screenshot shows the Microsoft Azure portal's Resource groups blade. On the left sidebar, the 'Resource groups' item is highlighted with a red box. In the main area, a search bar at the top has 'TestResources' typed into it and is also highlighted with a red box. Below the search bar, there are filter options for 'Subscription' (set to 'Prototype3'), 'Location' (set to 'All locations'), and 'Grouping' (set to 'No grouping'). A table lists one item: 'TestResources'. The 'NAME' column shows 'TestResources' with a checked checkbox. To the right of the table, there are columns for 'SUBSCRIPTION' and 'LOCATION'. At the bottom right of the table row, there is a blue button labeled 'Delete resource group' with a red box around it. Above the table, a message says 'Subscriptions: 1 of 7 selected - Don't see a subscription? Switch directories'.

You will be asked to confirm the deletion of the resource group. Type the name of your resource group to confirm, and click **Delete**.

After a few moments, the resource group and all of its contained resources are deleted.

Next steps

In this quickstart, you learned how to use Azure Cache for Redis from a .NET Core application. Continue to the next quickstart to use Azure Cache for Redis with an ASP.NET web app.

[Create an ASP.NET web app that uses an Azure Cache for Redis.](#)

Quickstart: Use Azure Cache for Redis with a .NET Framework application

12/10/2019 • 8 minutes to read • [Edit Online](#)

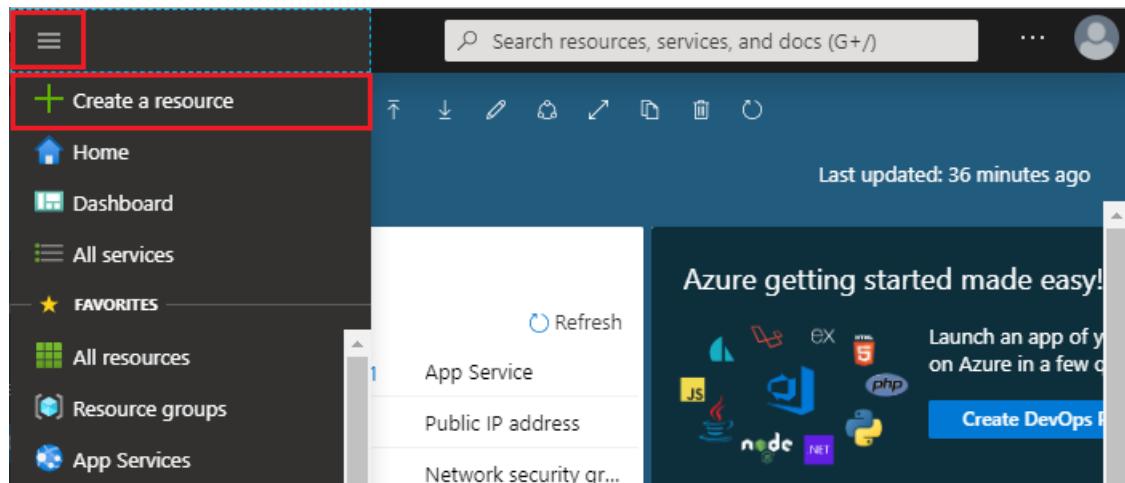
In this quickstart, you incorporate Azure Cache for Redis into a .NET Framework app to have access to a secure, dedicated cache that is accessible from any application within Azure. You specifically use the [StackExchange.Redis](#) client with C# code in a .NET console app.

Prerequisites

- Azure subscription - [create one for free](#)
- [Visual Studio 2019](#)
- [.NET Framework 4 or higher](#), which is required by the StackExchange.Redis client.

Create a cache

1. To create a cache, sign in to the [Azure portal](#) and select **Create a resource**.



2. On the **New** page, select **Databases** and then select **Azure Cache for Redis**.

New

Search the Marketplace

Azure Marketplace See all

Get started
Recently created
AI + Machine Learning
Analytics
Blockchain
Compute
Containers
Databases
Developer Tools
DevOps
Identity
Integration
Internet of Things
Media
Mixed Reality
IT & Management Tools
Networking
Software as a Service (SaaS)
Security
Storage
Web

Featured See all

 Azure SQL Managed Instance
[Quickstart tutorial](#)

 SQL Database
[Quickstart tutorial](#)

 SQL Data Warehouse
[Quickstart tutorial](#)

 Azure Database for MariaDB
[Learn more](#)

 Couchbase Enterprise Edition (Hourly Pricing) (preview)
[Learn more](#)

 Azure Database for MySQL
[Quickstart tutorial](#)

 Azure Database for PostgreSQL
[Quickstart tutorial](#)

 Azure Cosmos DB
[Quickstart tutorial](#)

 SQL Server 2017 Enterprise Windows Server 2016
[Learn more](#)

 Azure Cache for Redis
[Quickstart tutorial](#)

- On the **New Redis Cache** page, configure the settings for your new cache.

SETTING	SUGGESTED VALUE	DESCRIPTION
DNS name	Enter a globally unique name.	The cache name must be a string between 1 and 63 characters that contains only numbers, letters, or hyphens. The name must start and end with a number or letter, and can't contain consecutive hyphens. Your cache instance's <i>host name</i> will be < <i>DNS name</i> >.redis.cache.windows.net.
Subscription	Drop down and select your subscription.	The subscription under which to create this new Azure Cache for Redis instance.

SETTING	SUGGESTED VALUE	DESCRIPTION
Resource group	Drop down and select a resource group, or select Create new and enter a new resource group name.	Name for the resource group in which to create your cache and other resources. By putting all your app resources in one resource group, you can easily manage or delete them together.
Location	Drop down and select a location.	Select a region near other services that will use your cache.
Pricing tier	Drop down and select a Pricing tier .	The pricing tier determines the size, performance, and features that are available for the cache. For more information, see Azure Cache for Redis Overview .

4. Select **Create**.

New Redis Cache

DNS name *

 .redis.cache.windows.net

Subscription *

A Subscription

Resource group * ⓘ

(New) contoso-rg-tmp

[Create new](#)

Location *

Central US

Pricing tier ([View full pricing details](#)) *

Basic C0 (250 MB Cache)

(PREVIEW) Availability zone

Requires Premium tier

Redis Cluster ⓘ

Requires Premium tier

Data persistence ⓘ

Requires Premium tier

Virtual Network ⓘ

Requires Premium tier

Unblock port 6379 (not SSL encrypted)

Create [Automation options](#)

It takes a while for the cache to create. You can monitor progress on the Azure Cache for Redis [Overview](#) page. When **Status** shows as **Running**, the cache is ready to use.

The screenshot shows the Azure portal interface for managing a Redis cache. On the left, there's a sidebar with navigation links: Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, and Settings. Below the sidebar is a 'Search (Ctrl+/' bar. The main content area has a header with 'contoso-tmp' and 'Azure Cache for Redis'. It includes 'Console', 'Move', and 'Delete' buttons. The 'Overview' section is selected. The 'Status' field is highlighted with a red box and contains the text 'Running - Basic 250 MB'. Other visible details include the resource group 'contoso-rg-tmp', location 'Central US', subscription 'A Subscription', and subscription ID '12345678-1234-1234-1234-123456781234'. To the right, there are sections for host name ('contoso-tmp.redis.cache.windows.net'), ports ('Non-SSL port (6379) disabled'), and keys ('Show access keys...'). There are also links for best practices and new features.

Setting	Value
Resource group (change)	contoso-rg-tmp
Status	Running - Basic 250 MB
Location	Central US
Subscription (change)	A Subscription
Subscription ID	12345678-1234-1234-1234-123456781234
Host name	contoso-tmp.redis.cache.windows.net
Ports	Non-SSL port (6379) disabled
Keys	Show access keys...
Best practices	https://aka.ms/redis/p/bestpractices
New features	https://aka.ms/newfeatures

Retrieve host name, ports, and access keys from the Azure portal

To connect to an Azure Cache for Redis instance, cache clients need the host name, ports, and a key for the cache. Some clients might refer to these items by slightly different names. You can get the host name, ports, and keys from the [Azure portal](#).

- To get the access keys, from your cache left navigation, select **Access keys**.

- To get the host name and ports, from your cache left navigation, select **Properties**. The host name is of the form <DNS name>.redis.cache.windows.net.

The screenshot shows the Azure Cache for Redis Properties page for a cache named 'contoso-tmp'. The left sidebar lists various settings like Access keys, Advanced settings, Scale, Cluster size, Data persistence, Schedule updates, Geo-replication, Virtual Network, and Firewall. The 'Properties' section is highlighted with a red box. On the right, the Status is listed as 'Running'. The Host name is 'contoso-tmp.redis.cache.windows.net', with a 'Copy to clipboard' button and a copy icon. The Non-SSL Port is set to 'Disabled', and the SSL Port is '6380', both with copy icons. The Location is 'Central US'.

Create a file on your computer named *CacheSecrets.config* and place it in a location where it won't be checked in with the source code of your sample application. For this quickstart, the *CacheSecrets.config* file is located here, C:\AppSecrets\CacheSecrets.config.

Edit the *CacheSecrets.config* file and add the following contents:

```
<appSettings>
  <add key="CacheConnection" value=".redis.cache.windows.net,abortConnect=false,ssl=true,password=<access-key>"/>
</appSettings>
```

Replace `<cache-name>` with your cache host name.

Replace `<access-key>` with the primary key for your cache.

Create a console app

In Visual Studio, click **File > New > Project**.

Under **Visual C#**, click **Windows Classic Desktop** and then click **Console App**, and **OK** to create a new console application.

Configure the cache client

In this section, you will configure the console application to use the [StackExchange.Redis](#) client for .NET.

In Visual Studio, click **Tools > NuGet Package Manager > Package Manager Console**, and run the following command from the Package Manager Console window.

```
Install-Package StackExchange.Redis
```

Once the installation is completed, the *StackExchange.Redis* cache client is available to use with your project.

Connect to the cache

In Visual Studio, open your *App.config* file and update it to include an `appSettings` `file` attribute that references the *CacheSecrets.config* file.

```

<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <startup>
        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.7.1" />
    </startup>

    <appSettings file="C:\AppSecrets\CacheSecrets.config"></appSettings>

</configuration>

```

In Solution Explorer, right-click **References** and click **Add a reference**. Add a reference to the **System.Configuration** assembly.

Add the following `using` statements to *Program.cs*:

```

using StackExchange.Redis;
using System.Configuration;

```

The connection to the Azure Cache for Redis is managed by the `ConnectionMultiplexer` class. This class should be shared and reused throughout your client application. Do not create a new connection for each operation.

Never store credentials in source code. To keep this sample simple, I'm only using an external secrets config file. A better approach would be to use [Azure Key Vault with certificates](#).

In *Program.cs*, add the following members to the `Program` class of your console application:

```

private static Lazy<ConnectionMultiplexer> lazyConnection = new Lazy<ConnectionMultiplexer>(() =>
{
    string cacheConnection = ConfigurationManager.AppSettings["CacheConnection"].ToString();
    return ConnectionMultiplexer.Connect(cacheConnection);
});

public static ConnectionMultiplexer Connection
{
    get
    {
        return lazyConnection.Value;
    }
}

```

This approach to sharing a `ConnectionMultiplexer` instance in your application uses a static property that returns a connected instance. The code provides a thread-safe way to initialize only a single connected `ConnectionMultiplexer` instance. `abortConnect` is set to false, which means that the call succeeds even if a connection to the Azure Cache for Redis is not established. One key feature of `ConnectionMultiplexer` is that it automatically restores connectivity to the cache once the network issue or other causes are resolved.

The value of the `CacheConnection` appSetting is used to reference the cache connection string from the Azure portal as the password parameter.

Executing cache commands

Add the following code for the `Main` procedure of the `Program` class for your console application:

```

static void Main(string[] args)
{
    // Connection refers to a property that returns a ConnectionMultiplexer
    // as shown in the previous example.
    IDatabase cache = lazyConnection.Value.GetDatabase();

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

    // Simple PING command
    string cacheCommand = "PING";
    Console.WriteLine("\nCache command : " + cacheCommand);
    Console.WriteLine("Cache response : " + cache.Execute(cacheCommand).ToString());

    // Simple get and put of integral data types into the cache
    cacheCommand = "GET Message";
    Console.WriteLine("\nCache command : " + cacheCommand + " or StringGet()");
    Console.WriteLine("Cache response : " + cache.StringGet("Message").ToString());

    cacheCommand = "SET Message \"Hello! The cache is working from a .NET console app!\"";
    Console.WriteLine("\nCache command : " + cacheCommand + " or StringSet()");
    Console.WriteLine("Cache response : " + cache.StringSet("Message", "Hello! The cache is working
from a .NET console app!").ToString());

    // Demonstrate "SET Message" executed as expected...
    cacheCommand = "GET Message";
    Console.WriteLine("\nCache command : " + cacheCommand + " or StringGet()");
    Console.WriteLine("Cache response : " + cache.StringGet("Message").ToString());

    // Get the client list, useful to see if connection list is growing...
    cacheCommand = "CLIENT LIST";
    Console.WriteLine("\nCache command : " + cacheCommand);
    Console.WriteLine("Cache response : \n" + cache.Execute("CLIENT",
"LIST").ToString().Replace("id=", "id="));

    lazyConnection.Value.Dispose();
}

```

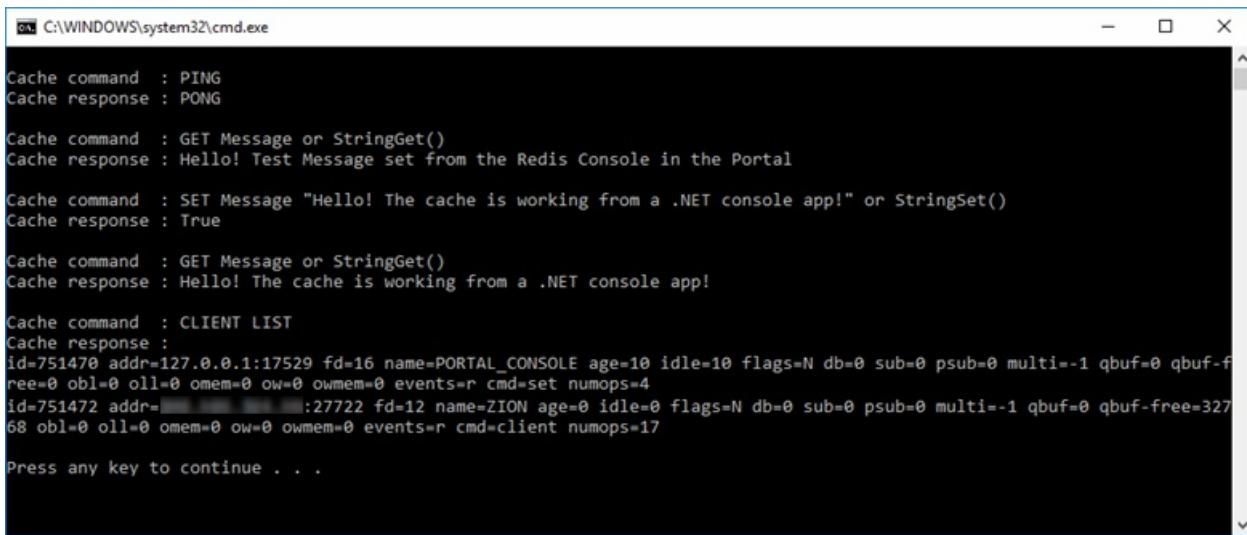
Azure Cache for Redis has a configurable number of databases (default of 16) that can be used to logically separate the data within an Azure Cache for Redis. The code connects to the default database, DB 0. For more information, see [What are Redis databases?](#) and [Default Redis server configuration](#).

Cache items can be stored and retrieved by using the `StringSet` and `StringGet` methods.

Redis stores most data as Redis strings, but these strings can contain many types of data, including serialized binary data, which can be used when storing .NET objects in the cache.

Press **Ctrl+F5** to build and run the console app.

In the example below, you can see the `Message` key previously had a cached value, which was set using the Redis Console in the Azure portal. The app updated that cached value. The app also executed the `PING` and `CLIENT LIST` commands.



A screenshot of a Windows Command Prompt window titled 'C:\WINDOWS\system32\cmd.exe'. The window displays a series of Redis commands and their responses. The commands include PING, GET, SET, and CLIENT LIST. The output shows the Redis server's memory usage and connection details. The window has standard window controls (minimize, maximize, close) at the top right.

```
Cache command : PING
Cache response : PONG

Cache command : GET Message or StringGet()
Cache response : Hello! Test Message set from the Redis Console in the Portal

Cache command : SET Message "Hello! The cache is working from a .NET console app!" or StringSet()
Cache response : True

Cache command : GET Message or StringGet()
Cache response : Hello! The cache is working from a .NET console app!

Cache command : CLIENT LIST
Cache response :
id=751470 addr=127.0.0.1:17529 fd=16 name=PORTAL_CONSOLE age=10 idle=10 flags=N db=0 sub=0 psub=0 multi=-1 qbuf=0 qbuf-free=0 obl=0 oll=0 omem=0 ow=0 owmem=0 events=r cmd=set numops=4
id=751472 addr=[REDACTED]:27722 fd=12 name=ZION age=0 idle=0 flags=N db=0 sub=0 psub=0 multi=-1 qbuf=0 qbuf-free=32768 obl=0 oll=0 omem=0 ow=0 owmem=0 events=r cmd=client numops=17

Press any key to continue . . .
```

Work with .NET objects in the cache

Azure Cache for Redis can cache both .NET objects and primitive data types, but before a .NET object can be cached it must be serialized. This .NET object serialization is the responsibility of the application developer, and gives the developer flexibility in the choice of the serializer.

One simple way to serialize objects is to use the `JsonConvert` serialization methods in [Newtonsoft.Json](#) and serialize to and from JSON. In this section, you will add a .NET object to the cache.

In Visual Studio, click **Tools > NuGet Package Manager > Package Manager Console**, and run the following command from the Package Manager Console window.

```
Install-Package Newtonsoft.Json
```

Add the following `using` statement to the top of *Program.cs*:

```
using Newtonsoft.Json;
```

Add the following `Employee` class definition to *Program.cs*:

```
class Employee
{
    public string Id { get; set; }
    public string Name { get; set; }
    public int Age { get; set; }

    public Employee(string EmployeeId, string Name, int Age)
    {
        this.Id = EmployeeId;
        this.Name = Name;
        this.Age = Age;
    }
}
```

At the bottom of `Main()` procedure in *Program.cs*, and before the call to `Dispose()`, add the following lines of code to cache and retrieve a serialized .NET object:

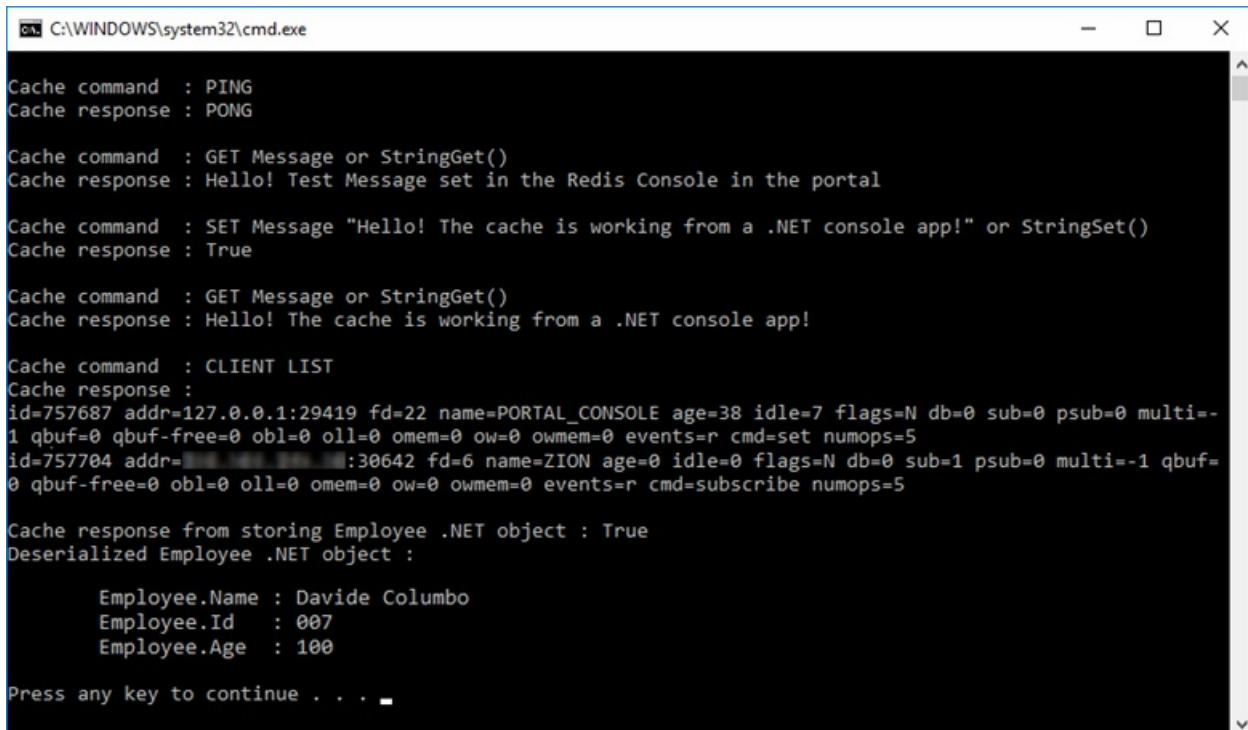
```

// Store .NET object to cache
Employee e007 = new Employee("007", "Davide Columbo", 100);
Console.WriteLine("Cache response from storing Employee .NET object : " +
    cache.StringSet("e007", JsonConvert.SerializeObject(e007)));

// Retrieve .NET object from cache
Employee e007FromCache = JsonConvert.DeserializeObject<Employee>(cache.StringGet("e007"));
Console.WriteLine("Deserialized Employee .NET object :\n");
Console.WriteLine("\tEmployee.Name : " + e007FromCache.Name);
Console.WriteLine("\tEmployee.Id   : " + e007FromCache.Id);
Console.WriteLine("\tEmployee.Age  : " + e007FromCache.Age + "\n");

```

Press **Ctrl+F5** to build and run the console app to test serialization of .NET objects.



The screenshot shows a command prompt window titled 'cmd' with the path 'C:\WINDOWS\system32\cmd.exe'. The window displays the following text:

```

Cache command : PING
Cache response : PONG

Cache command : GET Message or StringGet()
Cache response : Hello! Test Message set in the Redis Console in the portal

Cache command : SET Message "Hello! The cache is working from a .NET console app!" or StringSet()
Cache response : True

Cache command : GET Message or StringGet()
Cache response : Hello! The cache is working from a .NET console app!

Cache command : CLIENT LIST
Cache response :
id=757687 addr=127.0.0.1:29419 fd=22 name=PORTAL_CONSOLE age=38 idle=7 flags=N db=0 sub=0 psub=0 multi=-1
qbuf=0 qbuf-free=0 obl=0 oll=0 omem=0 ow=0 owmem=0 events=r cmd=set numops=5
id=757704 addr=[REDACTED]:30642 fd=6 name=ZION age=0 idle=0 flags=N db=0 sub=1 psub=0 multi=-1
qbuf=0 qbuf-free=0 obl=0 oll=0 omem=0 ow=0 owmem=0 events=r cmd=subscribe numops=5

Cache response from storing Employee .NET object : True
Deserialized Employee .NET object :

        Employee.Name : Davide Columbo
        Employee.Id   : 007
        Employee.Age  : 100

Press any key to continue . . .

```

Clean up resources

If you will be continuing to the next tutorial, you can keep the resources created in this quickstart and reuse them.

Otherwise, if you are finished with the quickstart sample application, you can delete the Azure resources created in this quickstart to avoid charges.

IMPORTANT

Deleting a resource group is irreversible and that the resource group and all the resources in it are permanently deleted.

Make sure that you do not accidentally delete the wrong resource group or resources. If you created the resources for hosting this sample inside an existing resource group that contains resources you want to keep, you can delete each resource individually from their respective blades instead of deleting the resource group.

Sign in to the [Azure portal](#) and click **Resource groups**.

In the **Filter by name...** textbox, type the name of your resource group. The instructions for this article used a resource group named *TestResources*. On your resource group in the result list, click ... then **Delete resource group**.

The screenshot shows the Microsoft Azure portal interface. On the left, the navigation bar includes 'Create a resource', 'All services', 'FAVORITES' (with 'Resource groups' highlighted), 'Dashboard', 'All resources', and 'Recent'. The main content area is titled 'Resource groups' under 'Microsoft'. It shows a table with one item selected: 'TestResources' (Subscription: Prototype3). A red box highlights the 'TestResources' input field in the 'Subscriptions' section. Another red box highlights the 'Delete resource group' button in the context menu that appears when clicking the three-dot ellipsis icon.

You will be asked to confirm the deletion of the resource group. Type the name of your resource group to confirm, and click **Delete**.

After a few moments, the resource group and all of its contained resources are deleted.

Next steps

In this quickstart, you learned how to use Azure Cache for Redis from a .NET application. Continue to the next quickstart to use Azure Cache for Redis with an ASP.NET web app.

[Create an ASP.NET web app that uses an Azure Cache for Redis.](#)

Quickstart: Use Azure Cache for Redis with Java

11/15/2019 • 5 minutes to read • [Edit Online](#)

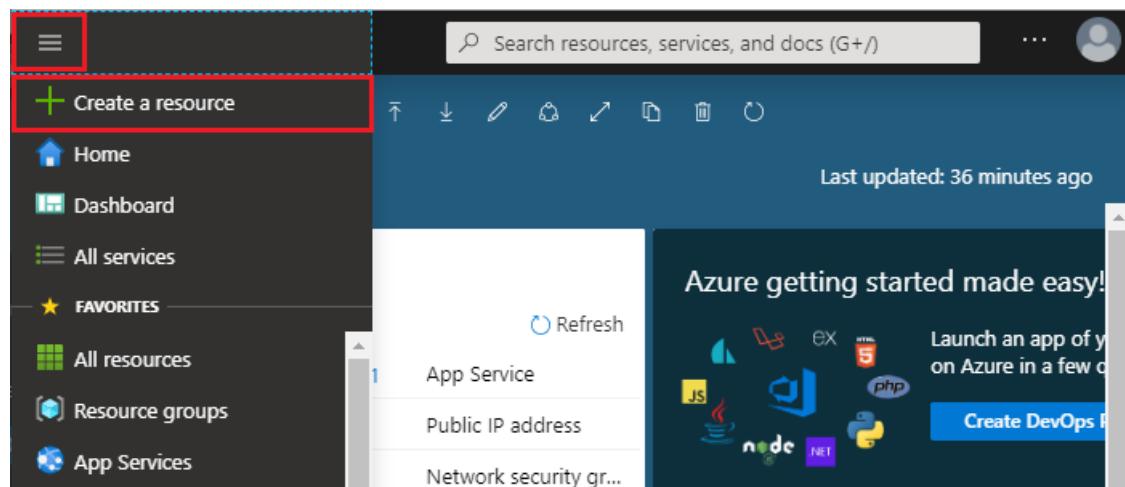
In this quickstart, you incorporate Azure Cache for Redis into a Java app using the [Jedis](#) Redis client to have access to a secure, dedicated cache that is accessible from any application within Azure.

Prerequisites

- Azure subscription - [create one for free](#)
- [Apache Maven](#)

Create an Azure Cache for Redis

1. To create a cache, sign in to the [Azure portal](#) and select **Create a resource**.



2. On the **New** page, select **Databases** and then select **Azure Cache for Redis**.

New

Search the Marketplace

Azure Marketplace See all

Get started
Recently created
AI + Machine Learning
Analytics
Blockchain
Compute
Containers
Databases
Developer Tools
DevOps
Identity
Integration
Internet of Things
Media
Mixed Reality
IT & Management Tools
Networking
Software as a Service (SaaS)
Security
Storage
Web

Featured See all

 Azure SQL Managed Instance
[Quickstart tutorial](#)

 SQL Database
[Quickstart tutorial](#)

 SQL Data Warehouse
[Quickstart tutorial](#)

 Azure Database for MariaDB
[Learn more](#)

 **Couchbase Enterprise Edition (Hourly Pricing) (preview)**
PREVIEW
[Learn more](#)

 Azure Database for MySQL
[Quickstart tutorial](#)

 Azure Database for PostgreSQL
[Quickstart tutorial](#)

 Azure Cosmos DB
[Quickstart tutorial](#)

 SQL Server 2017 Enterprise Windows Server 2016
[Learn more](#)

 **Azure Cache for Redis**
[Quickstart tutorial](#)

- On the **New Redis Cache** page, configure the settings for your new cache.

SETTING	SUGGESTED VALUE	DESCRIPTION
DNS name	Enter a globally unique name.	The cache name must be a string between 1 and 63 characters that contains only numbers, letters, or hyphens. The name must start and end with a number or letter, and can't contain consecutive hyphens. Your cache instance's <i>host name</i> will be < <i>DNS name</i> >.redis.cache.windows.net.
Subscription	Drop down and select your subscription.	The subscription under which to create this new Azure Cache for Redis instance.

SETTING	SUGGESTED VALUE	DESCRIPTION
Resource group	Drop down and select a resource group, or select Create new and enter a new resource group name.	Name for the resource group in which to create your cache and other resources. By putting all your app resources in one resource group, you can easily manage or delete them together.
Location	Drop down and select a location.	Select a region near other services that will use your cache.
Pricing tier	Drop down and select a Pricing tier .	The pricing tier determines the size, performance, and features that are available for the cache. For more information, see Azure Cache for Redis Overview .

4. Select **Create**.

New Redis Cache

DNS name *
contoso-tmp .redis.cache.windows.net

Subscription *
A Subscription

Resource group * ⓘ
(New) contoso-rg-tmp
[Create new](#)

Location *
Central US

Pricing tier ([View full pricing details](#)) *
Basic C0 (250 MB Cache)

(PREVIEW) Availability zone
Requires Premium tier

Redis Cluster ⓘ
Requires Premium tier >

Data persistence ⓘ
Requires Premium tier >

Virtual Network ⓘ
Requires Premium tier >

Unblock port 6379 (not SSL encrypted)

Create [Automation options](#)

It takes a while for the cache to create. You can monitor progress on the Azure Cache for Redis [Overview](#) page. When **Status** shows as **Running**, the cache is ready to use.

contoso-tmp
Azure Cache for Redis

Search (Ctrl+ /)

Console Move Delete

Resource group (change)
contoso-rg-tmp

Status
Running - Basic 250 MB

Location
Central US

Subscription (change)
A Subscription

Subscription ID
12345678-1234-1234-1234-123456781234

Host name
contoso-tmp.redis.cache.windows.net

Ports
Non-SSL port (6379) disabled

Keys
Show access keys...

Best practices
<https://aka.ms/redis/p/bestpractices>

New features
<https://aka.ms/newfeatures>

Retrieve host name, ports, and access keys from the Azure portal

To connect to an Azure Cache for Redis instance, cache clients need the host name, ports, and a key for the cache. Some clients might refer to these items by slightly different names. You can get the host name, ports, and keys from the [Azure portal](#).

- To get the access keys, from your cache left navigation, select **Access keys**.

contoso-tmp - Access keys
Azure Cache for Redis

Search (Ctrl+ /)

Regenerate Primary Regenerate Secondary

Primary **Copy to clipboard**
d2Y2ba56GvC2kEM3NmOrKuctu+emE1A1A1A1A1A1A1A1A=

Secondary
5MLIRS9d56Lgt+61AjWU9r6KJsFwS6691A1A1A1A1A1A1A1A=

Primary connection string (StackExchange.Redis)
contoso-tmp.redis.cache.windows.net:6380,password=d2Y2ba56GvC2kEM3NmOrKuctu+emE...

Secondary connection string (StackExchange.Redis)
contoso-tmp.redis.cache.windows.net:6380,password=5MLIRS9d56Lgt+61AjWU9r6KJsFwS669...

For information on other clients see:
<http://aka.ms/redisclients>

- To get the host name and ports, from your cache left navigation, select **Properties**. The host name is of the form <DNS name>.redis.cache.windows.net.

The screenshot shows the Azure Cache for Redis properties page for a resource named 'contoso-tmp'. On the left, there's a sidebar with various settings like Access keys, Advanced settings, Scale, Cluster size, Data persistence, Schedule updates, Geo-replication, Virtual Network, and Firewall. The 'Properties' option is selected and highlighted with a red box. The main pane displays the status as 'Running', host name as 'contoso-tmp.redis.cache.windows.net' (with a 'Copy to clipboard' button), non-SSL port as 'Disabled', SSL port as '6380', and location as 'Central US'.

Add environment variables for your **HOST NAME** and **Primary** access key. You will use these variables from your code instead of including the sensitive information directly in your code.

```
set REDISCACHEHOSTNAME=contosoCache.redis.cache.windows.net
set REDISCACHEKEY=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Create a new Java app

Using Maven, generate a new quickstart app:

```
mvn archetype:generate -DarchetypeGroupId=org.apache.maven.archetypes -DarchetypeArtifactId=maven-archetype-quickstart -DarchetypeVersion=1.3 -DgroupId=example.demo -DartifactId=redistest -Dversion=1.0
```

Change to the new *redistest* project directory.

Open the *pom.xml* file and add a dependency for [Jedis](#):

```
<dependency>
  <groupId>redis.clients</groupId>
  <artifactId>jedis</artifactId>
  <version>2.9.0</version>
  <type>jar</type>
  <scope>compile</scope>
</dependency>
```

Save the *pom.xml* file.

Open *App.java* and replace the code with the following code:

```

package example.demo;

import redis.clients.jedis.Jedis;
import redis.clients.jedis.JedisShardInfo;

/**
 * Redis test
 *
 */
public class App
{
    public static void main( String[] args )
    {

        boolean useSsl = true;
        String cacheHostname = System.getenv("REDISCACHEHOSTNAME");
        String cachekey = System.getenv("REDISCACHEKEY");

        // Connect to the Azure Cache for Redis over the SSL port using the key.
        JedisShardInfo shardInfo = new JedisShardInfo(cacheHostname, 6380, useSsl);
        shardInfo.setPassword(cachekey); /* Use your access key. */
        Jedis jedis = new Jedis(shardInfo);

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

        // Simple PING command
        System.out.println( "\nCache Command : Ping" );
        System.out.println( "Cache Response : " + jedis.ping());

        // Simple get and put of integral data types into the cache
        System.out.println( "\nCache Command : GET Message" );
        System.out.println( "Cache Response : " + jedis.get("Message"));

        System.out.println( "\nCache Command : SET Message" );
        System.out.println( "Cache Response : " + jedis.set("Message", "Hello! The cache is working from Java!"));

        // Demonstrate "SET Message" executed as expected...
        System.out.println( "\nCache Command : GET Message" );
        System.out.println( "Cache Response : " + jedis.get("Message"));

        // Get the client list, useful to see if connection list is growing...
        System.out.println( "\nCache Command : CLIENT LIST" );
        System.out.println( "Cache Response : " + jedis.clientList());

        jedis.close();
    }
}

```

This code shows you how to connect to an Azure Cache for Redis instance using the cache host name and key environment variables. The code also stores and retrieves a string value in the cache. The **PING** and **CLIENT LIST** commands are also executed.

Save *App.java*.

Build and run the app

Execute the following Maven command to build and run the app:

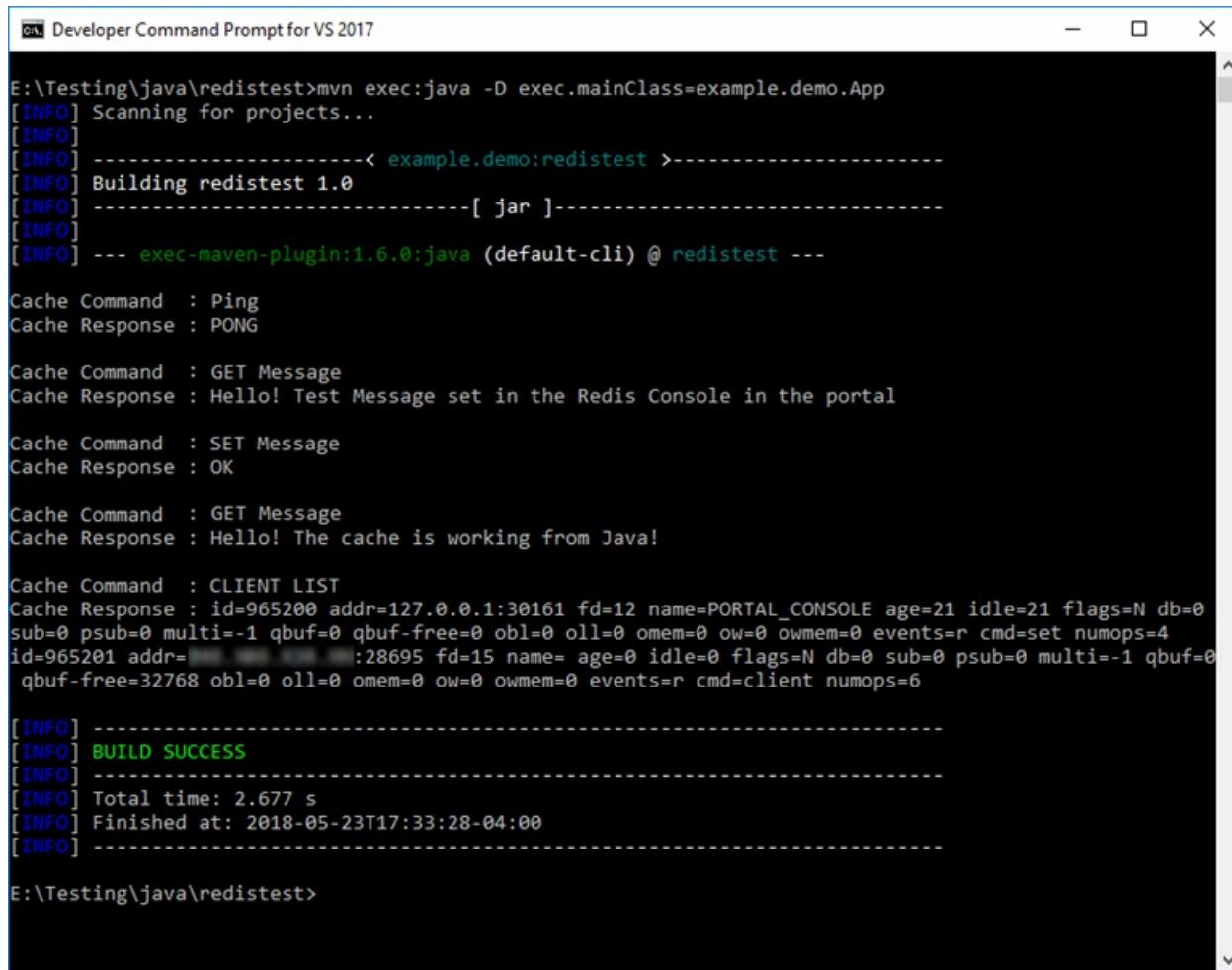
```

mvn compile
mvn exec:java -D exec.mainClass=example.demo.App

```

In the example below, you can see the **Message** key previously had a cached value, which was set using the Redis

Console in the Azure portal. The app updated that cached value. The app also executed the **PING** and **CLIENT LIST** commands.



The screenshot shows a terminal window titled "Developer Command Prompt for VS 2017". The command entered was "mvn exec:java -D exec.mainClass=example.demo.App". The output shows the application running, performing a "PING" command which returns "PONG", then a "GET Message" command which returns "Hello! Test Message set in the Redis Console in the portal", followed by a "SET Message" command which returns "OK", another "GET Message" command which returns "Hello! The cache is working from Java!", and finally a "CLIENT LIST" command which lists two Redis connections. The build concludes with "BUILD SUCCESS".

```
E:\Testing\java\redistest>mvn exec:java -D exec.mainClass=example.demo.App
[INFO] Scanning for projects...
[INFO]
[INFO] -----< example.demo:redistest >-----
[INFO] Building redistest 1.0
[INFO] -----[ jar ]-----
[INFO]
[INFO] --- exec-maven-plugin:1.6.0:java (default-cli) @ redistest ---
Cache Command : Ping
Cache Response : PONG

Cache Command : GET Message
Cache Response : Hello! Test Message set in the Redis Console in the portal

Cache Command : SET Message
Cache Response : OK

Cache Command : GET Message
Cache Response : Hello! The cache is working from Java!

Cache Command : CLIENT LIST
Cache Response : id=965200 addr=127.0.0.1:30161 fd=12 name=PORTAL_CONSOLE age=21 idle=21 flags=N db=0
sub=0 psub=0 multi=-1 qbuf=0 qbuf-free=0 obl=0 oll=0 omem=0 ow=0 owmem=0 events=r cmd=set numops=4
id=965201 addr=127.0.0.1:28695 fd=15 name=PORTAL_CONSOLE age=0 idle=0 flags=N db=0 sub=0 psub=0 multi=-1 qbuf=0
qbuf-free=32768 obl=0 oll=0 omem=0 ow=0 owmem=0 events=r cmd=client numops=6

[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 2.677 s
[INFO] Finished at: 2018-05-23T17:33:28-04:00
[INFO] -----
```

E:\Testing\java\redistest>

Clean up resources

If you will be continuing to the next tutorial, you can keep the resources created in this quickstart and reuse them.

Otherwise, if you are finished with the quickstart sample application, you can delete the Azure resources created in this quickstart to avoid charges.

IMPORTANT

Deleting a resource group is irreversible and that the resource group and all the resources in it are permanently deleted.

Make sure that you do not accidentally delete the wrong resource group or resources. If you created the resources for hosting this sample inside an existing resource group that contains resources you want to keep, you can delete each resource individually from their respective blades instead of deleting the resource group.

1. Sign in to the [Azure portal](#) and select **Resource groups**.
2. In the **Filter by name** textbox, type the name of your resource group. The instructions for this article used a resource group named *TestResources*. On your resource group in the result list, select ... then **Delete resource group**.

The screenshot shows the Microsoft Azure portal interface. On the left sidebar, under 'FAVORITES', the 'Resource groups' option is highlighted with a red box. The main content area is titled 'Resource groups' and shows a table of resources. At the top of the table, there are filters: 'Subscriptions: 1 of 7 selected - Don't see a subscription? Switch directories', 'TestResources' (selected), 'Prototype3', 'All locations', and 'No grouping'. The table has columns: NAME, SUBSCRIPTION, and LOCATION. One item is listed: 'TestResources' under 'Prototype3' in the 'All locations' group. To the right of this item is a context menu with options: 'Delete resource group' (highlighted with a red box) and '...'. The overall interface is dark-themed.

3. You will be asked to confirm the deletion of the resource group. Type the name of your resource group to confirm, and select **Delete**.

After a few moments, the resource group and all of its contained resources are deleted.

Next steps

In this quickstart, you learned how to use Azure Cache for Redis from a Java application. Continue to the next quickstart to use Azure Cache for Redis with an ASP.NET web app.

[Create an ASP.NET web app that uses an Azure Cache for Redis.](#)

Quickstart: Use Azure Cache for Redis with Node.js

11/15/2019 • 5 minutes to read • [Edit Online](#)

In this quickstart, you incorporate Azure Cache for Redis into a Node.js app to have access to a secure, dedicated cache that is accessible from any application within Azure.

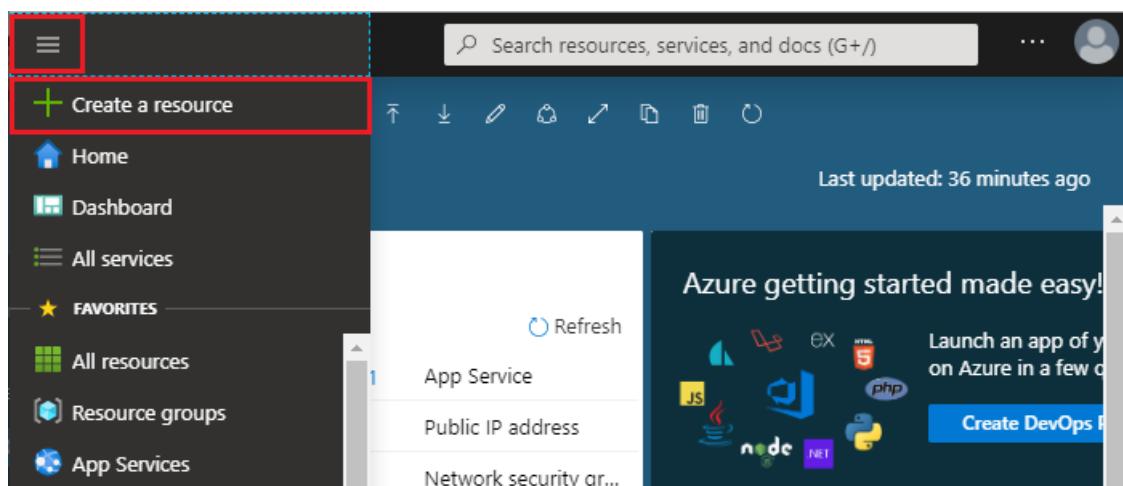
Prerequisites

- Azure subscription - [create one for free](#)
- [node_redis](#), which you can install with the command `npm install redis`.

For examples of using other Node.js clients, see the individual documentation for the Node.js clients listed at [Node.js Redis clients](#).

Create a cache

1. To create a cache, sign in to the [Azure portal](#) and select **Create a resource**.



2. On the **New** page, select **Databases** and then select **Azure Cache for Redis**.

New

Search the Marketplace

Azure Marketplace See all

Get started
Recently created
AI + Machine Learning
Analytics
Blockchain
Compute
Containers
Databases
Developer Tools
DevOps
Identity
Integration
Internet of Things
Media
Mixed Reality
IT & Management Tools
Networking
Software as a Service (SaaS)
Security
Storage
Web

Featured See all

 Azure SQL Managed Instance
[Quickstart tutorial](#)

 SQL Database
[Quickstart tutorial](#)

 SQL Data Warehouse
[Quickstart tutorial](#)

 Azure Database for MariaDB
[Learn more](#)

 **Couchbase Enterprise Edition (Hourly Pricing) (preview)**
PREVIEW
[Learn more](#)

 Azure Database for MySQL
[Quickstart tutorial](#)

 Azure Database for PostgreSQL
[Quickstart tutorial](#)

 Azure Cosmos DB
[Quickstart tutorial](#)

 SQL Server 2017 Enterprise Windows Server 2016
[Learn more](#)

 **Azure Cache for Redis**
[Quickstart tutorial](#)

- On the **New Redis Cache** page, configure the settings for your new cache.

SETTING	SUGGESTED VALUE	DESCRIPTION
DNS name	Enter a globally unique name.	The cache name must be a string between 1 and 63 characters that contains only numbers, letters, or hyphens. The name must start and end with a number or letter, and can't contain consecutive hyphens. Your cache instance's <i>host name</i> will be < <i>DNS name</i> >.redis.cache.windows.net.
Subscription	Drop down and select your subscription.	The subscription under which to create this new Azure Cache for Redis instance.

SETTING	SUGGESTED VALUE	DESCRIPTION
Resource group	Drop down and select a resource group, or select Create new and enter a new resource group name.	Name for the resource group in which to create your cache and other resources. By putting all your app resources in one resource group, you can easily manage or delete them together.
Location	Drop down and select a location.	Select a region near other services that will use your cache.
Pricing tier	Drop down and select a Pricing tier .	The pricing tier determines the size, performance, and features that are available for the cache. For more information, see Azure Cache for Redis Overview .

4. Select **Create**.

New Redis Cache

DNS name *
contoso-tmp .redis.cache.windows.net

Subscription *
A Subscription

Resource group * ⓘ
(New) contoso-rg-tmp
[Create new](#)

Location *
Central US

Pricing tier ([View full pricing details](#)) *
Basic C0 (250 MB Cache)

(PREVIEW) Availability zone
Requires Premium tier

Redis Cluster ⓘ
Requires Premium tier >

Data persistence ⓘ
Requires Premium tier >

Virtual Network ⓘ
Requires Premium tier >

Unblock port 6379 (not SSL encrypted)

Create [Automation options](#)

It takes a while for the cache to create. You can monitor progress on the Azure Cache for Redis [Overview](#) page. When **Status** shows as **Running**, the cache is ready to use.

The screenshot shows the Azure portal interface for the 'contoso-tmp' resource. The left sidebar contains navigation links: Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Settings, and Access keys. The main content area displays the following details:

Resource group (change) contoso-rg-tmp	
Status	Running - Basic 250 MB
Location	Central US
Subscription (change)	A Subscription
Subscription ID	12345678-1234-1234-1234-123456781234
Host name	contoso-tmp.redis.cache.windows.net
Ports	Non-SSL port (6379) disabled
Keys	Show access keys...
Best practices	https://aka.ms/redis/p/bestpractices
New features	https://aka.ms/newfeatures

Retrieve host name, ports, and access keys from the Azure portal

To connect to an Azure Cache for Redis instance, cache clients need the host name, ports, and a key for the cache. Some clients might refer to these items by slightly different names. You can get the host name, ports, and keys from the [Azure portal](#).

- To get the access keys, from your cache left navigation, select **Access keys**.

contoso-tmp - Access keys

Azure Cache for Redis

Search (Ctrl+ /)

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

Settings

Access keys

Regenerate Primary Regenerate Secondary

Primary

d2Y2ba56GvC2kEM3NmOrKuctu+emE1A1A1A1A1A1A1A1A=

Copy to clipboard

Secondary

5MLIRS9d56Lgt+61AjWU9r6KJsFwS6691A1A1A1A1A1A=

Primary connection string (StackExchange.Redis)

contoso-tmp.redis.cache.windows.net:6380,password=d2Y2ba56GvC2kEM3NmOrKuctu+emE...

Secondary connection string (StackExchange.Redis)

contoso-tmp.redis.cache.windows.net:6380,password=5MLIRS9d56Lgt+61AjWU9r6KJsFwS669...

For information on other clients see:
<http://aka.ms/redisclients>

- To get the host name and ports, from your cache left navigation, select **Properties**. The host name is of the form <DNS name>.redis.cache.windows.net.

The screenshot shows the Azure Cache for Redis Properties page for a cache named "contoso-tmp". The left sidebar lists settings like Access keys, Advanced settings, Scale, Cluster size, Data persistence, Schedule updates, Geo-replication, Virtual Network, and Firewall. The "Properties" section is highlighted with a red border. On the right, the status is shown as "Running". The "Host name" field contains "contoso-tmp.redis.cache.windows.net" with a "Copy to clipboard" button. The "Non-SSL Port" is set to "Disabled" and the "SSL Port" is set to "6380". The location is listed as "Central US".

Add environment variables for your **HOST NAME** and **Primary** access key. You will use these variables from your code instead of including the sensitive information directly in your code.

```
set REDISCACHEHOSTNAME=contosoCache.redis.cache.windows.net
set REDISCACHEKEY=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Connect to the cache

The latest builds of [node_redis](#) provide support for connecting to Azure Cache for Redis using SSL. The following example shows how to connect to Azure Cache for Redis using the SSL endpoint of 6380.

```
var redis = require("redis");

// Add your cache name and access key.
var client = redis.createClient(6380, process.env.REDISCACHEHOSTNAME,
    {auth_pass: process.env.REDISCACHEKEY, tls: {servername: process.env.REDISCACHEHOSTNAME}});
```

Don't create a new connections for each operation in your code. Instead, reuse connections as much as possible.

Create a new Node.js app

Create a new script file named *redistest.js*.

Add the following example JavaScript to the file. This code shows you how to connect to an Azure Cache for Redis instance using the cache host name and key environment variables. The code also stores and retrieves a string value in the cache. The `PING` and `CLIENT LIST` commands are also executed. For more examples of using Redis with the [node_redis](#) client, see <https://redis.js.org/>.

```

var redis = require("redis");
var bluebird = require("bluebird");

bluebird.promisifyAll(redis.RedisClient.prototype);
bluebird.promisifyAll(redis.Multi.prototype);

async function testCache() {

    // Connect to the Azure Cache for Redis over the SSL port using the key.
    var cacheConnection = redis.createClient(6380, process.env.REDISCACHEHOSTNAME,
        {auth_pass: process.env.REDISCACHEKEY, tls: {servername: process.env.REDISCACHEHOSTNAME}});

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

    // Simple PING command
    console.log("\nCache command: PING");
    console.log("Cache response : " + await cacheConnection.pingAsync());

    // Simple get and put of integral data types into the cache
    console.log("\nCache command: GET Message");
    console.log("Cache response : " + await cacheConnection.getAsync("Message"));

    console.log("\nCache command: SET Message");
    console.log("Cache response : " + await cacheConnection.setAsync("Message",
        "Hello! The cache is working from Node.js!"));

    // Demonstrate "SET Message" executed as expected...
    console.log("\nCache command: GET Message");
    console.log("Cache response : " + await cacheConnection.getAsync("Message"));

    // Get the client list, useful to see if connection list is growing...
    console.log("\nCache command: CLIENT LIST");
    console.log("Cache response : " + await cacheConnection.clientAsync("LIST"));
}

testCache();

```

Run the script with Node.js.

```
node redistest.js
```

In the example below, you can see the `Message` key previously had a cached value, which was set using the Redis Console in the Azure portal. The app updated that cached value. The app also executed the `PING` and `CLIENT LIST` commands.

```
Developer Command Prompt for VS 2017 - node redistest.js

E:\Testing\node>node redistest.js

Cache command: PING
Cache response : PONG

Cache command: GET Message
Cache response : Hello! Test Message set in the Redis Console in this portal

Cache command: SET Message
Cache response : OK

Cache command: GET Message
Cache response : Hello! The cache is working from Node.js!

Cache command: CLIENT LIST
Cache response : id=881265 addr=127.0.0.1:18102 fd=21 name=PORTAL_CONSOLE age=219 idle=7 flags=N db=0
sub=0 psub=0 multi=-1 qbuf=0 qbuf-free=0 obl=0 oll=0 omem=0 ow=0 owmem=0 events=r cmd=set numops=5
id=881383 addr=[REDACTED]:52827 fd=17 name= PORTAL_CONSOLE age=0 idle=0 flags=N db=0 sub=0 psub=0 multi=-1 qbuf=0
qbuf-free=32768 obl=0 oll=0 omem=0 ow=0 owmem=0 events=r cmd=client numops=7
```

Clean up resources

If you will be continuing to the next tutorial, you can keep the resources created in this quickstart and reuse them.

Otherwise, if you are finished with the quickstart sample application, you can delete the Azure resources created in this quickstart to avoid charges.

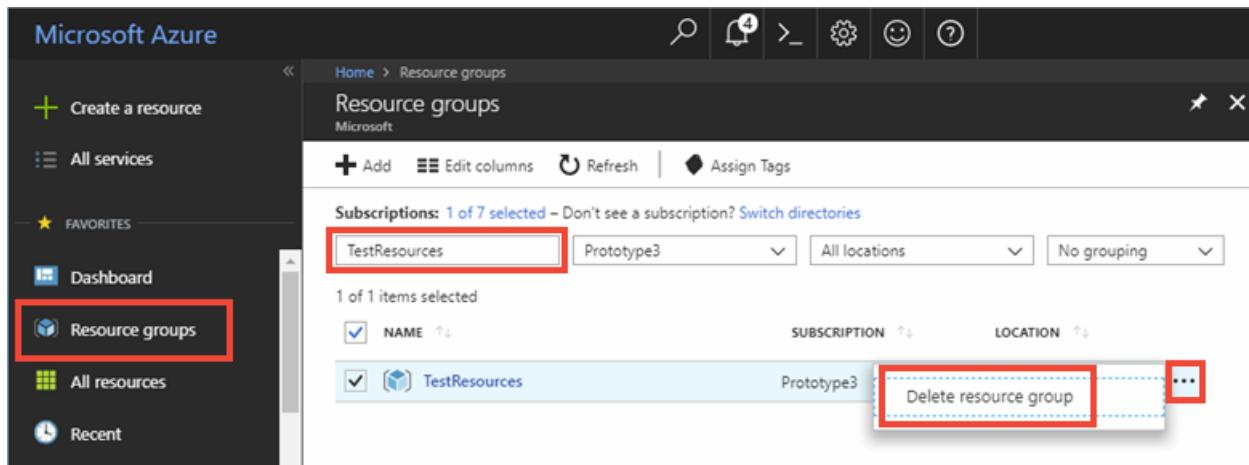
IMPORTANT

Deleting a resource group is irreversible and that the resource group and all the resources in it are permanently deleted.

Make sure that you do not accidentally delete the wrong resource group or resources. If you created the resources for hosting this sample inside an existing resource group that contains resources you want to keep, you can delete each resource individually from their respective blades instead of deleting the resource group.

Sign in to the [Azure portal](#) and select **Resource groups**.

In the **Filter by name** text box, enter the name of your resource group. The instructions for this article used a resource group named *TestResources*. On your resource group in the result list, select ... then **Delete resource group**.



You will be asked to confirm the deletion of the resource group. Enter the name of your resource group to confirm, and select **Delete**.

After a few moments, the resource group and all of its contained resources are deleted.

Next steps

In this quickstart, you learned how to use Azure Cache for Redis from a Node.js application. Continue to the next quickstart to use Azure Cache for Redis with an ASP.NET web app.

[Create an ASP.NET web app that uses an Azure Cache for Redis.](#)

Quickstart: Create a Python app that uses Azure Cache for Redis

12/10/2019 • 4 minutes to read • [Edit Online](#)

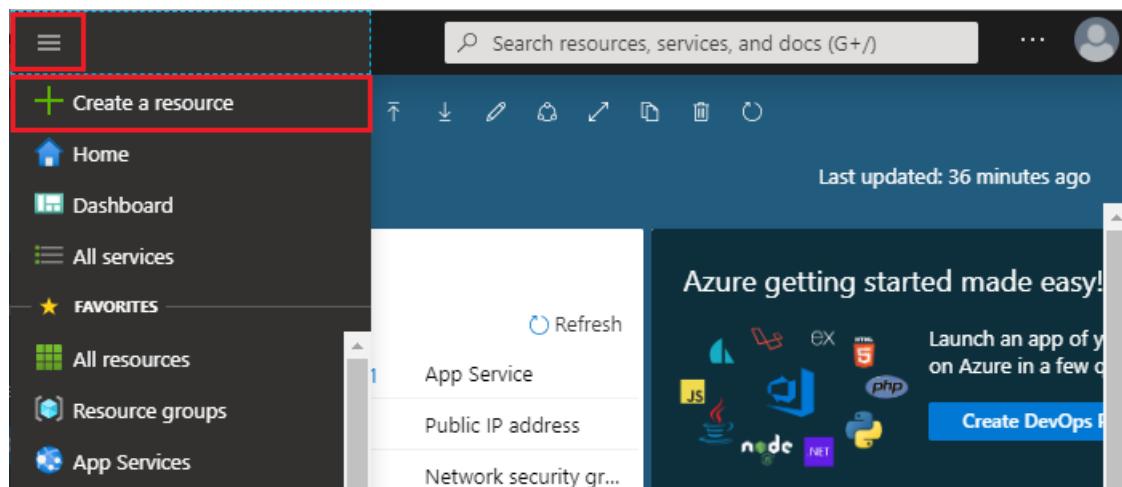
In this article, you incorporate Azure Cache for Redis into a Python app to have access to a secure, dedicated cache that is accessible from any application within Azure.

Prerequisites

- Azure subscription - [create one for free](#)
- [Python 2 or 3](#)

Create an Azure Cache for Redis instance

1. To create a cache, sign in to the [Azure portal](#) and select **Create a resource**.



2. On the **New** page, select **Databases** and then select **Azure Cache for Redis**.

New

Search the Marketplace

Azure Marketplace [See all](#)

Get started
Recently created
AI + Machine Learning
Analytics
Blockchain
Compute
Containers
Databases
Developer Tools
DevOps
Identity
Integration
Internet of Things
Media
Mixed Reality
IT & Management Tools
Networking
Software as a Service (SaaS)
Security
Storage
Web

Featured [See all](#)

	Azure SQL Managed Instance Quickstart tutorial
	SQL Database Quickstart tutorial
	SQL Data Warehouse Quickstart tutorial
	Azure Database for MariaDB Learn more
	Couchbase Enterprise Edition (Hourly Pricing) (preview) Learn more
	Azure Database for MySQL Quickstart tutorial
	Azure Database for PostgreSQL Quickstart tutorial
	Azure Cosmos DB Quickstart tutorial
	SQL Server 2017 Enterprise Windows Server 2016 Learn more
	Azure Cache for Redis Quickstart tutorial

- On the **New Redis Cache** page, configure the settings for your new cache.

SETTING	SUGGESTED VALUE	DESCRIPTION
DNS name	Enter a globally unique name.	The cache name must be a string between 1 and 63 characters that contains only numbers, letters, or hyphens. The name must start and end with a number or letter, and can't contain consecutive hyphens. Your cache instance's <i>host name</i> will be < <i>DNS name</i> >.redis.cache.windows.net.
Subscription	Drop down and select your subscription.	The subscription under which to create this new Azure Cache for Redis instance.

SETTING	SUGGESTED VALUE	DESCRIPTION
Resource group	Drop down and select a resource group, or select Create new and enter a new resource group name.	Name for the resource group in which to create your cache and other resources. By putting all your app resources in one resource group, you can easily manage or delete them together.
Location	Drop down and select a location.	Select a region near other services that will use your cache.
Pricing tier	Drop down and select a Pricing tier .	The pricing tier determines the size, performance, and features that are available for the cache. For more information, see Azure Cache for Redis Overview .

4. Select **Create**.

New Redis Cache

DNS name *

contoso-tmp.redis.cache.windows.net

Subscription *

A Subscription

Resource group * ⓘ

(New) contoso-rg-tmp

Create new

Location *

Central US

Pricing tier ([View full pricing details](#)) *

Basic C0 (250 MB Cache)

(PREVIEW) Availability zone

Requires Premium tier

Redis Cluster ⓘ

Requires Premium tier

Data persistence ⓘ

Requires Premium tier

Virtual Network ⓘ

Requires Premium tier

Unblock port 6379 (not SSL encrypted)

Create Automation options

It takes a while for the cache to create. You can monitor progress on the Azure Cache for Redis [Overview](#) page. When **Status** shows as **Running**, the cache is ready to use.

contoso-tmp
Azure Cache for Redis

Search (Ctrl+ /) <> Console Move Delete

Overview Status
Running - Basic 250 MB

Activity log Location Central US

Access control (IAM) Subscription (change)
A Subscription

Tags Subscription ID 12345678-1234-1234-1234-123456781234

Diagnose and solve problems

Settings

Access keys

Retrieve host name, ports, and access keys from the Azure portal

To connect to an Azure Cache for Redis instance, cache clients need the host name, ports, and a key for the cache. Some clients might refer to these items by slightly different names. You can get the host name, ports, and keys from the [Azure portal](#).

- To get the access keys, from your cache left navigation, select **Access keys**.

contoso-tmp - Access keys
Azure Cache for Redis

Search (Ctrl+ /) Regenerate Primary Regenerate Secondary

Primary **Copy to clipboard**
d2Y2ba56GvC2kEM3NmOrKuctu+emE1A1A1A1A1A1A1A=

Secondary
5MLIRS9d56Lgt+61AjWU9r6KJsFwS6691A1A1A1A1A1A=

Primary connection string (StackExchange.Redis)
contoso-tmp.redis.cache.windows.net:6380,password=d2Y2ba56GvC2kEM3NmOrKuctu+emE...

Secondary connection string (StackExchange.Redis)
contoso-tmp.redis.cache.windows.net:6380,password=5MLIRS9d56Lgt+61AjWU9r6KJsFwS669...

For information on other clients see:
<http://aka.ms/redisclients>

- To get the host name and ports, from your cache left navigation, select **Properties**. The host name is of the form <DNS name>.redis.cache.windows.net.

contoso-tmp - Properties

Azure Cache for Redis

Settings

- Access keys
- Advanced settings
- Scale
- Cluster size
- Data persistence
- Schedule updates
- Geo-replication
- Virtual Network
- Firewall

Properties

Status: Running

Host name: contoso-tmp.redis.cache.windows.net Copy to clipboard

Non-SSL Port: Disabled Copy

SSL Port: 6380 Copy

Location: Central US

Install redis-py

[Redis-py](#) is a Python interface to Azure Cache for Redis. Use the Python packages tool, *pip*, to install the *redis-py* package from a command prompt.

The following example used *pip3* for Python 3 to install *redis-py* on Windows 10 from an Administrator command prompt.

```
C:\Programs\Python\Python37-32\Scripts>pip3 install redis
Collecting redis
  Downloading https://files.pythonhosted.org/packages/32/ae/28613a62eea0d53d3db3147f8715f90da07667e99baeedf1010eb400f8c0/redis-3.3.11-py2.py3-none-any.whl (66kB)
    |
    ||████████████████████| 71kB 770kB/s
Installing collected packages: redis
Successfully installed redis-3.3.11

C:\Programs\Python\Python37-32\Scripts>pip3 install redis
Requirement already satisfied: redis in c:\programs\python\local\37-32\lib\site-packages
```

Read and write to the cache

Run Python from the command line and test your cache by using the following code. Replace `<Your Host Name>` and `<your Access Key>` with the values from your Azure Cache for Redis instance. Your host name is of the form `<DNS name>.redis.cache.windows.net`.

```
>>> import redis
>>> r = redis.StrictRedis(host='<Your Host Name>',
    port=6380, db=0, password='<Your Access Key>', ssl=True)
>>> r.set('foo', 'bar')
True
>>> r.get('foo')
b'bar'
```

IMPORTANT

For Azure Cache for Redis version 3.0 or higher, SSL certificate check is enforced. `ssl_ca_certs` must be explicitly set when connecting to Azure Cache for Redis. For RedHat Linux, `ssl_ca_certs` are in the `/etc/pki/tls/certs/ca-bundle.crt` certificate module.

Create a Python sample app

Create a new text file, add the following script, and save the file as `PythonApplication1.py`. Replace `<Your Host Name>` and `<Your Access Key>` with the values from your Azure Cache for Redis instance. Your host name is of the form `<DNS name>.redis.cache.windows.net`.

```
import redis

myHostname = "<Your Host Name>"
myPassword = "<Your Access Key>

r = redis.StrictRedis(host=myHostname, port=6380,
                      password=myPassword, ssl=True)

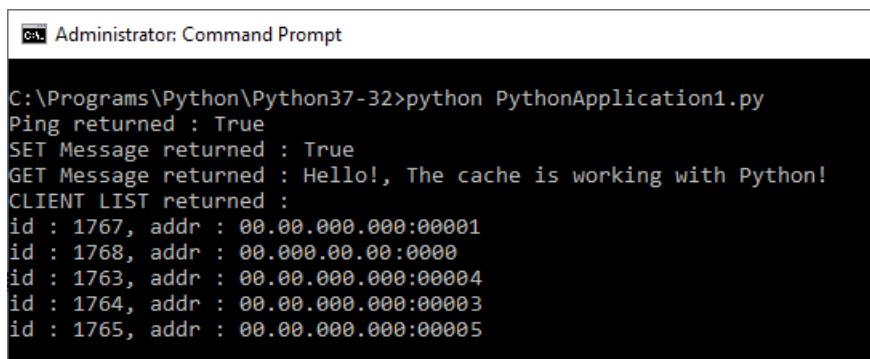
result = r.ping()
print("Ping returned : " + str(result))

result = r.set("Message", "Hello!, The cache is working with Python!")
print("SET Message returned : " + str(result))

result = r.get("Message")
print("GET Message returned : " + result.decode("utf-8"))

result = r.client_list()
print("CLIENT LIST returned : ")
for c in result:
    print("id : " + c['id'] + ", addr : " + c['addr'])
```

Run `PythonApplication1.py` with Python. You should see results like the following example:



```
C:\> Administrator: Command Prompt
C:\Programs\Python\Python37-32>python PythonApplication1.py
Ping returned : True
SET Message returned : True
GET Message returned : Hello!, The cache is working with Python!
CLIENT LIST returned :
id : 1767, addr : 00.00.000.000:00001
id : 1768, addr : 00.000.00.00:0000
id : 1763, addr : 00.00.000.000:00004
id : 1764, addr : 00.00.000.000:00003
id : 1765, addr : 00.00.000.000:00005
```

Clean up resources

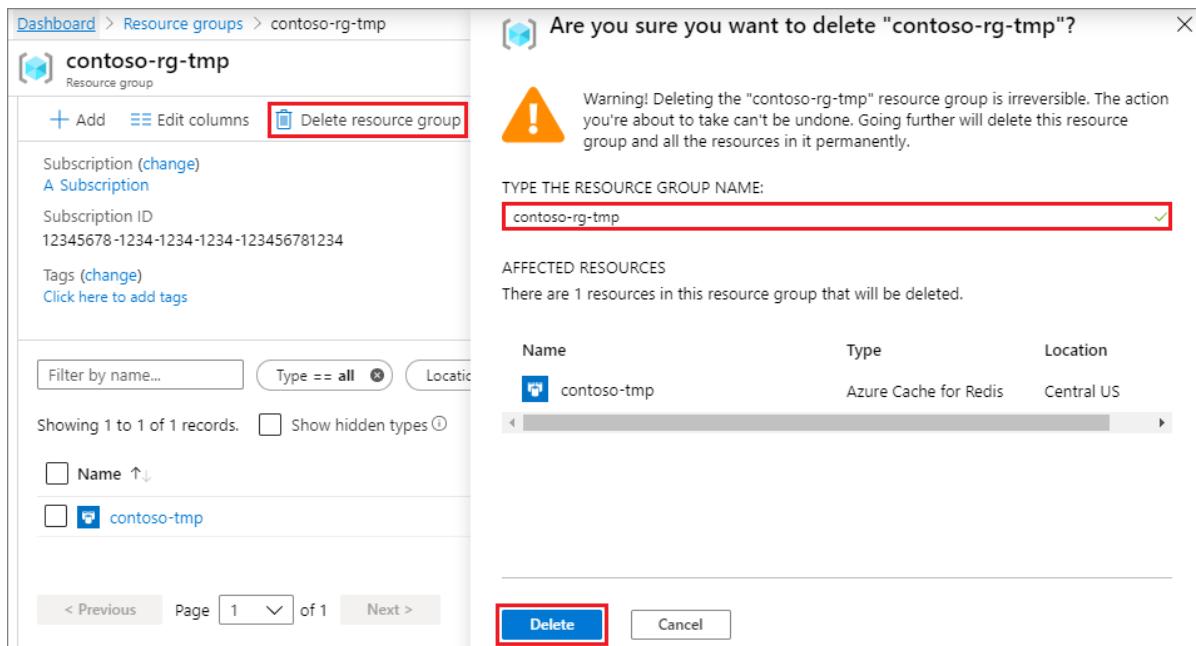
If you're finished with the Azure resource group and resources you created in this quickstart, you can delete them to avoid charges.

IMPORTANT

Deleting a resource group is irreversible, and the resource group and all the resources in it are permanently deleted. If you created your Azure Cache for Redis instance in an existing resource group that you want to keep, you can delete just the cache by selecting **Delete** from the cache **Overview** page.

To delete the resource group and its Redis Cache for Azure instance:

1. From the [Azure portal](#), search for and select **Resource groups**.
2. In the **Filter by name** text box, enter the name of the resource group that contains your cache instance, and then select it from the search results.
3. On your resource group page, select **Delete resource group**.
4. Type the resource group name, and then select **Delete**.



Next steps

[Create a simple ASP.NET web app that uses an Azure Cache for Redis.](#)

Tutorial: Create a cache-aside leaderboard on ASP.NET

12/10/2019 • 18 minutes to read • [Edit Online](#)

In this tutorial you will update the *ContosoTeamStats* ASP.NET web app, created in the [ASP.NET quickstart for Azure Cache for Redis](#), to include a leaderboard that uses the [cache-aside pattern](#) with Azure Cache for Redis. The sample application displays a list of team statistics from a database and demonstrates different ways to use Azure Cache for Redis to store and retrieve data from the cache to improve performance. When you complete the tutorial you have a running web app that reads and writes to a database, optimized with Azure Cache for Redis, and hosted in Azure.

In this tutorial, you learn how to:

- Improve data throughput and reduce database load by storing and retrieving data using Azure Cache for Redis.
- Use a Redis sorted set to retrieve the top five teams.
- Provision the Azure resources for the application using a Resource Manager template.
- Publish the application to Azure using Visual Studio.

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

Prerequisites

To complete this tutorial, you must have the following prerequisites:

- This tutorial continues where you left off in [ASP.NET quickstart for Azure Cache for Redis](#). If you haven't already, follow the quickstart first.
- Install [Visual Studio 2019](#) with the following workloads:
 - ASP.NET and web development
 - Azure Development
 - .NET desktop development with SQL Server Express LocalDB or [SQL Server 2017 Express edition](#).

Add a leaderboard to the project

In this section of the tutorial, you configure the *ContosoTeamStats* project with a leaderboard that reports the win, loss, and tie statistics for a list of fictional teams.

Add the Entity Framework to the project

1. In Visual Studio, open the *ContosoTeamStats* Solution that you created in the [ASP.NET quickstart for Azure Cache for Redis](#).
2. Click **Tools > NuGet Package Manager > Package Manager Console**.
3. Run the following command from the **Package Manager Console** window to install EntityFramework:

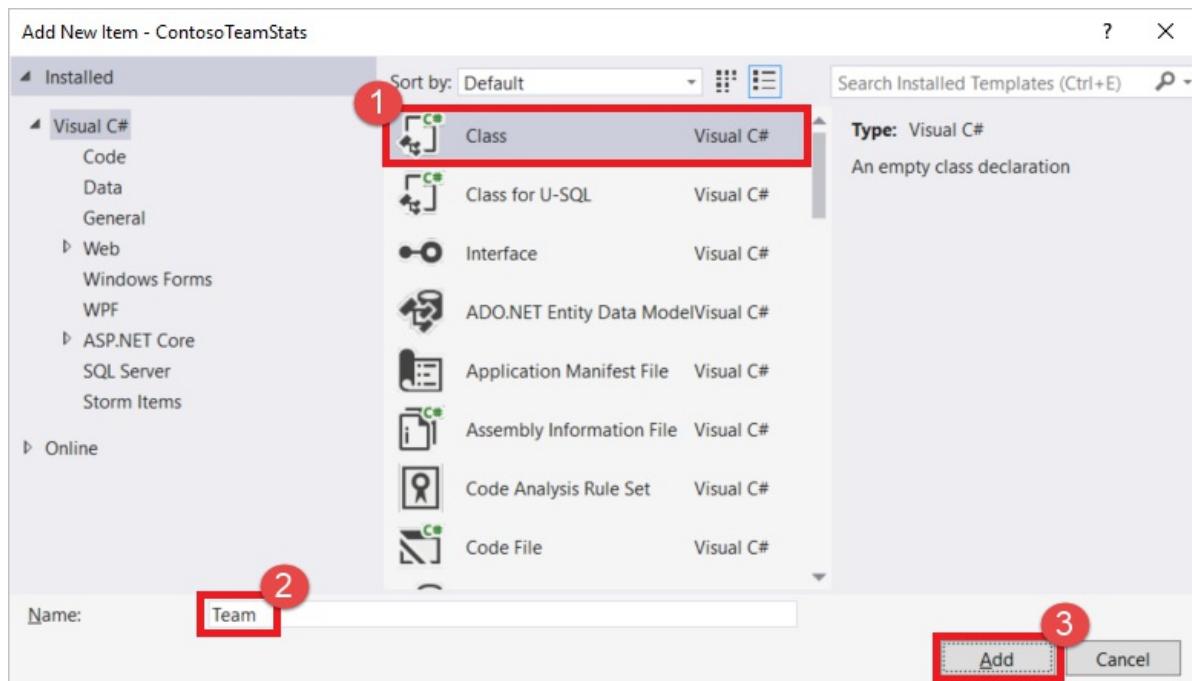
```
Install-Package EntityFramework
```

For more information about this package, see the [EntityFramework](#) NuGet page.

Add the Team model

1. Right-click **Models** in **Solution Explorer**, and choose **Add, Class**.

2. Enter `Team` for the class name and click **Add**.



3. Replace the `using` statements at the top of the `Team.cs` file with the following `using` statements:

```
using System;
using System.Collections.Generic;
using System.Data.Entity;
using System.Data.Entity.SqlServer;
```

4. Replace the definition of the `Team` class with the following code snippet that contains an updated `Team` class definition as well as some other Entity Framework helper classes. This tutorial is using the code first approach with Entity Framework. This approach allows Entity Framework to create the database from your code. For more information on the code first approach to Entity Framework that's used in this tutorial, see [Code first to a new database](#).

```

public class Team
{
    public int ID { get; set; }
    public string Name { get; set; }
    public int Wins { get; set; }
    public int Losses { get; set; }
    public int Ties { get; set; }

    static public void PlayGames(IEnumerable<Team> teams)
    {
        // Simple random generation of statistics.
        Random r = new Random();

        foreach (var t in teams)
        {
            t.Wins = r.Next(33);
            t.Losses = r.Next(33);
            t.Ties = r.Next(0, 5);
        }
    }
}

public class TeamContext : DbContext
{
    public TeamContext()
        : base("TeamContext")
    {

    }

    public DbSet<Team> Teams { get; set; }
}

public class TeamInitializer : CreateDatabaseIfNotExists<TeamContext>
{
    protected override void Seed(TeamContext context)
    {
        var teams = new List<Team>
        {
            new Team{Name="Adventure Works Cycles"},
            new Team{Name="Alpine Ski House"},
            new Team{Name="Blue Yonder Airlines"},
            new Team{Name="Coho Vineyard"},
            new Team{Name="Contoso, Ltd."},
            new Team{Name="Fabrikam, Inc."},
            new Team{Name="Lucerne Publishing"},
            new Team{Name="Northwind Traders"},
            new Team{Name="Consolidated Messenger"},
            new Team{Name="Fourth Coffee"},
            new Team{Name="Graphic Design Institute"},
            new Team{Name="Nod Publishers"}
        };

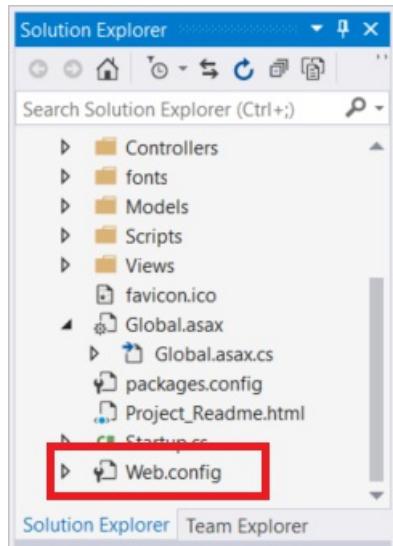
        Team.PlayGames(teams);

        teams.ForEach(t => context.Teams.Add(t));
        context.SaveChanges();
    }
}

public class TeamConfiguration : DbConfiguration
{
    public TeamConfiguration()
    {
        SetExecutionStrategy("System.Data.SqlClient", () => new SqlAzureExecutionStrategy());
    }
}

```

5. In **Solution Explorer**, double-click **Web.config** to open it.



6. Add the following `connectionStrings` section inside the `configuration` section. The name of the connection string must match the name of the Entity Framework database context class, which is `TeamContext`.

This connection string assumes you have met the [Prerequisites](#) and installed SQL Server Express LocalDB, which is part of the *.NET desktop development* workload installed with Visual Studio 2019.

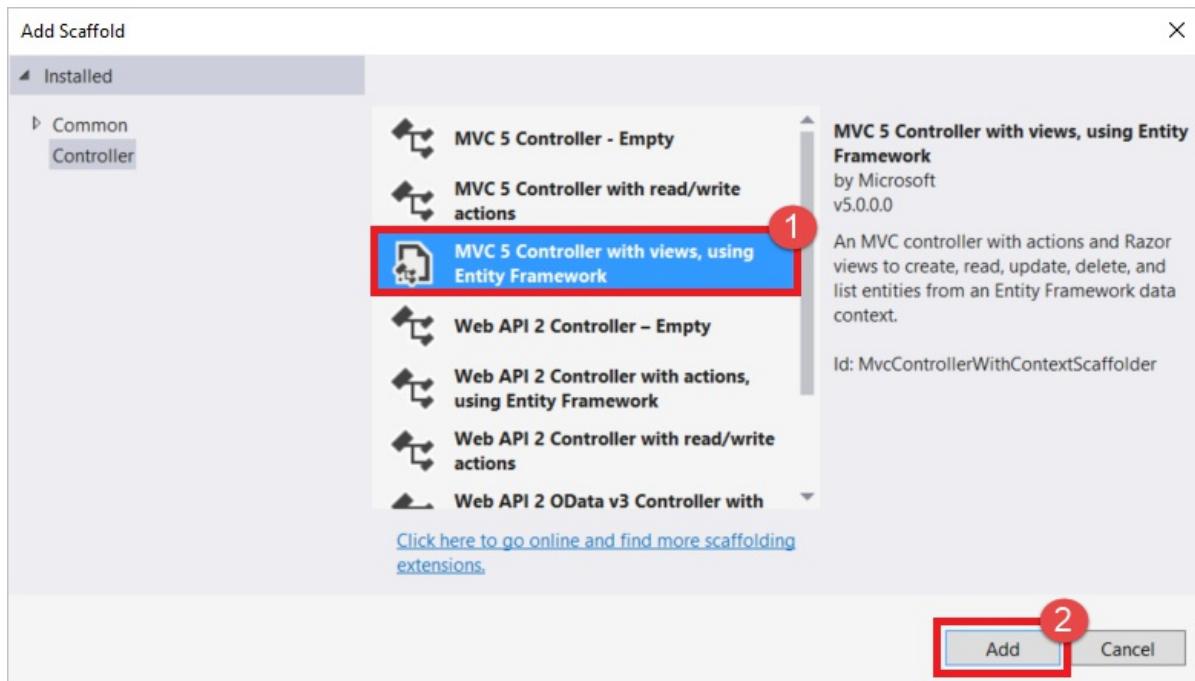
```
<connectionStrings>
  <add name="TeamContext" connectionString="Data Source=
  (LocalDB)\MSSQLLocalDB;AttachDbFilename=|DataDirectory|\Teams.mdf;Integrated Security=True"
  providerName="System.Data.SqlClient" />
</connectionStrings>
```

The following example shows the new `connectionStrings` section following `configSections` inside the `configuration` section:

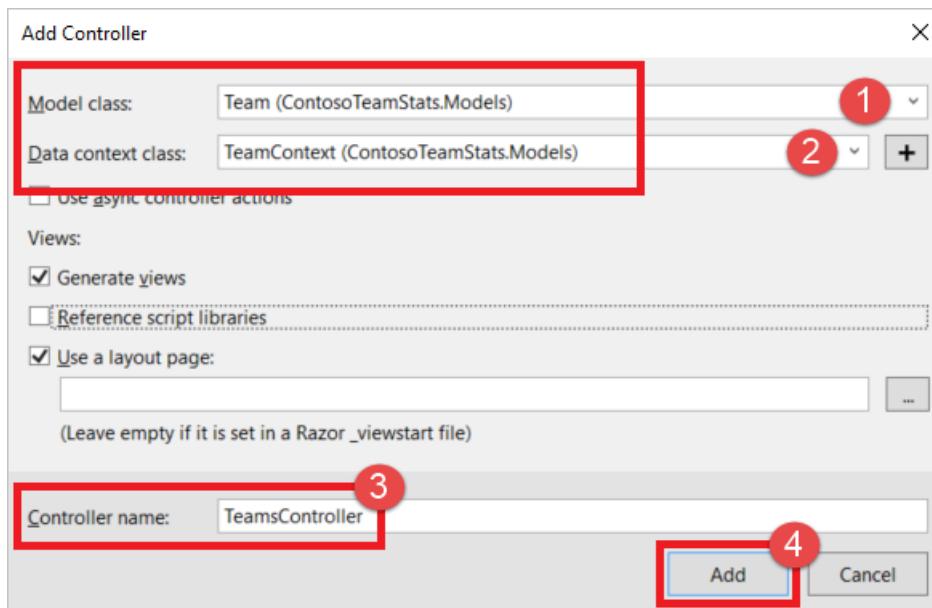
```
<configuration>
  <configSections>
    <section name="entityFramework"
    type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection, EntityFramework, Version=6.0.0.0,
    Culture=neutral, PublicKeyToken=b77a5c561934e089" requirePermission="false" />
  </configSections>
  <connectionStrings>
    <add name="TeamContext" connectionString="Data Source=
    (LocalDB)\MSSQLLocalDB;AttachDbFilename=|DataDirectory|\Teams.mdf;Integrated Security=True"
    providerName="System.Data.SqlClient" />
  </connectionStrings>
  ...
</configuration>
```

Add the TeamsController and views

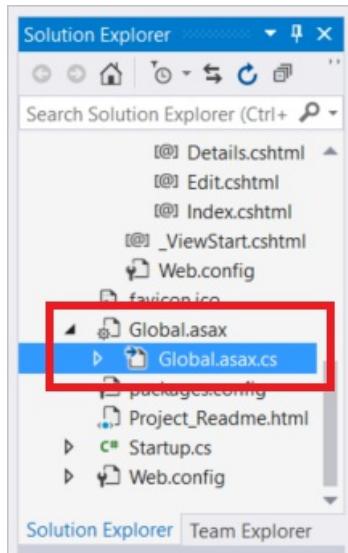
1. In Visual Studio, build the project.
2. In **Solution Explorer**, right-click the **Controllers** folder and choose **Add, Controller**.
3. Choose **MVC 5 Controller with views, using Entity Framework**, and click **Add**. If you get an error after clicking **Add**, ensure that you have built the project first.



4. Select **Team (ContosoTeamStats.Models)** from the **Model class** drop-down list. Select **TeamContext (ContosoTeamStats.Models)** from the **Data context class** drop-down list. Type `TeamsController` in the **Controller** name textbox (if it is not automatically populated). Click **Add** to create the controller class and add the default views.



5. In **Solution Explorer**, expand **Global.asax** and double-click **Global.asax.cs** to open it.



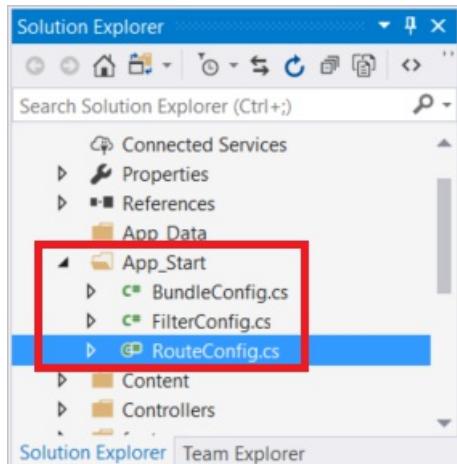
6. Add the following two `using` statements at the top of the file under the other `using` statements:

```
using System.Data.Entity;
using ContosoTeamStats.Models;
```

7. Add the following line of code at the end of the `Application_Start` method:

```
Database.SetInitializer<TeamContext>(new TeamInitializer());
```

8. In **Solution Explorer**, expand `App_Start` and double-click `RouteConfig.cs`.

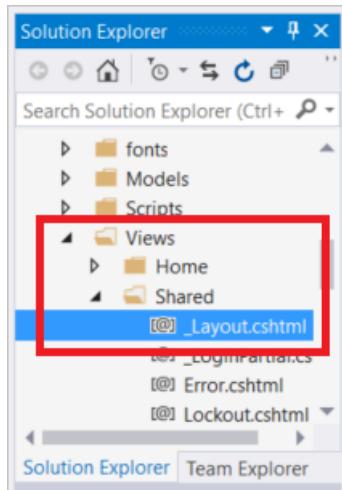


9. In the `RegisterRoutes` method, replace `controller = "Home"` in the `Default` route with `controller = "Teams"` as shown in the following example:

```
routes.MapRoute(
    name: "Default",
    url: "{controller}/{action}/{id}",
    defaults: new { controller = "Teams", action = "Index", id = UrlParameter.Optional }
);
```

Configure the Layout view

1. In **Solution Explorer**, expand the **Views** folder and then the **Shared** folder, and double-click `_Layout.cshtml`.



2. Change the contents of the `title` element and replace `My ASP.NET Application` with `Contoso Team Stats` as shown in the following example:

```
<title>@ViewBag.Title - Contoso Team Stats</title>
```

3. In the `body` section, add the following new `Html.ActionLink` statement for *Contoso Team Stats* just below the link for *Azure Cache for Redis Test*.

```
@Html.ActionLink("Contoso Team Stats", "Index", "Teams", new { area = "" }, new { @class = "navbar-brand" })`
```

The screenshot shows the code editor with the _Layout.cshtml file open. A red box highlights the `<title>@ViewBag.Title - Contoso Team Stats</title>` line. Another red box highlights the new `@Html.ActionLink("Contoso Team Stats", "Index", "Teams", new { area = "" }, new { @class = "navbar-brand" })`` line added below the existing one.

```

_Layout.cshtml > X
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <meta charset="utf-8" />
5      <meta name="viewport" content="width=device-width, initial-scale=1.0" />
6      <title>@ViewBag.Title - Contoso Team Stats</title>
7      <link href="~/Content/css" rel="stylesheet" />
8      <script src="~/bundles/modernizr" />
9  </head>
10 <body>
11     <div class="navbar navbar-inverse navbar-fixed-top">
12         <div class="container">
13             <div class="navbar-header">
14                 <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse">
15                     <span class="icon-bar"></span>
16                     <span class="icon-bar"></span>
17                     <span class="icon-bar"></span>
18                 </button>
19                 @Html.ActionLink("Azure Redis Cache Test", "RedisCache", "Home", new { area = "" }, new { @class = "navbar-brand" })
20                 @Html.ActionLink("Contoso Team Stats", "Index", "Teams", new { area = "" }, new { @class = "navbar-brand" })
21             </div>
22             <div class="navbar-collapse collapse">
23                 <ul class="nav navbar-nav">
24                     <li>@Html.ActionLink("Home", "Index", "Home")</li>
25                     <li>@Html.ActionLink("About", "About", "Home")</li>
26                 </ul>
27             </div>
28         </div>
29     </div>
30 
```

4. Press **Ctrl+F5** to build and run the application. This version of the application reads the results directly from the database. Note the **Create New**, **Edit**, **Details**, and **Delete** actions that were automatically added to the application by the **MVC 5 Controller with views, using Entity Framework** scaffold. In the next section of the tutorial, you'll add Azure Cache for Redis to optimize the data access and provide additional features to the application.

Index

[Create New](#)

Name	Wins	Losses	Ties	
Adventure Works Cycles	22	21	0	Edit Details Delete
Alpine Ski House	7	30	1	Edit Details Delete
Blue Yonder Airlines	20	9	2	Edit Details Delete
Coho Vineyard	14	3	1	Edit Details Delete
Contoso, Ltd.	29	9	1	Edit Details Delete
Fabrikam, Inc.	10	10	1	Edit Details Delete
Lucerne Publishing	11	5	2	Edit Details Delete
Northwind Traders	14	15	2	Edit Details Delete
Consolidated Messenger	23	3	0	Edit Details Delete
Fourth Coffee	32	5	3	Edit Details Delete
Graphic Design Institute	20	15	3	Edit Details Delete
Nod Publishers	29	3	1	Edit Details Delete

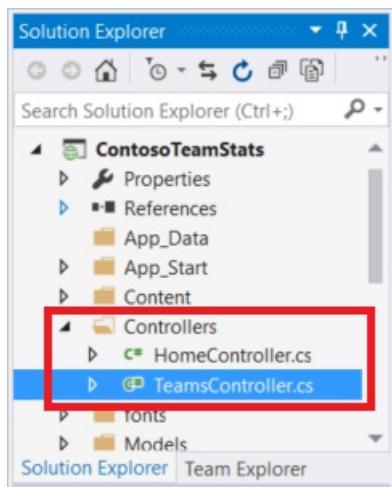
Configure the app for Azure Cache for Redis

In this section of the tutorial, you configure the sample application to store and retrieve Contoso team statistics from an Azure Cache for Redis instance by using the [StackExchange.Redis](#) cache client.

Add a cache connection to the Teams Controller

You already installed the [StackExchange.Redis](#) client library package in the quickstart. You also have already configured the `CacheConnection` app setting to be used locally, and with the published App Service. Use this same client library and `CacheConnection` information in the `TeamsController`.

1. In **Solution Explorer**, expand the **Controllers** folder and double-click **TeamsController.cs** to open it.



2. Add the following two `using` statements to **TeamsController.cs**:

```
using System.Configuration;
using StackExchange.Redis;
```

3. Add the following two properties to the `TeamsController` class:

```
// Redis Connection string info
private static Lazy<ConnectionMultiplexer> lazyConnection = new Lazy<ConnectionMultiplexer>(() =>
{
    string cacheConnection = ConfigurationManager.AppSettings["CacheConnection"].ToString();
    return ConnectionMultiplexer.Connect(cacheConnection);
});

public static ConnectionMultiplexer Connection
{
    get
    {
        return lazyConnection.Value;
    }
}
```

Update the TeamsController to read from the cache or the database

In this sample, team statistics can be retrieved from the database or from the cache. Team statistics are stored in the cache as a serialized `List<Team>`, and also as a sorted set using Redis data types. When retrieving items from a sorted set, you can retrieve some, all, or query for certain items. In this sample, you'll query the sorted set for the top 5 teams ranked by number of wins.

It is not required to store the team statistics in multiple formats in the cache in order to use Azure Cache for Redis. This tutorial uses multiple formats to demonstrate some of the different ways and different data types you can use to cache data.

1. Add the following `using` statements to the `TeamsController.cs` file at the top with the other `using` statements:

```
using System.Diagnostics;
using Newtonsoft.Json;
```

2. Replace the current `public ActionResult Index()` method implementation with the following implementation:

```

// GET: Teams
public ActionResult Index(string actionType, string resultType)
{
    List<Team> teams = null;

    switch(actionType)
    {
        case "playGames": // Play a new season of games.
            PlayGames();
            break;

        case "clearCache": // Clear the results from the cache.
            ClearCachedTeams();
            break;

        case "rebuildDB": // Rebuild the database with sample data.
            RebuildDB();
            break;
    }

    // Measure the time it takes to retrieve the results.
    Stopwatch sw = Stopwatch.StartNew();

    switch(resultType)
    {
        case "teamsSortedSet": // Retrieve teams from sorted set.
            teams = GetFromSortedSet();
            break;

        case "teamsSortedSetTop5": // Retrieve the top 5 teams from the sorted set.
            teams = GetFromSortedSetTop5();
            break;

        case "teamList": // Retrieve teams from the cached List<Team>.
            teams = GetFromList();
            break;

        case "fromDB": // Retrieve results from the database.
        default:
            teams = GetFromDB();
            break;
    }

    sw.Stop();
    double ms = sw.ElapsedTicks / (Stopwatch.Frequency / (1000.0));

    // Add the elapsed time of the operation to the ViewBag.msg.
    ViewBag.msg += " MS: " + ms.ToString();

    return View(teams);
}

```

3. Add the following three methods to the `TeamsController` class to implement the `playGames`, `clearCache`, and `rebuildDB` action types from the switch statement added in the previous code snippet.

The `PlayGames` method updates the team statistics by simulating a season of games, saves the results to the database, and clears the now outdated data from the cache.

```

void PlayGames()
{
    ViewBag.msg += "Updating team statistics. ";
    // Play a "season" of games.
    var teams = from t in db.Teams
                select t;

    Team.PlayGames(teams);

    db.SaveChanges();

    // Clear any cached results
    ClearCachedTeams();
}

```

The `RebuildDB` method reinitializes the database with the default set of teams, generates statistics for them, and clears the now outdated data from the cache.

```

void RebuildDB()
{
    ViewBag.msg += "Rebuilding DB. ";
    // Delete and re-initialize the database with sample data.
    db.Database.Delete();
    db.Database.Initialize(true);

    // Clear any cached results
    ClearCachedTeams();
}

```

The `ClearCachedTeams` method removes any cached team statistics from the cache.

```

void ClearCachedTeams()
{
    IDatabase cache = Connection.GetDatabase();
    cache.KeyDelete("teamsList");
    cache.KeyDelete("teamsSortedSet");
    ViewBag.msg += "Team data removed from cache. ";
}

```

- Add the following four methods to the `TeamsController` class to implement the various ways of retrieving the team statistics from the cache and the database. Each of these methods returns a `List<Team>`, which is then displayed by the view.

The `GetFromDB` method reads the team statistics from the database.

```

List<Team> GetFromDB()
{
    ViewBag.msg += "Results read from DB. ";
    var results = from t in db.Teams
                  orderby t.Wins descending
                  select t;

    return results.ToList<Team>();
}

```

The `GetFromList` method reads the team statistics from cache as a serialized `List<Team>`. If the statistics are not present in the cache, a cache miss occurs. For a cache miss, the team statistics are read from the database and then stored in the cache for the next request. In this sample, JSON.NET serialization is used to serialize the .NET objects to and from the cache. For more information, see [How to work with .NET objects](#)

in Azure Cache for Redis.

```
List<Team> GetFromList()
{
    List<Team> teams = null;

    IDatabase cache = Connection.GetDatabase();
    string serializedTeams = cache.StringGet("teamsList");
    if (!String.IsNullOrEmpty(serializedTeams))
    {
        teams = JsonConvert.DeserializeObject<List<Team>>(serializedTeams);

        ViewBag.msg += "List read from cache. ";
    }
    else
    {
        ViewBag.msg += "Teams list cache miss. ";
        // Get from database and store in cache
        teams = GetFromDB();

        ViewBag.msg += "Storing results to cache. ";
        cache.StringSet("teamsList", JsonConvert.SerializeObject(teams));
    }
    return teams;
}
```

The `GetFromSortedSet` method reads the team statistics from a cached sorted set. If there is a cache miss, the team statistics are read from the database and stored in the cache as a sorted set.

```
List<Team> GetFromSortedSet()
{
    List<Team> teams = null;
    IDatabase cache = Connection.GetDatabase();
    // If the key teamsSortedSet is not present, this method returns a 0 length collection.
    var teamsSortedSet = cache.SortedSetRangeByRankWithScores("teamsSortedSet", order:
Order.Descending);
    if (teamsSortedSet.Count() > 0)
    {
        ViewBag.msg += "Reading sorted set from cache. ";
        teams = new List<Team>();
        foreach (var t in teamsSortedSet)
        {
            Team tt = JsonConvert.DeserializeObject<Team>(t.Element);
            teams.Add(tt);
        }
    }
    else
    {
        ViewBag.msg += "Teams sorted set cache miss. ";

        // Read from DB
        teams = GetFromDB();

        ViewBag.msg += "Storing results to cache. ";
        foreach (var t in teams)
        {
            Console.WriteLine("Adding to sorted set: {0} - {1}", t.Name, t.Wins);
            cache.SortedSetAdd("teamsSortedSet", JsonConvert.SerializeObject(t), t.Wins);
        }
    }
    return teams;
}
```

The `GetFromSortedSetTop5` method reads the top five teams from the cached sorted set. It starts by checking

the cache for the existence of the `teamsSortedSet` key. If this key is not present, the `GetFromSortedSet` method is called to read the team statistics and store them in the cache. Next, the cached sorted set is queried for the top five teams, which are returned.

```
List<Team> GetFromSortedSetTop5()
{
    List<Team> teams = null;
    IDatabase cache = Connection.GetDatabase();

    // If the key teamsSortedSet is not present, this method returns a 0 length collection.
    var teamsSortedSet = cache.SortedSetRangeByRankWithScores("teamsSortedSet", stop: 4, order:
Order.Descending);
    if(teamsSortedSet.Count() == 0)
    {
        // Load the entire sorted set into the cache.
        GetFromSortedSet();

        // Retrieve the top 5 teams.
        teamsSortedSet = cache.SortedSetRangeByRankWithScores("teamsSortedSet", stop: 4, order:
Order.Descending);
    }

    ViewBag.msg += "Retrieving top 5 teams from cache. ";
    // Get the top 5 teams from the sorted set
    teams = new List<Team>();
    foreach (var team in teamsSortedSet)
    {
        teams.Add(JsonConvert.DeserializeObject<Team>(team.Element));
    }
    return teams;
}
```

Update the Create, Edit, and Delete methods to work with the cache

The scaffolding code that was generated as part of this sample includes methods to add, edit, and delete teams. Anytime a team is added, edited, or removed, the data in the cache becomes outdated. In this section, you'll modify these three methods to clear the cached teams so that the cache will be refreshed.

1. Browse to the `Create(Team team)` method in the `TeamsController` class. Add a call to the `ClearCachedTeams` method, as shown in the following example:

```
// POST: Teams/Create
// To protect from overposting attacks, please enable the specific properties you want to bind to, for
// more details see https://go.microsoft.com/fwlink/?LinkId=317598.
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Create([Bind(Include = "ID,Name,Wins,Losses,Ties")] Team team)
{
    if (ModelState.IsValid)
    {
        db.Teams.Add(team);
        db.SaveChanges();
        // When a team is added, the cache is out of date.
        // Clear the cached teams.
        ClearCachedTeams();
        return RedirectToAction("Index");
    }

    return View(team);
}
```

2. Browse to the `Edit(Team team)` method in the `TeamsController` class. Add a call to the `ClearCachedTeams` method, as shown in the following example:

```

// POST: Teams/Edit/5
// To protect from overposting attacks, please enable the specific properties you want to bind to, for
// more details see https://go.microsoft.com/fwlink/?LinkId=317598.
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Edit([Bind(Include = "ID,Name,Wins,Losses,Ties")] Team team)
{
    if (ModelState.IsValid)
    {
        db.Entry(team).State = EntityState.Modified;
        db.SaveChanges();
        // When a team is edited, the cache is out of date.
        // Clear the cached teams.
        ClearCachedTeams();
        return RedirectToAction("Index");
    }
    return View(team);
}

```

- Browse to the `DeleteConfirmed(int id)` method in the `TeamsController` class. Add a call to the `ClearCachedTeams` method, as shown in the following example:

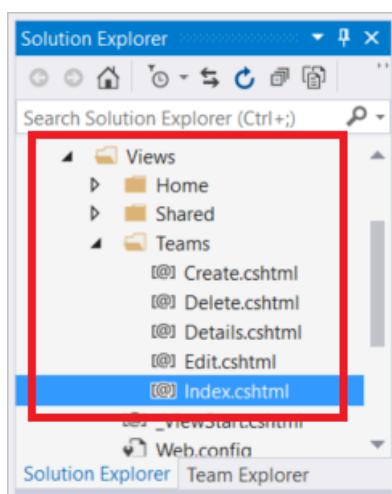
```

// POST: Teams/Delete/5
[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
public ActionResult DeleteConfirmed(int id)
{
    Team team = db.Teams.Find(id);
    db.Teams.Remove(team);
    db.SaveChanges();
    // When a team is deleted, the cache is out of date.
    // Clear the cached teams.
    ClearCachedTeams();
    return RedirectToAction("Index");
}

```

Add caching methods to the Teams Index view

- In **Solution Explorer**, expand the **Views** folder, then the **Teams** folder, and double-click **Index.cshtml**.



- Near the top of the file, look for the following paragraph element:



```

Index.cshtml ✘ X
@model IEnumerable<ContosoTeamStats.Models.Team>

 @{
    ViewBag.Title = "Index";
}

<h2>Index</h2>



@Html.ActionLink("Create New", "Create")



| @Html.DisplayNameFor(model => model.Name) | @Html.DisplayNameFor(model => model.Wins) | @Html.DisplayNameFor(model => model.Losses) | @Html.DisplayNameFor(model => model.Ties) |
|-------------------------------------------|-------------------------------------------|---------------------------------------------|-------------------------------------------|
|-------------------------------------------|-------------------------------------------|---------------------------------------------|-------------------------------------------|


```

This link creates a new team. Replace the paragraph element with the following table. This table has action links for creating a new team, playing a new season of games, clearing the cache, retrieving the teams from the cache in several formats, retrieving the teams from the database, and rebuilding the database with fresh sample data.

```

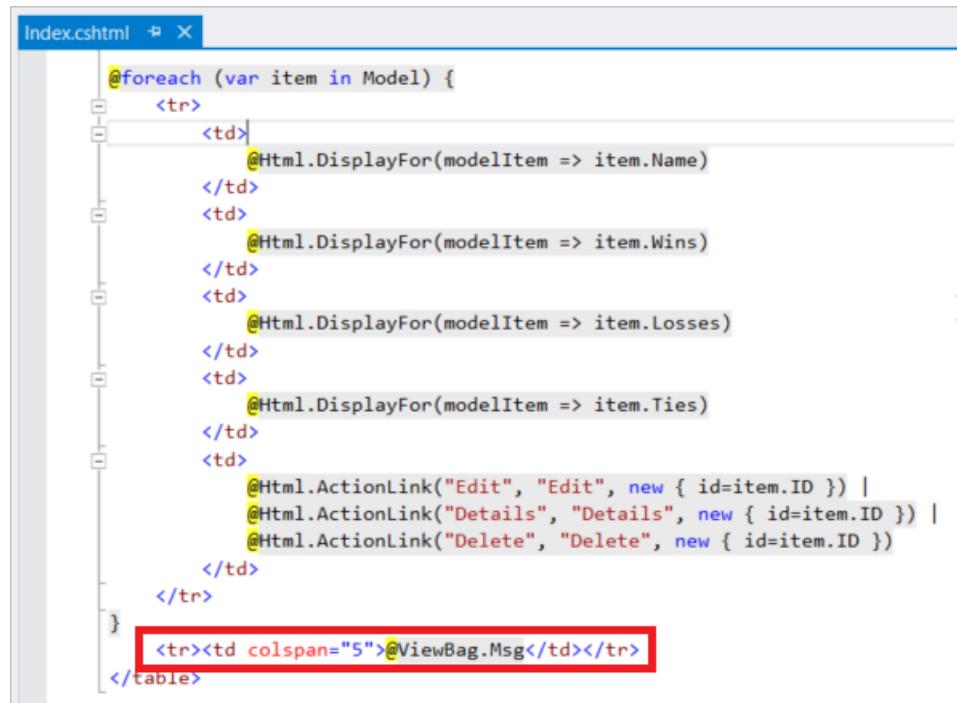

|                                          |                                                                            |                                                                             |                                                                                |                                                                                           |                                                                                                |
|------------------------------------------|----------------------------------------------------------------------------|-----------------------------------------------------------------------------|--------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------|
| @Html.ActionLink("Create New", "Create") | @Html.ActionLink("Play Season", "Index", new { actionType = "playGames" }) | @Html.ActionLink("Clear Cache", "Index", new { actionType = "clearCache" }) | @Html.ActionLink("List from Cache", "Index", new { resultType = "teamsList" }) | @Html.ActionLink("Sorted Set from Cache", "Index", new { resultType = "teamsSortedSet" }) | @Html.ActionLink("Top 5 Teams from Cache", "Index", new { resultType = "teamsSortedSetTop5" }) |
|                                          |                                                                            |                                                                             |                                                                                |                                                                                           | @Html.ActionLink("Load from DB", "Index", new { resultType = "fromDB" })                       |
|                                          |                                                                            |                                                                             |                                                                                |                                                                                           | @Html.ActionLink("Rebuild DB", "Index", new { actionType = "rebuildDB" })                      |


```

3. Scroll to the bottom of the **Index.cshtml** file and add the following `tr` element so that it is the last row in the last table in the file:

```
<tr><td colspan="5">@ViewBag.Msg</td></tr>
```

This row displays the value of `ViewBag.Msg` which contains a status report about the current operation. The `ViewBag.Msg` is set when you click any of the action links from the previous step.



```
Index.cshtml # X
@foreach (var item in Model) {
    <tr>
        <td>
            @Html.DisplayFor(modelItem => item.Name)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Wins)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Losses)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Ties)
        </td>
        <td>
            @Html.ActionLink("Edit", "Edit", new { id=item.ID }) |
            @Html.ActionLink("Details", "Details", new { id=item.ID }) |
            @Html.ActionLink("Delete", "Delete", new { id=item.ID })
        </td>
    </tr>
}
<tr><td colspan="5">@ViewBag.Msg</td></tr>
</table>
```

4. Press **F6** to build the project.

Run the app locally

Run the application locally on your machine to verify the functionality that has been added to support the teams.

In this test, the application and database, are both running locally. However, the Azure Cache for Redis is hosted remotely in Azure. Therefore, the cache will likely under-perform the database slightly. For best performance, the client application and Azure Cache for Redis instance should be in the same location. In the next section, you will deploy all resources to Azure to see the improved performance from using a cache.

To run the app locally:

1. Press **Ctrl+F5** to run the application.

2. Test each of the new methods that were added to the view. Since the cache is remote in these tests, the database should slightly outperform the cache.

Publish and run in Azure

Provision a SQL Azure database for the app

In this section, you will provision a new SQL Azure database for the app to use while hosted in Azure.

1. In the [Azure portal](#), Click **Create a resource** in the upper left-hand corner of the Azure portal.
2. On the **New** page, click **Databases > SQL Database**.
3. Use the following settings for the new SQL Database:

SETTING	SUGGESTED VALUE	DESCRIPTION
Database name	<i>ContosoTeamsDatabase</i>	For valid database names, see Database Identifiers .
Subscription	<i>Your subscription</i>	Select the same subscription you used to create the cache and host the App Service.
Resource group	<i>TestResourceGroup</i>	Click Use existing and use the same resource group where you placed your cache and App Service.

SETTING	SUGGESTED VALUE	DESCRIPTION
Select source	Blank database	Start with a blank database.

4. Under **Server**, click **Configure required settings > Create a new server** and provide the following information and then click the **Select** button:

SETTING	SUGGESTED VALUE	DESCRIPTION
Server name	Any globally unique name	For valid server names, see Naming rules and restrictions .
Server admin login	Any valid name	For valid login names, see Database Identifiers .
Password	Any valid password	Your password must have at least 8 characters and must contain characters from three of the following categories: upper case characters, lower case characters, numbers, and non-alphanumeric characters.
Location	<i>East US</i>	Select the same region where you created the cache and App Service.

5. Click **Pin to dashboard** and then **Create** to create the new database and server.
6. Once the new database is created, click **Show database connection strings** and copy the **ADO.NET** connection string.

The screenshot shows the Azure portal interface for a SQL database named 'ContosoTeamsDatabase'. The left sidebar has tabs for 'Overview', 'Activity log', and 'Tags', with 'Overview' being the active tab. The main content area displays various database properties: Resource group (WesmcTestResources), Status (Online), Location (East US), Subscription (Prototype3), Server name (wesmcitest-db-server.database.windows.net), Elastic database pool (No elastic pool), and Pricing tier (Standard S0: 10 DTUs). A red box highlights the 'Connection strings' link in the top right corner of the main content area.

7. In the Azure portal, navigate to your App Service and click **Application Settings**, then **Add new connection string** under the Connection strings section.
8. Add a new connection string named *TeamContext* to match the Entity Framework database context class. Paste the connection string for your new database as the value. Be sure to replace the following placeholders in the connection string and click **Save**:

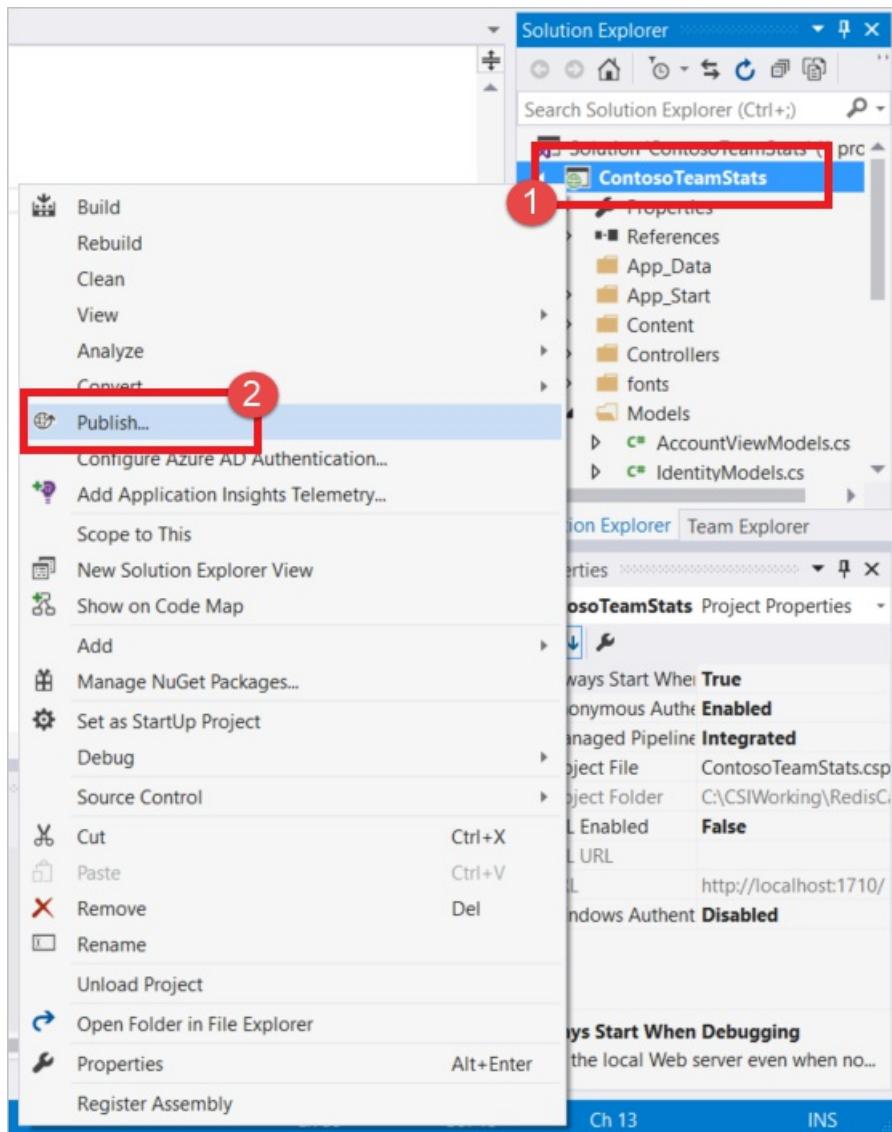
PLACEHOLDER	SUGGESTED VALUE
{your_username}	Use the server admin login for the database server you just created.
{your_password}	Use the password for the database server you just created.

By adding the username and password as an Application Setting, your username and password are not included in your code. This approach helps protect those credentials.

Publish the application updates to Azure

In this step of the tutorial, you'll publish the application updates to Azure to run it in the cloud.

1. Right-click the **ContosoTeamStats** project in Visual Studio and choose **Publish**.



2. Click **Publish** to use the same publishing profile you created in the quickstart.
3. Once publishing is complete, Visual Studio launches the app in your default web browser.

Index

Create New	Play Season	Clear Cache	List from Cache	Sorted Set from Cache	Top 5 Teams from Cache	Load from DB	Rebuild DB
Name			Wins	Losses	Ties		
Contoso, Ltd.			29	0	3	Edit Details Delete	
Alpine Ski House			28	3	1	Edit Details Delete	
Fabrikam, Inc.			18	2	3	Edit Details Delete	
Adventure Works Cycles			13	22	3	Edit Details Delete	
Lucerne Publishing			12	0	2	Edit Details Delete	
Consolidated Messenger			9	14	3	Edit Details Delete	
Fourth Coffee			9	11	4	Edit Details Delete	
Northwind Traders			8	22	2	Edit Details Delete	
Coho Vineyard			8	13	4	Edit Details Delete	
Blue Yonder Airlines			7	27	2	Edit Details Delete	
Graphic Design Institute			6	16	2	Edit Details Delete	
Nod Publishers			4	13	1	Edit Details Delete	

Results read from DB. MS: 3.05434047925727

The following table describes each action link from the sample application:

ACTION	DESCRIPTION
Create New	Create a new Team.
Play Season	Play a season of games, update the team stats, and clear any outdated team data from the cache.
Clear Cache	Clear the team stats from the cache.
List from Cache	Retrieve the team stats from the cache. If there is a cache miss, load the stats from the database and save to the cache for next time.
Sorted Set from Cache	Retrieve the team stats from the cache using a sorted set. If there is a cache miss, load the stats from the database and save to the cache using a sorted set.
Top 5 Teams from Cache	Retrieve the top 5 teams from the cache using a sorted set. If there is a cache miss, load the stats from the database and save to the cache using a sorted set.
Load from DB	Retrieve the team stats from the database.

ACTION	DESCRIPTION
Rebuild DB	Rebuild the database and reload it with sample team data.
Edit / Details / Delete	Edit a team, view details for a team, delete a team.

Click some of the actions and experiment with retrieving the data from the different sources. Note the differences in the time it takes to complete the various ways of retrieving the data from the database and the cache.

Clean up resources

When you are finished with the sample tutorial application, you can delete the Azure resources used in order to conserve cost and resources. All of your resources should be contained in the same resource group, you can delete them together in one operation by deleting the resource group. The instructions for this topic used a resource group named *TestResources*.

IMPORTANT

Deleting a resource group is irreversible and that the resource group and all the resources in it are permanently deleted. Make sure that you do not accidentally delete the wrong resource group or resources. If you created the resources for hosting this sample inside an existing resource group, that contains resources you want to keep, you can delete each resource individually from their respective blades.

1. Sign in to the [Azure portal](#) and click **Resource groups**.
2. Type the name of your resource group into the **Filter items...** textbox.
3. Click ... to the right of your resource group and click **Delete resource group**.

The screenshot shows the Microsoft Azure portal interface. On the left, there's a sidebar with options like 'Create a resource', 'All services', 'FAVORITES' (which includes 'Dashboard', 'Resource groups' - this option is highlighted with a red box, 'All resources', and 'Recent'). The main area is titled 'Resource groups' and shows a table with one item: 'TestResources'. The table has columns for 'NAME', 'SUBSCRIPTION', and 'LOCATION'. The 'NAME' column shows 'TestResources'. The 'SUBSCRIPTION' column shows 'Prototype3'. The 'LOCATION' column shows 'All locations'. At the bottom right of the table row for 'TestResources', there's a blue button labeled 'Delete resource group' with three dots next to it, which is also highlighted with a red box. Above the table, there's a search bar and some filter options: 'Subscriptions: 1 of 7 selected - Don't see a subscription? Switch directories', 'TestResources' (selected), 'Prototype3', 'All locations', and 'No grouping'.

4. You will be asked to confirm the deletion of the resource group. Type the name of your resource group to confirm, and click **Delete**.

After a few moments, the resource group and all of its contained resources are deleted.

Next steps

[How to Scale Azure Cache for Redis](#)

Azure Cache for Redis samples

12/13/2019 • 2 minutes to read • [Edit Online](#)

This topic provides a list of Azure Cache for Redis samples, covering scenarios such as connecting to a cache, reading and writing data to and from a cache, and using the ASP.NET Azure Cache for Redis providers. Some of the samples are downloadable projects, and some provide step-by-step guidance and include code snippets but do not link to a downloadable project.

Hello world samples

The samples in this section show the basics of connecting to an Azure Cache for Redis instance and reading and writing data to the cache using a variety of languages and Redis clients.

The [Hello world](#) sample shows how to perform various cache operations using the [StackExchange.Redis](#) .NET client.

This sample shows how to:

- Use various connection options
- Read and write objects to and from the cache using synchronous and asynchronous operations
- Use Redis MGET/MSET commands to return values of specified keys
- Perform Redis transactional operations
- Work with Redis lists and sorted sets
- Store .NET objects using JsonConvert serializers
- Use Redis sets to implement tagging
- Work with Redis Cluster

For more information, see the [StackExchange.Redis](#) documentation on GitHub, and for more usage scenarios see the [StackExchange.Redis.Tests](#) unit tests.

[How to use Azure Cache for Redis with Python](#) shows how to get started with Azure Cache for Redis using Python and the [redis-py](#) client.

[Work with .NET objects in the cache](#) shows you one way to serialize .NET objects so you can write them to and read them from an Azure Cache for Redis instance.

Use Azure Cache for Redis as a Scale out Backplane for ASP.NET SignalR

The [Use Azure Cache for Redis as a Scale out Backplane for ASP.NET SignalR](#) sample demonstrates how you can use Azure Cache for Redis as a SignalR backplane. For more information about backplane, see [SignalR Scaleout with Redis](#).

Azure Cache for Redis customer query sample

This sample demonstrates compares performance between accessing data from a cache and accessing data from persistence storage. This sample has two projects.

- [Demo how Azure Cache for Redis can improve performance by Caching data](#)
- [Seed the Database and Cache for the demo](#)

ASP.NET Session State and Output Caching

The [Use Azure Cache for Redis to store ASP.NET SessionState and OutputCache](#) sample demonstrates how you to use Azure Cache for Redis to store ASP.NET Session and Output Cache using the SessionState and OutputCache providers for Redis.

Manage Azure Cache for Redis with MAML

The [Manage Azure Cache for Redis using Azure Management Libraries](#) sample demonstrates how can you use Azure Management Libraries to manage - (Create/ Update/ delete) your Cache.

Custom monitoring sample

The [Access Azure Cache for Redis Monitoring data](#) sample demonstrates how you can access monitoring data for your Azure Cache for Redis outside of the Azure Portal.

A Twitter-style clone written using PHP and Redis

The [Retwis](#) sample is the Redis Hello World. It is a minimal Twitter-style social network clone written using Redis and PHP using the [Predis](#) client. The source code is designed to be very simple and at the same time to show different Redis data structures.

Bandwidth monitor

The [Bandwidth monitor](#) sample allows you to monitor the bandwidth used on the client. To measure the bandwidth, run the sample on the cache client machine, make calls to the cache, and observe the bandwidth reported by the bandwidth monitor sample.

Manage Azure Cache for Redis with Azure PowerShell

1/14/2020 • 21 minutes to read • [Edit Online](#)

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

This topic shows you how to perform common tasks such as create, update, and scale your Azure Cache for Redis instances, how to regenerate access keys, and how to view information about your caches. For a complete list of Azure Cache for Redis PowerShell cmdlets, see [Azure Cache for Redis cmdlets](#).

NOTE

Azure has two different deployment models you can use to create and work with resources: [Azure Resource Manager and classic](#). This article covers the use of the Resource Manager deployment model. We recommend the Resource Manager deployment model for new deployments instead of the classic deployment model.

For more information about the classic deployment model, see [Azure Resource Manager vs. classic deployment: Understand deployment models and the state of your resources](#).

Prerequisites

If you have already installed Azure PowerShell, you must have Azure PowerShell version 1.0.0 or later. You can check the version of Azure PowerShell that you have installed with this command at the Azure PowerShell command prompt.

```
Get-Module Az | format-table version
```

First, you must log in to Azure with this command.

```
Connect-AzAccount
```

Specify the email address of your Azure account and its password in the Microsoft Azure sign-in dialog.

Next, if you have multiple Azure subscriptions, you need to set your Azure subscription. To see a list of your current subscriptions, run this command.

```
Get-AzSubscription | sort SubscriptionName | Select SubscriptionName
```

To specify the subscription, run the following command. In the following example, the subscription name is `ContosoSubscription`.

```
Select-AzSubscription -SubscriptionName ContosoSubscription
```

Before you can use Windows PowerShell with Azure Resource Manager, you need the following:

- Windows PowerShell, Version 3.0 or 4.0. To find the version of Windows PowerShell, type: `$PSVersionTable` and verify the value of `PSVersion` is 3.0 or 4.0. To install a compatible version, see [Windows Management Framework 3.0](#) or [Windows Management Framework 4.0](#).

To get detailed help for any cmdlet you see in this tutorial, use the `Get-Help` cmdlet.

```
Get-Help <cmdlet-name> -Detailed
```

For example, to get help for the `New-AzRedisCache` cmdlet, type:

```
Get-Help New-AzRedisCache -Detailed
```

How to connect to other clouds

By default the Azure environment is `AzureCloud`, which represents the global Azure cloud instance. To connect to a different instance, use the `Connect-AzAccount` command with the `-Environment` or `-EnvironmentName` command line switch with the desired environment or environment name.

To see the list of available environments, run the `Get-AzEnvironment` cmdlet.

To connect to the Azure Government Cloud

To connect to the Azure Government Cloud, use one of the following commands.

```
Connect-AzAccount -EnvironmentName AzureUSGovernment
```

or

```
Connect-AzAccount -Environment (Get-AzEnvironment -Name AzureUSGovernment)
```

To create a cache in the Azure Government Cloud, use one of the following locations.

- USGov Virginia
- USGov Iowa

For more information about the Azure Government Cloud, see [Microsoft Azure Government](#) and [Microsoft Azure Government Developer Guide](#).

To connect to the Azure China Cloud

To connect to the Azure China Cloud, use one of the following commands.

```
Connect-AzAccount -EnvironmentName AzureChinaCloud
```

or

```
Connect-AzAccount -Environment (Get-AzEnvironment -Name AzureChinaCloud)
```

To create a cache in the Azure China Cloud, use one of the following locations.

- China East
- China North

For more information about the Azure China Cloud, see [AzureChinaCloud for Azure operated by 21Vianet in China](#).

To connect to Microsoft Azure Germany

To connect to Microsoft Azure Germany, use one of the following commands.

```
Connect-AzAccount -EnvironmentName AzureGermanCloud
```

or

```
Connect-AzAccount -Environment (Get-AzEnvironment -Name AzureGermanCloud)
```

To create a cache in Microsoft Azure Germany, use one of the following locations.

- Germany Central
- Germany Northeast

For more information about Microsoft Azure Germany, see [Microsoft Azure Germany](#).

Properties used for Azure Cache for Redis PowerShell

The following table contains properties and descriptions for commonly used parameters when creating and managing your Azure Cache for Redis instances using Azure PowerShell.

PARAMETER	DESCRIPTION	DEFAULT
Name	Name of the cache	
Location	Location of the cache	
ResourceGroupName	Resource group name in which to create the cache	
Size	The size of the cache. Valid values are: P1, P2, P3, P4, C0, C1, C2, C3, C4, C5, C6, 250MB, 1GB, 2.5GB, 6GB, 13GB, 26GB, 53GB	1GB
ShardCount	The number of shards to create when creating a premium cache with clustering enabled. Valid values are: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10	
SKU	Specifies the SKU of the cache. Valid values are: Basic, Standard, Premium	Standard
RedisConfiguration	Specifies Redis configuration settings. For details on each setting, see the following RedisConfiguration properties table.	
EnableNonSslPort	Indicates whether the non-SSL port is enabled.	False

PARAMETER	DESCRIPTION	DEFAULT
MaxMemoryPolicy	This parameter has been deprecated - use RedisConfiguration instead.	
StaticIP	When hosting your cache in a VNET, specifies a unique IP address in the subnet for the cache. If not provided, one is chosen for you from the subnet.	
Subnet	When hosting your cache in a VNET, specifies the name of the subnet in which to deploy the cache.	
VirtualNetwork	When hosting your cache in a VNET, specifies the resource ID of the VNET in which to deploy the cache.	
KeyType	Specifies which access key to regenerate when renewing access keys. Valid values are: Primary, Secondary	

RedisConfiguration properties

PROPERTY	DESCRIPTION	PRICING TIERS
rdb-backup-enabled	Whether Redis data persistence is enabled	Premium only
rdb-storage-connection-string	The connection string to the storage account for Redis data persistence	Premium only
rdb-backup-frequency	The backup frequency for Redis data persistence	Premium only
maxmemory-reserved	Configures the memory reserved for non-cache processes	Standard and Premium
maxmemory-policy	Configures the eviction policy for the cache	All pricing tiers
notify-keyspace-events	Configures keyspace notifications	Standard and Premium
hash-max-ziplist-entries	Configures memory optimization for small aggregate data types	Standard and Premium
hash-max-ziplist-value	Configures memory optimization for small aggregate data types	Standard and Premium
set-max-intset-entries	Configures memory optimization for small aggregate data types	Standard and Premium
zset-max-ziplist-entries	Configures memory optimization for small aggregate data types	Standard and Premium

PROPERTY	DESCRIPTION	PRICING TIERS
zset-max-ziplist-value	Configures memory optimization for small aggregate data types	Standard and Premium
databases	Configures the number of databases. This property can be configured only at cache creation.	Standard and Premium

To create an Azure Cache for Redis

New Azure Cache for Redis instances are created using the [New-AzRedisCache](#) cmdlet.

IMPORTANT

The first time you create an Azure Cache for Redis in a subscription using the Azure portal, the portal registers the `Microsoft.Cache` namespace for that subscription. If you attempt to create the first Azure Cache for Redis in a subscription using PowerShell, you must first register that namespace using the following command; otherwise cmdlets such as `New-AzRedisCache` and `Get-AzRedisCache` fail.

```
Register-AzResourceProvider -ProviderNamespace "Microsoft.Cache"
```

To see a list of available parameters and their descriptions for `New-AzRedisCache`, run the following command.

```
PS C:\> Get-Help New-AzRedisCache -detailed

NAME
    New-AzRedisCache

SYNOPSIS
    Creates a new Azure Cache for Redis.

SYNTAX
    New-AzRedisCache -Name <String> -ResourceGroupName <String> -Location <String> [-RedisVersion <String>]
        [-Size <String>] [-Sku <String>] [-MaxMemoryPolicy <String>] [-RedisConfiguration <Hashtable>] [-
        EnableNonSslPort
            <Boolean>] [-ShardCount <Integer>] [-VirtualNetwork <String>] [-Subnet <String>] [-StaticIP <String>]
            [<CommonParameters>]

DESCRIPTION
    The New-AzRedisCache cmdlet creates a new Azure Cache for Redis.

PARAMETERS
    -Name <String>
        Name of the Azure Cache for Redis to create.

    -ResourceGroupName <String>
        Name of resource group in which to create the Azure Cache for Redis.

    -Location <String>
        Location in which to create the Azure Cache for Redis.

    -RedisVersion <String>
        RedisVersion is deprecated and will be removed in future release.

    -Size <String>
        Size of the Azure Cache for Redis. The default value is 1GB or C1. Possible values are P1, P2, P3, P4,
        C0, C1, C2, C3,
        C4, C5, C6, 250MB, 1GB, 2.5GB, 6GB, 13GB, 26GB, 53GB.
```

```

-Sku <String>
  Sku of Azure Cache for Redis. The default value is Standard. Possible values are Basic, Standard and Premium.

-MaxMemoryPolicy <String>
  The 'MaxMemoryPolicy' setting has been deprecated. Please use 'RedisConfiguration' setting to set MaxMemoryPolicy. e.g. -RedisConfiguration @{"maxmemory-policy" = "allkeys-lru"}

-RedisConfiguration <Hashtable>
  All Redis Configuration Settings. Few possible keys: rdb-backup-enabled, rdb-storage-connection-string, rdb-backup-frequency, maxmemory-reserved, maxmemory-policy, notify-keyspace-events, hash-max-ziplist-entries, hash-max-ziplist-value, set-max-intset-entries, zset-max-ziplist-entries, zset-max-ziplist-value, databases.

-EnableNonSslPort <Boolean>
  EnableNonSslPort is used by Azure Cache for Redis. If no value is provided, the default value is false and the non-SSL port will be disabled. Possible values are true and false.

-ShardCount <Integer>
  The number of shards to create on a Premium Cluster Cache.

-VirtualNetwork <String>
  The exact ARM resource ID of the virtual network to deploy the Azure Cache for Redis in. Example format: /subscriptions/{subid}/resourceGroups/{resourceGroupName}/providers/Microsoft.ClassicNetwork/VirtualNetworks/{vnetName}

-Subnet <String>
  Required when deploying an Azure Cache for Redis inside an existing Azure Virtual Network.

-StaticIP <String>
  Required when deploying an Azure Cache for Redis inside an existing Azure Virtual Network.

<CommonParameters>
  This cmdlet supports the common parameters: Verbose, Debug, ErrorAction, ErrorVariable, WarningAction, WarningVariable, OutBuffer, PipelineVariable, and OutVariable. For more information, see about_CommonParameters (https://go.microsoft.com/fwlink/?LinkID=113216).

```

To create a cache with default parameters, run the following command.

```
New-AzRedisCache -ResourceGroupName myGroup -Name mycache -Location "North Central US"
```

`ResourceGroupName`, `Name`, and `Location` are required parameters, but the rest are optional and have default values. Running the previous command creates a Standard SKU Azure Cache for Redis instance with the specified name, location, and resource group, that is 1 GB in size with the non-SSL port disabled.

To create a premium cache, specify a size of P1 (6 GB - 60 GB), P2 (13 GB - 130 GB), P3 (26 GB - 260 GB), or P4 (53 GB - 530 GB). To enable clustering, specify a shard count using the `ShardCount` parameter. The following example creates a P1 premium cache with 3 shards. A P1 premium cache is 6 GB in size, and since we specified three shards the total size is 18 GB (3 x 6 GB).

```
New-AzRedisCache -ResourceGroupName myGroup -Name mycache -Location "North Central US" -Sku Premium -Size P1 -ShardCount 3
```

To specify values for the `RedisConfiguration` parameter, enclose the values inside `{}` as key/value pairs like `@{"maxmemory-policy" = "allkeys-random", "notify-keyspace-events" = "KEA"}`. The following example creates a standard 1 GB cache with `allkeys-random` maxmemory policy and keyspace notifications configured with `KEA`. For more information, see [Keyspace notifications \(advanced settings\)](#) and [Memory policies](#).

```
New-AzRedisCache -ResourceGroupName myGroup -Name mycache -Location "North Central US" -RedisConfiguration @{"maxmemory-policy" = "allkeys-random", "notify-keyspace-events" = "KEA"}
```

To configure the databases setting during cache creation

The `databases` setting can be configured only during cache creation. The following example creates a premium P3 (26 GB) cache with 48 databases using the [New-AzRedisCache](#) cmdlet.

```
New-AzRedisCache -ResourceGroupName myGroup -Name mycache -Location "North Central US" -Sku Premium -Size P3 -RedisConfiguration @{"databases" = "48"}
```

For more information on the `databases` property, see [Default Azure Cache for Redis server configuration](#). For more information on creating a cache using the [New-AzRedisCache](#) cmdlet, see the previous To create an Azure Cache for Redis section.

To update an Azure Cache for Redis

Azure Cache for Redis instances are updated using the [Set-AzRedisCache](#) cmdlet.

To see a list of available parameters and their descriptions for `Set-AzRedisCache`, run the following command.

```
PS C:\> Get-Help Set-AzRedisCache -detailed
```

NAME

Set-AzRedisCache

SYNOPSIS

Set Azure Cache for Redis updatable parameters.

SYNTAX

```
Set-AzRedisCache -Name <String> -ResourceGroupName <String> [-Size <String>] [-Sku <String>]
[-MaxMemoryPolicy <String>] [-RedisConfiguration <Hashtable>] [-EnableNonSslPort <Boolean>] [-ShardCount
<Integer>] [<CommonParameters>]
```

DESCRIPTION

The Set-AzRedisCache cmdlet sets Azure Cache for Redis parameters.

PARAMETERS

-Name <String>

Name of the Azure Cache for Redis to update.

-ResourceGroupName <String>

Name of the resource group for the cache.

-Size <String>

Size of the Azure Cache for Redis. The default value is 1GB or C1. Possible values are P1, P2, P3, P4, C0, C1, C2, C3, C4, C5, C6, 250MB, 1GB, 2.5GB, 6GB, 13GB, 26GB, 53GB.

-Sku <String>

Sku of Azure Cache for Redis. The default value is Standard. Possible values are Basic, Standard and Premium.

-MaxMemoryPolicy <String>

The 'MaxMemoryPolicy' setting has been deprecated. Please use 'RedisConfiguration' setting to set MaxMemoryPolicy. e.g. -RedisConfiguration @{"maxmemory-policy" = "allkeys-lru"}

-RedisConfiguration <Hashtable>

All Redis Configuration Settings. Few possible keys: rdb-backup-enabled, rdb-storage-connection-string, rdb-backup-frequency, maxmemory-reserved, maxmemory-policy, notify-keyspace-events, hash-max-ziplist-entries, hash-max-ziplist-value, set-max-intset-entries, zset-max-ziplist-entries, zset-max-ziplist-value.

-EnableNonSslPort <Boolean>

EnableNonSslPort is used by Azure Cache for Redis. The default value is null and no change will be made to the currently configured value. Possible values are true and false.

-ShardCount <Integer>

The number of shards to create on a Premium Cluster Cache.

<CommonParameters>

This cmdlet supports the common parameters: Verbose, Debug, ErrorAction, ErrorVariable, WarningAction, WarningVariable, OutBuffer, PipelineVariable, and OutVariable. For more information, see about_CommonParameters (<https://go.microsoft.com/fwlink/?LinkID=113216>).

The `Set-AzRedisCache` cmdlet can be used to update properties such as `Size`, `Sku`, `EnableNonSslPort`, and the `RedisConfiguration` values.

The following command updates the maxmemory-policy for the Azure Cache for Redis named myCache.

```
Set-AzRedisCache -ResourceGroupName "myGroup" -Name "myCache" -RedisConfiguration @{"maxmemory-policy" =
"allkeys-random"}
```

To scale an Azure Cache for Redis

`Set-AzRedisCache` can be used to scale an Azure Cache for Redis instance when the `Size`, `Sku`, or `ShardCount` properties are modified.

NOTE

Scaling a cache using PowerShell is subject to the same limits and guidelines as scaling a cache from the Azure portal. You can scale to a different pricing tier with the following restrictions.

- You can't scale from a higher pricing tier to a lower pricing tier.
- You can't scale from a **Premium** cache down to a **Standard** or a **Basic** cache.
- You can't scale from a **Standard** cache down to a **Basic** cache.
- You can scale from a **Basic** cache to a **Standard** cache but you can't change the size at the same time. If you need a different size, you can do a subsequent scaling operation to the desired size.
- You can't scale from a **Basic** cache directly to a **Premium** cache. You must scale from **Basic** to **Standard** in one scaling operation, and then from **Standard** to **Premium** in a subsequent scaling operation.
- You can't scale from a larger size down to the **C0 (250 MB)** size.

For more information, see [How to Scale Azure Cache for Redis](#).

The following example shows how to scale a cache named `myCache` to a 2.5 GB cache. Note that this command works for both a Basic or a Standard cache.

```
Set-AzRedisCache -ResourceGroupName myGroup -Name myCache -Size 2.5GB
```

After this command is issued, the status of the cache is returned (similar to calling `Get-AzRedisCache`). Note that the `ProvisioningState` is `Scaling`.

```
PS C:\> Set-AzRedisCache -Name myCache -ResourceGroupName myGroup -Size 2.5GB
```

```
Name      : mycache
Id       : /subscriptions/12ad12bd-abdc-2231-a2ed-a2b8b246bbad4/resourceGroups/mygroup/providers/Microsoft.Cache/Redis/mycache
Location   : South Central US
Type      : Microsoft.Cache/Redis
HostName   : mycache.redis.cache.windows.net
Port      : 6379
ProvisioningState : Scaling
SslPort    : 6380
RedisConfiguration : {[maxmemory-policy, volatile-lru], [maxmemory-reserved, 150], [notify-keyspace-events, KEA],
                     [maxmemory-delta, 150]...}
EnableNonSslPort : False
RedisVersion : 3.0
Size      : 1GB
Sku       : Standard
ResourceGroupName : mygroup
PrimaryKey  : ....
SecondaryKey : ....
VirtualNetwork :
Subnet     :
StaticIP   :
TenantSettings : {}
ShardCount  :
```

When the scaling operation is complete, the `ProvisioningState` changes to `Succeeded`. If you need to make a subsequent scaling operation, such as changing from Basic to Standard and then changing the size, you must wait

until the previous operation is complete or you receive an error similar to the following.

```
Set-AzRedisCache : Conflict: The resource '...' is not in a stable state, and is currently unable to accept the update request.
```

To get information about an Azure Cache for Redis

You can retrieve information about a cache using the [Get-AzRedisCache](#) cmdlet.

To see a list of available parameters and their descriptions for [Get-AzRedisCache](#), run the following command.

```
PS C:\> Get-Help Get-AzRedisCache -detailed
```

NAME
Get-AzRedisCache

SYNOPSIS
Gets details about a single cache or all caches in the specified resource group or all caches in the current subscription.

SYNTAX
`Get-AzRedisCache [-Name <String>] [-ResourceGroupName <String>] [<CommonParameters>]`

DESCRIPTION
The `Get-AzRedisCache` cmdlet gets the details about a cache or caches depending on input parameters. If both `ResourceGroupName` and `Name` parameters are provided then `Get-AzRedisCache` will return details about the specific cache name provided.

If only `ResourceGroupName` is provided then it will return details about all caches in the specified resource group.

If no parameters are given then it will return details about all caches in the current subscription.

PARAMETERS

- `Name` <String>
The name of the cache. When this parameter is provided along with `ResourceGroupName`, `Get-AzRedisCache` returns the details for the cache.
- `ResourceGroupName` <String>
The name of the resource group that contains the cache or caches. If `ResourceGroupName` is provided with `Name` then `Get-AzRedisCache` returns the details of the cache specified by `Name`. If only the `ResourceGroup` parameter is provided, then details for all caches in the resource group are returned.
- <CommonParameters>
This cmdlet supports the common parameters: `Verbose`, `Debug`, `ErrorAction`, `ErrorVariable`, `WarningAction`, `WarningVariable`, `OutBuffer`, `PipelineVariable`, and `OutVariable`. For more information, see [about_CommonParameters](#) (<https://go.microsoft.com/fwlink/?LinkID=113216>).

To return information about all caches in the current subscription, run [Get-AzRedisCache](#) without any parameters.

```
Get-AzRedisCache
```

To return information about all caches in a specific resource group, run [Get-AzRedisCache](#) with the `ResourceGroupName` parameter.

```
Get-AzRedisCache -ResourceGroupName myGroup
```

To return information about a specific cache, run `Get-AzRedisCache` with the `Name` parameter containing the name of the cache, and the `ResourceGroupName` parameter with the resource group containing that cache.

```
PS C:\> Get-AzRedisCache -Name myCache -ResourceGroupName myGroup

Name          : mycache
Id           : /subscriptions/12ad12bd-abdc-2231-a2ed-a2b8b246bbad4/resourceGroups/myGroup/providers/Microsoft.Cache/Redis/mycache
Location      : South Central US
Type          : Microsoft.Cache/Redis
HostName      : mycache.redis.cache.windows.net
Port          : 6379
ProvisioningState : Succeeded
SslPort        : 6380
RedisConfiguration : {[maxmemory-policy, volatile-lru], [maxmemory-reserved, 62], [notify-keyspace-events, KEA],
                      [maxclients, 1000]...}
EnableNonSslPort : False
RedisVersion   : 3.0
Size          : 1GB
Sku           : Standard
ResourceGroupName : myGroup
VirtualNetwork  :
Subnet         :
StaticIP       :
TenantSettings : {}
ShardCount     :
```

To retrieve the access keys for an Azure Cache for Redis

To retrieve the access keys for your cache, you can use the `Get-AzRedisCacheKey` cmdlet.

To see a list of available parameters and their descriptions for `Get-AzRedisCacheKey`, run the following command.

```
PS C:\> Get-Help Get-AzRedisCacheKey -detailed

NAME
    Get-AzRedisCacheKey

SYNOPSIS
    Gets the accesskeys for the specified Azure Cache for Redis.

SYNTAX
    Get-AzRedisCacheKey -Name <String> -ResourceGroupName <String> [<CommonParameters>]

DESCRIPTION
    The Get-AzRedisCacheKey cmdlet gets the access keys for the specified cache.

PARAMETERS
    -Name <String>
        Name of the Azure Cache for Redis.

    -ResourceGroupName <String>
        Name of the resource group for the cache.

    <CommonParameters>
        This cmdlet supports the common parameters: Verbose, Debug,
        ErrorAction, ErrorVariable, WarningAction, WarningVariable,
        OutBuffer, PipelineVariable, and OutVariable. For more information, see
        about_CommonParameters (https://go.microsoft.com/fwlink/?LinkID=113216).
```

To retrieve the keys for your cache, call the `Get-AzRedisCacheKey` cmdlet and pass in the name of your cache the

name of the resource group that contains the cache.

```
PS C:\> Get-AzRedisCacheKey -Name myCache -ResourceGroupName myGroup

PrimaryKey : b2wdt43sfetlju4hfbryfnregrd9wgIcc6IA3zA01lY=
SecondaryKey : ABhfB757JgjIgt785JgKH9865eifmekfnn649303JKL=
```

To regenerate access keys for your Azure Cache for Redis

To regenerate the access keys for your cache, you can use the [New-AzRedisCacheKey](#) cmdlet.

To see a list of available parameters and their descriptions for [New-AzRedisCacheKey](#), run the following command.

```
PS C:\> Get-Help New-AzRedisCacheKey -detailed

NAME
    New-AzRedisCacheKey

SYNOPSIS
    Regenerates the access key of an Azure Cache for Redis.

SYNTAX
    New-AzRedisCacheKey -Name <String> -ResourceGroupName <String> -KeyType <String> [-Force]
    [<CommonParameters>]

DESCRIPTION
    The New-AzRedisCacheKey cmdlet regenerate the access key of an Azure Cache for Redis.

PARAMETERS
    -Name <String>
        Name of the Azure Cache for Redis.

    -ResourceGroupName <String>
        Name of the resource group for the cache.

    -KeyType <String>
        Specifies whether to regenerate the primary or secondary access key. Possible values are Primary or Secondary.

    -Force
        When the Force parameter is provided, the specified access key is regenerated without any confirmation prompts.

    <CommonParameters>
        This cmdlet supports the common parameters: Verbose, Debug,
        ErrorAction, ErrorVariable, WarningAction, WarningVariable,
        OutBuffer, PipelineVariable, and OutVariable. For more information, see
        about_CommonParameters (https://go.microsoft.com/fwlink/?LinkID=113216).
```

To regenerate the primary or secondary key for your cache, call the [New-AzRedisCacheKey](#) cmdlet and pass in the name, resource group, and specify either [Primary](#) or [Secondary](#) for the [KeyType](#) parameter. In the following example, the secondary access key for a cache is regenerated.

```
PS C:\> New-AzRedisCacheKey -Name myCache -ResourceGroupName myGroup -KeyType Secondary
```

Confirm

Are you sure you want to regenerate Secondary key for Azure Cache for Redis 'myCache'?
[Y] Yes [N] No [S] Suspend [?] Help (default is "Y"): Y

```
PrimaryKey : b2wdt43sfet1ju4hfbryfnregnd9wgIcc6IA3zA01lY=
SecondaryKey : c53hj3kh4jhHjPJk810jji785JgKH9865eifmekfn6=
```

To delete an Azure Cache for Redis

To delete an Azure Cache for Redis, use the [Remove-AzRedisCache](#) cmdlet.

To see a list of available parameters and their descriptions for [Remove-AzRedisCache](#), run the following command.

```
PS C:\> Get-Help Remove-AzRedisCache -detailed
```

NAME

Remove-AzRedisCache

SYNOPSIS

Remove Azure Cache for Redis if exists.

SYNTAX

```
Remove-AzRedisCache -Name <String> -ResourceGroupName <String> [-Force] [-PassThru] [<CommonParameters>]
```

DESCRIPTION

The Remove-AzRedisCache cmdlet removes an Azure Cache for Redis if it exists.

PARAMETERS

-Name <String>

Name of the Azure Cache for Redis to remove.

-ResourceGroupName <String>

Name of the resource group of the cache to remove.

-Force

When the Force parameter is provided, the cache is removed without any confirmation prompts.

-PassThru

By default Remove-AzRedisCache removes the cache and does not return any value. If the PassThru parameter is provided then Remove-AzRedisCache returns a boolean value indicating the success of the operation.

<CommonParameters>

This cmdlet supports the common parameters: Verbose, Debug, ErrorAction, ErrorVariable, WarningAction, WarningVariable, OutBuffer, PipelineVariable, and OutVariable. For more information, see [about_CommonParameters](#) (<https://go.microsoft.com/fwlink/?LinkID=113216>).

In the following example, the cache named [myCache](#) is removed.

```
PS C:\> Remove-AzRedisCache -Name myCache -ResourceGroupName myGroup
```

Confirm

Are you sure you want to remove Azure Cache for Redis 'myCache'?
[Y] Yes [N] No [S] Suspend [?] Help (default is "Y"): Y

To import an Azure Cache for Redis

You can import data into an Azure Cache for Redis instance using the `Import-AzRedisCache` cmdlet.

IMPORTANT

Import/Export is only available for [premium tier](#) caches. For more information about Import/Export, see [Import and Export data in Azure Cache for Redis](#).

To see a list of available parameters and their descriptions for `Import-AzRedisCache`, run the following command.

```
PS C:\> Get-Help Import-AzRedisCache -detailed

NAME
    Import-AzRedisCache

SYNOPSIS
    Import data from blobs to Azure Cache for Redis.

SYNTAX
    Import-AzRedisCache -Name <String> -ResourceGroupName <String> -Files <String[]> [-Format <String>] [-Force]
        [-PassThru] [<CommonParameters>]

DESCRIPTION
    The Import-AzRedisCache cmdlet imports data from the specified blobs into Azure Cache for Redis.

PARAMETERS
    -Name <String>
        The name of the cache.

    -ResourceGroupName <String>
        The name of the resource group that contains the cache.

    -Files <String[]>
        SAS urls of blobs whose content should be imported into the cache.

    -Format <String>
        Format for the blob. Currently "rdb" is the only supported, with other formats expected in the future.

    -Force
        When the Force parameter is provided, import will be performed without any confirmation prompts.

    -PassThru
        By default Import-AzRedisCache imports data in cache and does not return any value. If the PassThru parameter is provided then Import-AzRedisCache returns a boolean value indicating the success of the operation.

    <CommonParameters>
        This cmdlet supports the common parameters: Verbose, Debug, ErrorAction, ErrorVariable, WarningAction, WarningVariable, OutBuffer, PipelineVariable, and OutVariable. For more information, see about_CommonParameters (https://go.microsoft.com/fwlink/?LinkID=113216).
```

The following command imports data from the blob specified by the SAS uri into Azure Cache for Redis.

```
PS C:\> Import-AzRedisCache -ResourceGroupName "resourceGroupName" -Name "cacheName" -Files @("https://mystorageaccount.blob.core.windows.net/mycontainername/blobname?sv=2015-04-05&sr=b&sig=caIwutG2uDa0NZ8mjdnJdgOY8%2F8mhwRuGNdICU%2B0pI4%3D&st=2016-05-27T00%3A00%3A00Z&se=2016-05-28T00%3A00%3A00Z&sp=rwd") -Force
```

To export an Azure Cache for Redis

You can export data from an Azure Cache for Redis instance using the `Export-AzRedisCache` cmdlet.

IMPORTANT

Import/Export is only available for [premium tier](#) caches. For more information about Import/Export, see [Import and Export data in Azure Cache for Redis](#).

To see a list of available parameters and their descriptions for `Export-AzRedisCache`, run the following command.

```
PS C:\> Get-Help Export-AzRedisCache -detailed

NAME
    Export-AzRedisCache

SYNOPSIS
    Exports data from Azure Cache for Redis to a specified container.

SYNTAX
    Export-AzRedisCache -Name <String> -ResourceGroupName <String> -Prefix <String> -Container <String> [-Format <String>] [-PassThru] [<CommonParameters>]

DESCRIPTION
    The Export-AzRedisCache cmdlet exports data from Azure Cache for Redis to a specified container.

PARAMETERS
    -Name <String>
        The name of the cache.

    -ResourceGroupName <String>
        The name of the resource group that contains the cache.

    -Prefix <String>
        Prefix to use for blob names.

    -Container <String>
        SAS url of container where data should be exported.

    -Format <String>
        Format for the blob. Currently "rdb" is the only supported, with other formats expected in the future.

    -PassThru
        By default Export-AzRedisCache does not return any value. If the PassThru parameter is provided then Export-AzRedisCache returns a boolean value indicating the success of the operation.

    <CommonParameters>
        This cmdlet supports the common parameters: Verbose, Debug, ErrorAction, ErrorVariable, WarningAction, WarningVariable, OutBuffer, PipelineVariable, and OutVariable. For more information, see about_CommonParameters (https://go.microsoft.com/fwlink/?LinkID=113216).
```

The following command exports data from an Azure Cache for Redis instance into the container specified by the SAS uri.

```
PS C:\>Export-AzRedisCache -ResourceGroupName "resourceGroupName" -Name "cacheName" -Prefix "blobprefix"
-Container "https://mystorageaccount.blob.core.windows.net/mycontainer?sv=2015-04-
05&sr=c&sig=HezztBZ3DURmEGDduauE7
pvETY4kqlPI8JCNa8ATmaw%3D&st=2016-05-27T00%3A00%3A00Z&se=2016-05-28T00%3A00%3A00Z&sp=rwdl"
```

To reboot an Azure Cache for Redis

You can reboot your Azure Cache for Redis instance using the `Reset-AzRedisCache` cmdlet.

IMPORTANT

Reboot is only available for premium tier caches. For more information about rebooting your cache, see [Cache administration - reboot](#).

To see a list of available parameters and their descriptions for `Reset-AzRedisCache`, run the following command.

```
PS C:\> Get-Help Reset-AzRedisCache -detailed

NAME
    Reset-AzRedisCache

SYNOPSIS
    Reboot specified node(s) of an Azure Cache for Redis instance.

SYNTAX
    Reset-AzRedisCache -Name <String> -ResourceGroupName <String> -RebootType <String> [-ShardId <Integer>]
    [-Force] [-PassThru] [<CommonParameters>]

DESCRIPTION
    The Reset-AzRedisCache cmdlet reboots the specified node(s) of an Azure Cache for Redis instance.

PARAMETERS
    -Name <String>
        The name of the cache.

    -ResourceGroupName <String>
        The name of the resource group that contains the cache.

    -RebootType <String>
        Which node to reboot. Possible values are "PrimaryNode", "SecondaryNode", "AllNodes".

    -ShardId <Integer>
        Which shard to reboot when rebooting a premium cache with clustering enabled.

    -Force
        When the Force parameter is provided, reset will be performed without any confirmation prompts.

    -PassThru
        By default Reset-AzRedisCache does not return any value. If the PassThru parameter is provided
        then Reset-AzRedisCache returns a boolean value indicating the success of the operation.

    <CommonParameters>
        This cmdlet supports the common parameters: Verbose, Debug,
        ErrorAction, ErrorVariable, WarningAction, WarningVariable,
        OutBuffer, PipelineVariable, and OutVariable. For more information, see
        about_CommonParameters (https://go.microsoft.com/fwlink/?LinkID=113216).
```

The following command reboots both nodes of the specified cache.

```
PS C:\>Reset-AzRedisCache -ResourceGroupName "resourceGroupName" -Name "cacheName" -RebootType "AllNodes"  
-Force
```

Next steps

To learn more about using Windows PowerShell with Azure, see the following resources:

- [Azure Cache for Redis cmdlet documentation on MSDN](#)
- [Azure Resource Manager Cmdlets](#): Learn to use the cmdlets in the Azure Resource Manager module.
- [Using Resource groups to manage your Azure resources](#): Learn how to create and manage resource groups in the Azure portal.
- [Azure blog](#): Learn about new features in Azure.
- [Windows PowerShell blog](#): Learn about new features in Windows PowerShell.
- ["Hey, Scripting Guy!" Blog](#): Get real-world tips and tricks from the Windows PowerShell community.

Azure CLI Samples for Azure Cache for Redis

12/13/2019 • 2 minutes to read • [Edit Online](#)

The following table includes links to bash scripts built using the Azure CLI.

Create cache	
Create a cache	Creates a resource group and a basic tier Azure Cache for Redis.
Create a premium cache with clustering	Creates a resource group and a premium tier cache with clustering enabled.
Get cache details	Gets details of an Azure Cache for Redis instance, including provisioning status.
Get the hostname, ports, and keys	Gets the hostname, ports, and keys for an Azure Cache for Redis instance.
Web app plus cache	
Connect a web app to an Azure Cache for Redis	Creates an Azure web app and an Azure Cache for Redis, then adds the redis connection details to the app settings.
Delete cache	
Delete a cache	Deletes an Azure Cache for Redis instance

For more information about the Azure CLI, see [Install the Azure CLI](#) and [Get started with Azure CLI](#).

Failover and patching for Azure Cache for Redis

11/15/2019 • 6 minutes to read • [Edit Online](#)

To build resilient and successful client applications, it's critical to understand failover in the context of the Azure Cache for Redis service. A failover can be a part of planned management operations, or might be caused by unplanned hardware or network failures. A common use of cache failover comes when the management service patches the Azure Cache for Redis binaries. This article covers what a failover is, how it occurs during patching, and how to build a resilient client application.

What is a failover?

Let's start with an overview of failover for Azure Cache for Redis.

A quick summary of cache architecture

A cache is constructed of multiple virtual machines with separate, private IP addresses. Each virtual machine, also known as a node, is connected to a shared load balancer with a single virtual IP address. Each node runs the Redis server process and is accessible by means of the host name and the Redis ports. Each node is considered either a master or a replica node. When a client application connects to a cache, its traffic goes through this load balancer and is automatically routed to the master node.

In a Basic cache, the single node is always a master. In a Standard or Premium cache, there are two nodes: one is chosen as the master and the other is the replica. Because Standard and Premium caches have multiple nodes, one node might be unavailable while the other continues to process requests. Clustered caches are made of many shards, each with distinct master and replica nodes. One shard might be down while the others remain available.

NOTE

A Basic cache doesn't have multiple nodes and doesn't offer a service-level agreement (SLA) for its availability. Basic caches are recommended only for development and testing purposes. Use a Standard or Premium cache for a multi-node deployment, to increase availability.

Explanation of a failover

A failover occurs when a replica node promotes itself to become a master node, and the old master node closes existing connections. After the master node comes back up, it notices the change in roles and demotes itself to become a replica. It then connects to the new master and synchronizes data. A failover might be planned or unplanned.

A *planned failover* takes place during system updates, such as Redis patching or OS upgrades, and management operations, such as scaling and rebooting. Because the nodes receive advance notice of the update, they can cooperatively swap roles and quickly update the load balancer of the change. A planned failover typically finishes in less than 1 second.

An *unplanned failover* might happen because of hardware failure, network failure, or other unexpected outages to the master node. The replica node promotes itself to master, but the process takes longer. A replica node must first detect that its master node is not available before it can initiate the failover process. The replica node must also verify that this unplanned failure is not transient or local, to avoid an unnecessary failover. This delay in detection means that an unplanned failover typically finishes within 10 to 15 seconds.

How does patching occur?

The Azure Cache for Redis service regularly updates your cache with the latest platform features and fixes. To patch a cache, the service follows these steps:

1. The management service selects one node to be patched.
2. If the selected node is a master node, the corresponding replica node cooperatively promotes itself. This promotion is considered a planned failover.
3. The selected node reboots to take the new changes and comes back up as a replica node.
4. The replica node connects to the master node and synchronizes data.
5. When the data sync is complete, the patching process repeats for the remaining nodes.

Because patching is a planned failover, the replica node quickly promotes itself to become a master and begins servicing requests and new connections. Basic caches don't have a replica node and are unavailable until the update is complete. Each shard of a clustered cache is patched separately and won't close connections to another shard.

IMPORTANT

Nodes are patched one at a time to prevent data loss. Basic caches will have data loss. Clustered caches are patched one shard at a time.

Multiple caches in the same resource group and region are also patched one at a time. Caches that are in different resource groups or different regions might be patched simultaneously.

Because full data synchronization happens before the process repeats, data loss is unlikely to occur when you use a Standard or Premium cache. You can further guard against data loss by [exporting](#) data and enabling [persistence](#).

Additional cache load

Whenever a failover occurs, the Standard and Premium caches need to replicate data from one node to the other. This replication causes some load increase in both server memory and CPU. If the cache instance is already heavily loaded, client applications might experience increased latency. In extreme cases, client applications might receive time-out exceptions. To help mitigate the impact of this additional load, [configure](#) the cache's `maxmemory-reserved` setting.

How does a failover affect my client application?

The number of errors seen by the client application depends on how many operations were pending on that connection at the time of the failover. Any connection that's routed through the node that closed its connections will see errors. Many client libraries can throw different types of errors when connections break, including time-out exceptions, connection exceptions, or socket exceptions. The number and type of exceptions depends on where in the code path the request is when the cache closes its connections. For instance, an operation that sends a request but hasn't received a response when the failover occurs might get a time-out exception. New requests on the closed connection object receive connection exceptions until the reconnection happens successfully.

Most client libraries attempt to reconnect to the cache if they're configured to do so. However, unforeseen bugs can occasionally place the library objects into an unrecoverable state. If errors persist for longer than a preconfigured amount of time, the connection object should be recreated. In Microsoft.NET and other object-oriented languages, recreating the connection without restarting the application can be accomplished by using a [Lazy<T>](#) pattern.

How do I make my application resilient?

Because you can't avoid failovers completely, write your client applications for resiliency to connection breaks and failed requests. Although most client libraries automatically reconnect to the cache endpoint, few of them attempt to retry failed requests. Depending on the application scenario, it might make sense to use retry logic with backoff.

To test a client application's resiliency, use a [reboot](#) as a manual trigger for connection breaks. Additionally, we recommend that you [schedule updates](#) on a cache. Tell the management service to apply Redis runtime patches during specified weekly windows. These windows are typically periods when client application traffic is low, to avoid potential incidents.

Client network-configuration changes

Certain client-side network-configuration changes can trigger "No connection available" errors. Such changes might include:

- Swapping a client application's virtual IP address between staging and production slots.
- Scaling the size or number of instances of your application.

Such changes can cause a connectivity issue that lasts less than one minute. Your client application will probably lose its connection to other external network resources in addition to the Azure Cache for Redis service.

Next steps

- [Schedule updates](#) for your cache.
- Test application resiliency by using a [reboot](#).
- [Configure](#) memory reservations and policies.

Best practices for Azure Cache for Redis

1/6/2020 • 6 minutes to read • [Edit Online](#)

By following these best practices, you can help maximize the performance and cost-effective use of your Azure Cache for Redis instance.

Configuration and concepts

- **Use Standard or Premium tier for production systems.** The Basic tier is a single node system with no data replication and no SLA. Also, use at least a C1 cache. C0 caches are meant for simple dev/test scenarios since they have a shared CPU core, little memory, and are prone to "noisy neighbor" issues.
- **Remember that Redis is an in-memory data store.** [This article](#) outlines some scenarios where data loss can occur.
- **Develop your system such that it can handle connection blips** because of patching and failover.
- **Configure your maxmemory-reserved setting to improve system responsiveness** under memory pressure conditions. A sufficient reservation setting is especially important for write-heavy workloads or if you're storing larger values (100 KB or more) in Redis. You should start with 10% of the size of your cache and increase this percentage if you have write-heavy loads.
- **Redis works best with smaller values**, so consider chopping up bigger data into multiple keys. In [this Redis discussion](#), some considerations are listed that you should consider carefully. Read [this article](#) for an example problem that can be caused by large values.
- **Locate your cache instance and your application in the same region.** Connecting to a cache in a different region can significantly increase latency and reduce reliability. While you can connect from outside of Azure, it's not recommended *especially when using Redis as a cache*. If you're using Redis as just a key/value store, latency may not be the primary concern.
- **Reuse connections.** Creating new connections is expensive and increases latency, so reuse connections as much as possible. If you choose to create new connections, make sure to close the old connections before you release them (even in managed memory languages like .NET or Java).
- **Configure your client library to use a connect timeout of at least 15 seconds**, giving the system time to connect even under higher CPU conditions. A small connection timeout value doesn't guarantee that the connection is established in that time frame. If something goes wrong (high client CPU, high server CPU, and so on), then a short connection timeout value will cause the connection attempt to fail. This behavior often makes a bad situation worse. Instead of helping, shorter timeouts aggravate the problem by forcing the system to restart the process of trying to reconnect, which can lead to a *connect -> fail -> retry* loop. We generally recommend that you leave your connection Timeout at 15 seconds or higher. It's better to let your connection attempt succeed after 15 or 20 seconds than to have it fail quickly only to retry. Such a retry loop can cause your outage to last longer than if you let the system just take longer initially.

NOTE

This guidance is specific to the *connection attempt* and not related to the time you're willing to wait for an *operation* like GET or SET to complete.

- **Avoid expensive operations** - Some Redis operations, like the **KEYS** command, are very expensive and should be avoided. For more information, see some considerations around [long-running commands](#)

- **Use TLS encryption** - Azure Cache for Redis requires TLS encrypted communications by default. TLS versions 1.0, 1.1 and 1.2 are currently supported. However, TLS 1.0 and 1.1 are on a path to deprecation industry-wide, so use TLS 1.2 if at all possible. If your client library or tool doesn't support TLS, then enabling unencrypted connections can be done [through the Azure portal](#) or [management APIs](#). In such cases where encrypted connections aren't possible, placing your cache and client application into a virtual network would be recommended. For more information about which ports are used in the virtual network cache scenario, see this [table](#).

Memory management

There are several things related to memory usage within your Redis server instance that you may want to consider. Here are a few:

- **Choose an eviction policy that works for your application.** The default policy for Azure Redis is *volatile-lru*, which means that only keys that have a TTL value set will be eligible for eviction. If no keys have a TTL value, then the system won't evict any keys. If you want the system to allow any key to be evicted if under memory pressure, then you may want to consider the *allkeys-lru* policy.
- **Set an expiration value on your keys.** An expiration will remove keys proactively instead of waiting until there's memory pressure. When eviction does kick in because of memory pressure, it can cause additional load on your server. For more information, see the documentation for the [EXPIRE](#) and [EXPIREAT](#) commands.

Client library specific guidance

- [StackExchange.Redis \(.NET\)](#)
- [Java - Which client should I use?](#)
- [Lettuce \(Java\)](#)
- [Jedis \(Java\)](#)
- [Node.js](#)
- [PHP](#)
- [Asp.Net Session State Provider](#)

When is it safe to retry?

Unfortunately, there's no easy answer. Each application needs to decide what operations can be retried and which can't. Each operation has different requirements and inter-key dependencies. Here are some things you might consider:

- You can get client-side errors even though Redis successfully ran the command you asked it to run. For example:
 - Timeouts are a client-side concept. If the operation reached the server, the server will run the command even if the client gives up waiting.
 - When an error occurs on the socket connection, it's not possible to know if the operation actually ran on the server. For example, the connection error can happen after the server processed the request but before the client receives the response.
- How does my application react if I accidentally run the same operation twice? For instance, what if I increment an integer twice instead of once? Is my application writing to the same key from multiple places? What if my retry logic overwrites a value set by some other part of my app?

If you would like to test how your code works under error conditions, consider using the [Reboot feature](#). Rebooting allows you to see how connection blips affect your application.

Performance testing

- **Start by using** `redis-benchmark.exe` to get a feel for possible throughput/latency before writing your own perf tests. Redis-benchmark documentation can be [found here](#). Note that redis-benchmark doesn't support SSL, so you'll have to [enable the Non-SSL port through the Portal](#) before you run the test. A [windows compatible version of redis-benchmark.exe can be found here](#)
- The client VM used for testing should be **in the same region** as your Redis cache instance.
- **We recommend using Dv2 VM Series** for your client as they have better hardware and will give the best results.
- Make sure the client VM you use has **at least as much compute and bandwidth* as the cache being tested.
- **Enable VRSS** on the client machine if you are on Windows. [See here for details](#). Example powershell script:

```
PowerShell -ExecutionPolicy Unrestricted Enable-NetAdapterRSS -Name ( Get-NetAdapter).Name
```

- **Consider using Premium tier Redis instances.** These cache sizes will have better network latency and throughput because they're running on better hardware for both CPU and Network.

NOTE

Our observed performance results are [published here](#) for your reference. Also, be aware that SSL/TLS adds some overhead, so you may get different latencies and/or throughput if you're using transport encryption.

Redis-Benchmark examples

Pre-test setup: Prepare the cache instance with data required for the latency and throughput testing commands listed below.

```
redis-benchmark.exe -h yourcache.redis.cache.windows.net -a yourAccesskey -t SET -n 10 -d 1024
```

To test latency: Test GET requests using a 1k payload.

```
redis-benchmark.exe -h yourcache.redis.cache.windows.net -a yourAccesskey -t GET -d 1024 -P 50 -c 4
```

To test throughput: Pipelined GET requests with 1k payload.

```
redis-benchmark.exe -h yourcache.redis.cache.windows.net -a yourAccesskey -t GET -n 1000000 -d 1024 -P 50 -c 50
```

Azure Cache for Redis FAQ

12/17/2019 • 28 minutes to read • [Edit Online](#)

Learn the answers to common questions, patterns, and best practices for Azure Cache for Redis.

What if my question isn't answered here?

If your question isn't listed here, let us know and we'll help you find an answer.

- You can post a question in the comments at the end of this FAQ and engage with the Azure Cache team and other community members about this article.
- To reach a wider audience, you can post a question on the [Azure Cache MSDN Forum](#) and engage with the Azure Cache team and other members of the community.
- If you want to make a feature request, you can submit your requests and ideas to [Azure Cache for Redis User Voice](#).
- You can also send an email to us at [Azure Cache External Feedback](#).

Azure Cache for Redis basics

The FAQs in this section cover some of the basics of Azure Cache for Redis.

- [What is Azure Cache for Redis?](#)
- [How can I get started with Azure Cache for Redis?](#)

The following FAQs cover basic concepts and questions about Azure Cache for Redis and are answered in one of the other FAQ sections.

- [What Azure Cache for Redis offering and size should I use?](#)
- [What Azure Cache for Redis clients can I use?](#)
- [Is there a local emulator for Azure Cache for Redis?](#)
- [How do I monitor the health and performance of my cache?](#)

Planning FAQs

- [What Azure Cache for Redis offering and size should I use?](#)
- [Azure Cache for Redis performance](#)
- [In what region should I locate my cache?](#)
- [How am I billed for Azure Cache for Redis?](#)
- [Can I use Azure Cache for Redis with Azure Government Cloud, Azure China Cloud, or Microsoft Azure Germany?](#)

Development FAQs

- [What do the StackExchange.Redis configuration options do?](#)
- [What Azure Cache for Redis clients can I use?](#)
- [Is there a local emulator for Azure Cache for Redis?](#)
- [How can I run Redis commands?](#)
- [Why doesn't Azure Cache for Redis have an MSDN class library reference like some of the other Azure services?](#)

- [Can I use Azure Cache for Redis as a PHP session cache?](#)
- [What are Redis databases?](#)

Security FAQs

- [When should I enable the non-SSL port for connecting to Redis?](#)

Production FAQs

- [What are some production best practices?](#)
- [What are some of the considerations when using common Redis commands?](#)
- [How can I benchmark and test the performance of my cache?](#)
- [Important details about ThreadPool growth](#)
- [Enable server GC to get more throughput on the client when using StackExchange.Redis](#)
- [Performance considerations around connections](#)

Monitoring and troubleshooting FAQs

The FAQs in this section cover common monitoring and troubleshooting questions. For more information about monitoring and troubleshooting your Azure Cache for Redis instances, see [How to monitor Azure Cache for Redis](#) and the various troubleshoot guides.

- [How do I monitor the health and performance of my cache?](#)
- [Why am I seeing timeouts?](#)
- [Why was my client disconnected from the cache?](#)

Prior Cache offering FAQs

- [Which Azure Cache offering is right for me?](#)

What is Azure Cache for Redis?

Azure Cache for Redis is based on the popular open-source software [Redis](#). It gives you access to a secure, dedicated Azure Cache for Redis, managed by Microsoft, and accessible from any application within Azure. For a more detailed overview, see the [Azure Cache for Redis](#) product page on Azure.com.

How can I get started with Azure Cache for Redis?

There are several ways you can get started with Azure Cache for Redis.

- You can check out one of our tutorials available for [.NET](#), [ASP.NET](#), [Java](#), [Node.js](#), and [Python](#).
- You can watch [How to Build High-Performance Apps Using Microsoft Azure Cache for Redis](#).
- You can check out the client documentation for the clients that match the development language of your project to see how to use Redis. There are many Redis clients that can be used with Azure Cache for Redis. For a list of Redis clients, see <https://redis.io/clients>.

If you don't already have an Azure account, you can:

- [Open an Azure account for free](#). You get credits that can be used to try out paid Azure services. Even after the credits are used up, you can keep the account and use free Azure services and features.
- [Activate Visual Studio subscriber benefits](#). Your MSDN subscription gives you credits every month that you can use for paid Azure services.

What Azure Cache for Redis offering and size should I use?

Each Azure Cache for Redis offering provides different levels of **size**, **bandwidth**, **high availability**, and **SLA** options.

The following are considerations for choosing a Cache offering.

- **Memory:** The Basic and Standard tiers offer 250 MB – 53 GB. The Premium tier offers up to 1.2 TB (as a cluster) or 120 GB (non-clustered). For more information, see [Azure Cache for Redis Pricing](#).
- **Network Performance:** If you have a workload that requires high throughput, the Premium tier offers more bandwidth compared to Standard or Basic. Also within each tier, larger size caches have more bandwidth because of the underlying VM that hosts the cache. For more information, see the [following table](#).
- **Throughput:** The Premium tier offers the maximum available throughput. If the cache server or client reaches the bandwidth limits, you may receive timeouts on the client side. For more information, see the following table.
- **High Availability/SLA:** Azure Cache for Redis guarantees that a Standard/Premium cache is available at least 99.9% of the time. To learn more about our SLA, see [Azure Cache for Redis Pricing](#). The SLA only covers connectivity to the Cache endpoints. The SLA does not cover protection from data loss. We recommend using the Redis data persistence feature in the Premium tier to increase resiliency against data loss.
- **Redis Data Persistence:** The Premium tier allows you to persist the cache data in an Azure Storage account. In a Basic/Standard cache, all the data is stored only in memory. Underlying infrastructure issues might result in potential data loss. We recommend using the Redis data persistence feature in the Premium tier to increase resiliency against data loss. Azure Cache for Redis offers RDB and AOF (coming soon) options in Redis persistence. For more information, see [How to configure persistence for a Premium Azure Cache for Redis](#).
- **Redis Cluster:** To create caches larger than 120 GB, or to shard data across multiple Redis nodes, you can use Redis clustering, which is available in the Premium tier. Each node consists of a primary/replica cache pair for high availability. For more information, see [How to configure clustering for a Premium Azure Cache for Redis](#).
- **Enhanced security and network isolation:** Azure Virtual Network (VNET) deployment provides enhanced security and isolation for your Azure Cache for Redis, as well as subnets, access control policies, and other features to further restrict access. For more information, see [How to configure Virtual Network support for a Premium Azure Cache for Redis](#).
- **Configure Redis:** In both the Standard and Premium tiers, you can configure Redis for Keyspace notifications.
- **Maximum number of client connections:** The Premium tier offers the maximum number of clients that can connect to Redis, with a higher number of connections for larger sized caches. Clustering does not increase the number of connections available for a clustered cache. For more information, see [Azure Cache for Redis pricing](#).
- **Dedicated Core for Redis Server:** In the Premium tier, all cache sizes have a dedicated core for Redis. In the Basic/Standard tiers, the C1 size and above have a dedicated core for Redis server.
- **Redis is single-threaded** so having more than two cores does not provide additional benefit over having just two cores, but larger VM sizes typically have more bandwidth than smaller sizes. If the cache server or client reaches the bandwidth limits, then you receive timeouts on the client side.
- **Performance improvements:** Caches in the Premium tier are deployed on hardware that has faster processors, giving better performance compared to the Basic or Standard tier. Premium tier Caches have higher throughput and lower latencies.

Azure Cache for Redis performance

The following table shows the maximum bandwidth values observed while testing various sizes of Standard and Premium caches using `redis-benchmark.exe` from an IaaS VM against the Azure Cache for Redis endpoint. For SSL throughput, `redis-benchmark` is used with `stunnel` to connect to the Azure Cache for Redis endpoint.

NOTE

These values are not guaranteed and there is no SLA for these numbers, but should be typical. You should load test your own application to determine the right cache size for your application. These numbers might change as we post newer results periodically.

From this table, we can draw the following conclusions:

- Throughput for the caches that are the same size is higher in the Premium tier as compared to the Standard tier.

For example, with a 6 GB Cache, throughput of P1 is 180,000 requests per second (RPS) as compared to 100,000 RPS for C3.

- With Redis clustering, throughput increases linearly as you increase the number of shards (nodes) in the cluster. For example, if you create a P4 cluster of 10 shards, then the available throughput is $400,000 * 10 = 4$ million RPS.
- Throughput for bigger key sizes is higher in the Premium tier as compared to the Standard Tier.

Pricing Tier	Size	CPU Cores	Available Bandwidth	1-KB Value Size	1-KB Value Size
Standard cache sizes			Megabits per sec (Mb/s) / Megabytes per sec (MB/s)	Requests per second (RPS) Non-SSL	Requests per second (RPS) SSL
C0	250 MB	Shared	100 / 12.5	15,000	7,500
C1	1 GB	1	500 / 62.5	38,000	20,720
C2	2.5 GB	2	500 / 62.5	41,000	37,000
C3	6 GB	4	1000 / 125	100,000	90,000
C4	13 GB	2	500 / 62.5	60,000	55,000
C5	26 GB	4	1,000 / 125	102,000	93,000
C6	53 GB	8	2,000 / 250	126,000	120,000
Premium cache sizes		CPU cores per shard	Megabits per sec (Mb/s) / Megabytes per sec (MB/s)	Requests per second (RPS) Non-SSL, per shard	Requests per second (RPS) SSL, per shard
P1	6 GB	2	1,500 / 187.5	180,000	172,000
P2	13 GB	4	3,000 / 375	350,000	341,000
P3	26 GB	4	3,000 / 375	350,000	341,000
P4	53 GB	8	6,000 / 750	400,000	373,000
P5	120 GB	20	6,000 / 750	400,000	373,000

For instructions on setting up stunnel or downloading the Redis tools such as `redis-benchmark.exe`, see the [How can I run Redis commands?](#) section.

In what region should I locate my cache?

For best performance and lowest latency, locate your Azure Cache for Redis in the same region as your cache client application.

How am I billed for Azure Cache for Redis?

Azure Cache for Redis pricing is [here](#). The pricing page lists pricing as an hourly rate. Caches are billed on a per-minute basis from the time that the cache is created until the time that a cache is deleted. There is no option for

stopping or pausing the billing of a cache.

Can I use Azure Cache for Redis with Azure Government Cloud, Azure China Cloud, or Microsoft Azure Germany?

Yes, Azure Cache for Redis is available in Azure Government Cloud, Azure China 21Vianet Cloud, and Microsoft Azure Germany. The URLs for accessing and managing Azure Cache for Redis are different in these clouds compared with Azure Public Cloud.

CLOUD	DNS SUFFIX FOR REDIS
Public	*.redis.cache.windows.net
US Gov	*.redis.cache.usgovcloudapi.net
Germany	*.redis.cache.cloudapi.de
China	*.redis.cache.chinacloudapi.cn

For more information on considerations when using Azure Cache for Redis with other clouds, see the following links.

- [Azure Government Databases - Azure Cache for Redis](#)
- [Azure China 21Vianet Cloud - Azure Cache for Redis](#)
- [Microsoft Azure Germany](#)

For information on using Azure Cache for Redis with PowerShell in Azure Government Cloud, Azure China 21Vianet Cloud, and Microsoft Azure Germany, see [How to connect to other clouds - Azure Cache for Redis PowerShell](#).

What do the StackExchange.Redis configuration options do?

StackExchange.Redis has many options. This section talks about some of the common settings. For more detailed information about StackExchange.Redis options, see [StackExchange.Redis configuration](#).

CONFIGURATIONOPTIONS	DESCRIPTION	RECOMMENDATION
AbortOnConnectFail	When set to true, the connection will not reconnect after a network failure.	Set to false and let StackExchange.Redis reconnect automatically.
ConnectRetry	The number of times to repeat connection attempts during initial connect.	See the following notes for guidance.
ConnectTimeout	Timeout in ms for connect operations.	See the following notes for guidance.

Usually the default values of the client are sufficient. You can fine-tune the options based on your workload.

• Retries

- For ConnectRetry and ConnectTimeout, the general guidance is to fail fast and retry again. This guidance is based on your workload and how much time on average it takes for your client to issue a Redis command and receive a response.
- Let StackExchange.Redis automatically reconnect instead of checking connection status and reconnecting yourself. **Avoid using the ConnectionMultiplexer.isConnected property.**
- Snowballing - sometimes you may run into an issue where you are retrying and the retries snowball and never recovers. If snowballing occurs, you should consider using an exponential backoff retry algorithm

as described in [Retry general guidance](#) published by the Microsoft Patterns & Practices group.

- **Timeout values**

- Consider your workload and set the values accordingly. If you are storing large values, set the timeout to a higher value.
- Set `AbortOnConnectFail` to false and let StackExchange.Redis reconnect for you.
- Use a single `ConnectionMultiplexer` instance for the application. You can use a `LazyConnection` to create a single instance that is returned by a `Connection` property, as shown in [Connect to the cache using the ConnectionMultiplexer class](#).
- Set the `ConnectionMultiplexer.ClientName` property to an app instance unique name for diagnostic purposes.
- Use multiple `ConnectionMultiplexer` instances for custom workloads.
 - You can follow this model if you have varying load in your application. For example:
 - You can have one multiplexer for dealing with large keys.
 - You can have one multiplexer for dealing with small keys.
 - You can set different values for connection timeouts and retry logic for each `ConnectionMultiplexer` that you use.
 - Set the `ClientName` property on each multiplexer to help with diagnostics.
 - This guidance may lead to more streamlined latency per `ConnectionMultiplexer`.

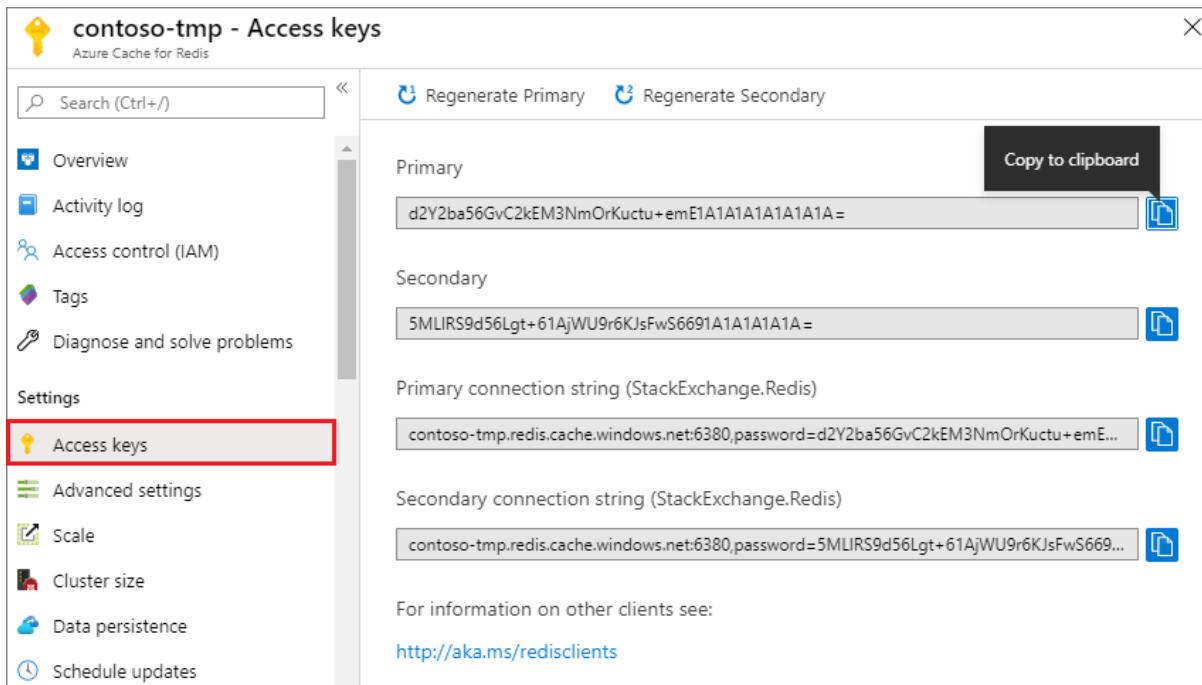
What Azure Cache for Redis clients can I use?

One of the great things about Redis is that there are many clients supporting many different development languages. For a current list of clients, see [Redis clients](#). For tutorials that cover several different languages and clients, see [How to use Azure Cache for Redis](#) and its sibling articles in the table of contents.

Retrieve host name, ports, and access keys from the Azure portal

To connect to an Azure Cache for Redis instance, cache clients need the host name, ports, and a key for the cache. Some clients might refer to these items by slightly different names. You can get the host name, ports, and keys from the [Azure portal](#).

- To get the access keys, from your cache left navigation, select **Access keys**.



- To get the host name and ports, from your cache left navigation, select **Properties**. The host name is of the form `<DNS name>.redis.cache.windows.net`.

The screenshot shows the Azure Cache for Redis properties page for a cache named 'contoso-tmp'. The left sidebar lists various settings like Access keys, Advanced settings, Scale, Cluster size, Data persistence, Schedule updates, Geo-replication, Virtual Network, Firewall, and Properties. The 'Properties' section is highlighted with a red box. On the right, the status is shown as 'Running'. The host name is 'contoso-tmp.redis.cache.windows.net', with a 'Copy to clipboard' button next to it. Non-SSL Port is set to 'Disabled'. SSL Port is set to '6380'. The location is 'Central US'.

Is there a local emulator for Azure Cache for Redis?

There is no local emulator for Azure Cache for Redis, but you can run the MSOpenTech version of redis-server.exe from the [Redis command-line tools](#) on your local machine and connect to it to get a similar experience to a local cache emulator, as shown in the following example:

```
private static Lazy<ConnectionMultiplexer>
    lazyConnection = new Lazy<ConnectionMultiplexer>
    (() =>
    {
        // Connect to a locally running instance of Redis to simulate a local cache emulator experience.
        return ConnectionMultiplexer.Connect("127.0.0.1:6379");
    });

    public static ConnectionMultiplexer Connection
    {
        get
        {
            return lazyConnection.Value;
        }
    }
}
```

You can optionally configure a [redis.conf](#) file to more closely match the [default cache settings](#) for your online Azure Cache for Redis if desired.

How can I run Redis commands?

You can use any of the commands listed at [Redis commands](#) except for the commands listed at [Redis commands not supported in Azure Cache for Redis](#). You have several options to run Redis commands.

- If you have a Standard or Premium cache, you can run Redis commands using the [Redis Console](#). The Redis console provides a secure way to run Redis commands in the Azure portal.
- You can also use the Redis command-line tools. To use them, perform the following steps:
 - Download the [Redis command-line tools](#).
 - Connect to the cache using `redis-cli.exe`. Pass in the cache endpoint using the -h switch and the key using -a as shown in the following example:
 - `redis-cli -h <Azure Cache for Redis name>.redis.cache.windows.net -a <key>`

NOTE

The Redis command-line tools do not work with the SSL port, but you can use a utility such as `stunnel` to securely connect the tools to the SSL port by following the directions in the [How to use the Redis command-line tool with Azure Cache for Redis](#) article.

Why doesn't Azure Cache for Redis have an MSDN class library reference like some of the other Azure services?

Microsoft Azure Cache for Redis is based on the popular open-source Azure Cache for Redis. It can be accessed by a wide variety of [Redis clients](#) for many programming languages. Each client has its own API that makes calls to the Azure Cache for Redis instance using [Redis commands](#).

Because each client is different, there is not one centralized class reference on MSDN, and each client maintains its own reference documentation. In addition to the reference documentation, there are several tutorials showing how to get started with Azure Cache for Redis using different languages and cache clients. To access these tutorials, see [How to use Azure Cache for Redis](#) and its sibling articles in the table of contents.

Can I use Azure Cache for Redis as a PHP session cache?

Yes, to use Azure Cache for Redis as a PHP session cache, specify the connection string to your Azure Cache for Redis instance in `session.save_path`.

IMPORTANT

When using Azure Cache for Redis as a PHP session cache, you must URL encode the security key used to connect to the cache, as shown in the following example:

```
session.save_path = "tcp://mycache.redis.cache.windows.net:6379?auth=<url encoded primary or secondary key here>;"
```

If the key is not URL encoded, you may receive an exception with a message like: `Failed to parse session.save_path`

For more information about using Azure Cache for Redis as a PHP session cache with the `PhpRedis` client, see [PHP Session handler](#).

What are Redis databases?

Redis Databases are just a logical separation of data within the same Redis instance. The cache memory is shared between all the databases and actual memory consumption of a given database depends on the keys/values stored in that database. For example, a C6 cache has 53 GB of memory, and a P5 has 120 GB. You can choose to put all 53 GB / 120 GB into one database or you can split it up between multiple databases.

NOTE

When using a Premium Azure Cache for Redis with clustering enabled, only database 0 is available. This limitation is an intrinsic Redis limitation and is not specific to Azure Cache for Redis. For more information, see [Do I need to make any changes to my client application to use clustering?](#)

When should I enable the non-SSL port for connecting to Redis?

Redis server does not natively support SSL, but Azure Cache for Redis does. If you are connecting to Azure Cache for Redis and your client supports SSL, like `StackExchange.Redis`, then you should use SSL.

NOTE

The non-SSL port is disabled by default for new Azure Cache for Redis instances. If your client does not support SSL, then you must enable the non-SSL port by following the directions in the [Access ports](#) section of the [Configure a cache in Azure Cache for Redis](#) article.

Redis tools such as `redis-cli` do not work with the SSL port, but you can use a utility such as `stunnel` to securely connect the tools to the SSL port by following the directions in the [Announcing ASP.NET Session State Provider for Redis Preview Release](#) blog post.

For instructions on downloading the Redis tools, see the [How can I run Redis commands?](#) section.

What are some production best practices?

- [StackExchange.Redis best practices](#)
- [Configuration and concepts](#)
- [Performance testing](#)

StackExchange.Redis best practices

- Set `AbortConnect` to false, then let the `ConnectionMultiplexer` reconnect automatically. [See here for details.](#)
- Reuse the `ConnectionMultiplexer` - do not create a new one for each request. The `Lazy<ConnectionMultiplexer>` pattern [shown here](#) is recommended.
- Redis works best with smaller values, so consider chopping up bigger data into multiple keys. In [this Redis discussion](#), 100 kb is considered large. Read [this article](#) for an example problem that can be caused by large values.
- Configure your [ThreadPool settings](#) to avoid timeouts.
- Use at least the default `connectTimeout` of 5 seconds. This interval gives StackExchange.Redis sufficient time to re-establish the connection in the event of a network blip.
- Be aware of the performance costs associated with different operations you are running. For instance, the `KEYS` command is an O(n) operation and should be avoided. The [redis.io site](#) has details around the time complexity for each operation that it supports. Click each command to see the complexity for each operation.

Configuration and concepts

- Use Standard or Premium Tier for Production systems. The Basic Tier is a single node system with no data replication and no SLA. Also, use at least a C1 cache. C0 caches are typically used for simple dev/test scenarios.
- Remember that Redis is an **In-Memory** data store. Read [this article](#) so that you are aware of scenarios where data loss can occur.
- Develop your system such that it can handle connection blips [due to patching and failover](#).

Performance testing

- Start by using `redis-benchmark.exe` to get a feel for possible throughput before writing your own perf tests. Because `redis-benchmark` does not support SSL, you must [enable the Non-SSL port through the Azure portal](#) before you run the test. For examples, see [How can I benchmark and test the performance of my cache?](#)
- The client VM used for testing should be in the same region as your Azure Cache for Redis instance.
- We recommend using Dv2 VM Series for your client as they have better hardware and should give the best results.
- Make sure your client VM you choose has at least as much computing and bandwidth capability as the cache you are testing.
- Enable VRSS on the client machine if you are on Windows. [See here for details.](#)
- Premium tier Redis instances have better network latency and throughput because they are running on better hardware for both CPU and Network.

What are some of the considerations when using common Redis commands?

- Avoid using certain Redis commands that take a long time to complete, unless you fully understand the impact of these commands. For example, do not run the [KEYS](#) command in production. Depending on the number of keys, it could take a long time to return. Redis is a single-threaded server and it processes commands one at a time. If you have other commands issued after KEYS, they will not be processed until Redis processes the KEYS command. The [redis.io site](#) has details around the time complexity for each operation that it supports. Click each command to see the complexity for each operation.
- Key sizes - should I use small key/values or large key/values? It depends on the scenario. If your scenario requires larger keys, you can adjust the ConnectionTimeout, then retry values and adjust your retry logic. From a Redis server perspective, smaller values give better performance.
- These considerations don't mean that you can't store larger values in Redis; you must be aware of the following considerations. Latencies will be higher. If you have one set of data that is larger and one that is smaller, you can use multiple ConnectionMultiplexer instances, each configured with a different set of timeout and retry values, as described in the previous [What do the StackExchange.Redis configuration options do](#) section.

How can I benchmark and test the performance of my cache?

- [Enable cache diagnostics](#) so you can [monitor](#) the health of your cache. You can view the metrics in the Azure portal and you can also [download and review](#) them using the tools of your choice.
- You can use redis-benchmark.exe to load test your Redis server.
- Ensure that the load testing client and the Azure Cache for Redis are in the same region.
- Use redis-cli.exe and monitor the cache using the INFO command.
- If your load is causing high memory fragmentation, you should scale up to a larger cache size.
- For instructions on downloading the Redis tools, see the [How can I run Redis commands?](#) section.

The following commands provide an example of using redis-benchmark.exe. For accurate results, run these commands from a VM in the same region as your cache.

- Test Pipelined SET requests using a 1k payload

```
redis-benchmark.exe -h **yourcache**.redis.cache.windows.net -a **yourAccesskey** -t SET -n 1000000 -d 1024 -P 50
```

- Test Pipelined GET requests using a 1k payload. NOTE: Run the SET test shown above first to populate cache

```
redis-benchmark.exe -h **yourcache**.redis.cache.windows.net -a **yourAccesskey** -t GET -n 1000000 -d 1024 -P 50
```

Important details about ThreadPool growth

The CLR ThreadPool has two types of threads - "Worker" and "I/O Completion Port" (IOCP) threads.

- Worker threads are used for things like processing the `Task.Run(...)`, or `ThreadPool.QueueUserWorkItem(...)` methods. These threads are also used by various components in the CLR when work needs to happen on a background thread.
- IOCP threads are used when asynchronous IO happens, such as when reading from the network.

The thread pool provides new worker threads or I/O completion threads on demand (without any throttling) until it reaches the "Minimum" setting for each type of thread. By default, the minimum number of threads is set to the number of processors on a system.

Once the number of existing (busy) threads hits the "minimum" number of threads, the ThreadPool will throttle the rate at which it injects new threads to one thread per 500 milliseconds. Typically, if your system gets a burst of work needing an IOCP thread, it will process that work quickly. However, if the burst of work is more than the configured "Minimum" setting, there will be some delay in processing some of the work as the ThreadPool waits for one of two things to happen.

1. An existing thread becomes free to process the work.

2. No existing thread becomes free for 500 ms, so a new thread is created.

Basically, it means that when the number of Busy threads is greater than Min threads, you are likely paying a 500-ms delay before network traffic is processed by the application. Also, it is important to note that when an existing thread stays idle for longer than 15 seconds (based on what I remember), it will be cleaned up and this cycle of growth and shrinkage can repeat.

If we look at an example error message from StackExchange.Redis (build 1.0.450 or later), you will see that it now prints ThreadPool statistics (see IOCP and WORKER details below).

```
System.TimeoutException: Timeout performing GET MyKey, inst: 2, mgr: Inactive,  
queue: 6, qu: 0, qs: 6, qc: 0, wr: 0, wq: 0, in: 0, ar: 0,  
IOCP: (Busy=6,Free=994,Min=4,Max=1000),  
WORKER: (Busy=3,Free=997,Min=4,Max=1000)
```

In the previous example, you can see that for IOCP thread there are six busy threads and the system is configured to allow four minimum threads. In this case, the client would have likely seen two 500-ms delays, because $6 > 4$.

Note that StackExchange.Redis can hit timeouts if growth of either IOCP or WORKER threads gets throttled.

Recommendation

Given this information, we strongly recommend that customers set the minimum configuration value for IOCP and WORKER threads to something larger than the default value. We can't give one-size-fits-all guidance on what this value should be because the right value for one application will likely be too high or low for another application. This setting can also impact the performance of other parts of complicated applications, so each customer needs to fine-tune this setting to their specific needs. A good starting place is 200 or 300, then test and tweak as needed.

How to configure this setting:

- We recommend changing this setting programmatically by using the [ThreadPool.SetMinThreads \(...\)](#) method in `global.asax.cs`. For example:

```
private readonly int minThreads = 200;  
void Application_Start(object sender, EventArgs e)  
{  
    // Code that runs on application startup  
    AreaRegistration.RegisterAllAreas();  
    RouteConfig.RegisterRoutes(RouteTable.Routes);  
    BundleConfig.RegisterBundles(BundleTable.Bundles);  
    ThreadPool.SetMinThreads(minThreads, minThreads);  
}
```

NOTE

The value specified by this method is a global setting, affecting the whole AppDomain. For example, if you have a 4-core machine and want to set `minWorkerThreads` and `minIoThreads` to 50 per CPU during run-time, you would use `ThreadPool.SetMinThreads(200, 200)`.

- It is also possible to specify the minimum threads setting by using the [minIoThreads](#) or [minWorkerThreads](#) configuration setting under the `<processModel>` configuration element in `Machine.config`, usually located at `%SystemRoot%\Microsoft.NET\Framework\[versionNumber]\CONFIG\`. **Setting the number of minimum threads in this way is generally not recommended, because it is a System-wide setting.**

NOTE

The value specified in this configuration element is a *per-core* setting. For example, if you have a 4-core machine and want your *minIoThreads* setting to be 200 at runtime, you would use `<processModel minIoThreads="50"/>`.

Enable server GC to get more throughput on the client when using StackExchange.Redis

Enabling server GC can optimize the client and provide better performance and throughput when using StackExchange.Redis. For more information on server GC and how to enable it, see the following articles:

- [To enable server GC](#)
- [Fundamentals of Garbage Collection](#)
- [Garbage Collection and Performance](#)

Performance considerations around connections

Each pricing tier has different limits for client connections, memory, and bandwidth. While each size of cache allows *up to* a certain number of connections, each connection to Redis has overhead associated with it. An example of such overhead would be CPU and memory usage as a result of TLS/SSL encryption. The maximum connection limit for a given cache size assumes a lightly loaded cache. If load from connection overhead *plus* load from client operations exceeds capacity for the system, the cache can experience capacity issues even if you have not exceeded the connection limit for the current cache size.

For more information about the different connections limits for each tier, see [Azure Cache for Redis pricing](#). For more information about connections and other default configurations, see [Default Redis server configuration](#).

How do I monitor the health and performance of my cache?

Microsoft Azure Cache for Redis instances can be monitored in the [Azure portal](#). You can view metrics, pin metrics charts to the Startboard, customize the date and time range of monitoring charts, add and remove metrics from the charts, and set alerts when certain conditions are met. For more information, see [Monitor Azure Cache for Redis](#).

The Azure Cache for Redis **Resource menu** also contains several tools for monitoring and troubleshooting your caches.

- **Diagnose and solve problems** provides information about common issues and strategies for resolving them.
- **Resource health** watches your resource and tells you if it's running as expected. For more information about the Azure Resource health service, see [Azure Resource health overview](#).
- **New support request** provides options to open a support request for your cache.

These tools enable you to monitor the health of your Azure Cache for Redis instances and help you manage your caching applications. For more information, see the "Support & troubleshooting settings" section of [How to configure Azure Cache for Redis](#).

Why am I seeing timeouts?

Timeouts happen in the client that you use to talk to Redis. When a command is sent to the Redis server, the command is queued up and Redis server eventually picks up the command and executes it. However the client can time out during this process and if it does an exception is raised on the calling side. For more information on troubleshooting timeout issues, see [client-side troubleshooting](#) and [StackExchange.Redis timeout exceptions](#).

Why was my client disconnected from the cache?

The following are some common reason for a cache disconnect.

- Client-side causes
 - The client application was redeployed.
 - The client application performed a scaling operation.
 - In the case of Cloud Services or Web Apps, this may be due to autoscaling.

- The networking layer on the client side changed.
 - Transient errors occurred in the client or in the network nodes between the client and the server.
 - The bandwidth threshold limits were reached.
 - CPU bound operations took too long to complete.
- Server-side causes
 - On the standard cache offering, the Azure Cache for Redis service initiated a fail-over from the primary node to the secondary node.
 - Azure was patching the instance where the cache was deployed
 - This can be for Redis server updates or general VM maintenance.

Which Azure Cache offering is right for me?

IMPORTANT

As per last year's [announcement](#), Azure Managed Cache Service and Azure In-Role Cache service **have been retired** on November 30, 2016. Our recommendation is to use [Azure Cache for Redis](#). For information on migrating, see [Migrate from Managed Cache Service to Azure Cache for Redis](#).

Azure Cache for Redis

Azure Cache for Redis is Generally Available in sizes up to 120 GB and has an availability SLA of 99.9%. The new [premium tier](#) offers sizes up to 1.2 TB and support for clustering, VNET, and persistence, with a 99.9% SLA.

Azure Cache for Redis gives customers the ability to use a secure, dedicated Azure Cache for Redis, managed by Microsoft. With this offer, you get to leverage the rich feature set and ecosystem provided by Redis, and reliable hosting and monitoring from Microsoft.

Unlike traditional caches that deal only with key-value pairs, Redis is popular for its highly performant data types. Redis also supports running atomic operations on these types, like appending to a string; incrementing the value in a hash; pushing to a list; computing set intersection, union and difference; or getting the member with highest ranking in a sorted set. Other features include support for transactions, pub/sub, Lua scripting, keys with a limited time-to-live, and configuration settings to make Redis behave more like a traditional cache.

Another key aspect to Redis success is the healthy, vibrant open- source ecosystem built around it. This is reflected in the diverse set of Redis clients available across multiple languages. This ecosystem and wide range of clients allow Azure Cache for Redis to be used by nearly any workload you would build inside of Azure.

For more information about getting started with Azure Cache for Redis, see [How to Use Azure Cache for Redis](#) and [Azure Cache for Redis documentation](#).

Managed Cache service

[Managed Cache service was retired November 30, 2016](#).

To view archived documentation, see [Archived Managed Cache Service Documentation](#).

In-Role Cache

[In-Role Cache was retired November 30, 2016](#).

To view archived documentation, see [Archived In-Role Cache Documentation](#).

How to configure Redis clustering for a Premium Azure Cache for Redis

1/23/2020 • 9 minutes to read • [Edit Online](#)

Azure Cache for Redis has different cache offerings, which provide flexibility in the choice of cache size and features, including Premium tier features such as clustering, persistence, and virtual network support. This article describes how to configure clustering in a premium Azure Cache for Redis instance.

For information on other premium cache features, see [Introduction to the Azure Cache for Redis Premium tier](#).

What is Redis Cluster?

Azure Cache for Redis offers Redis cluster as [implemented in Redis](#). With Redis Cluster, you get the following benefits:

- The ability to automatically split your dataset among multiple nodes.
- The ability to continue operations when a subset of the nodes is experiencing failures or are unable to communicate with the rest of the cluster.
- More throughput: Throughput increases linearly as you increase the number of shards.
- More memory size: Increases linearly as you increase the number of shards.

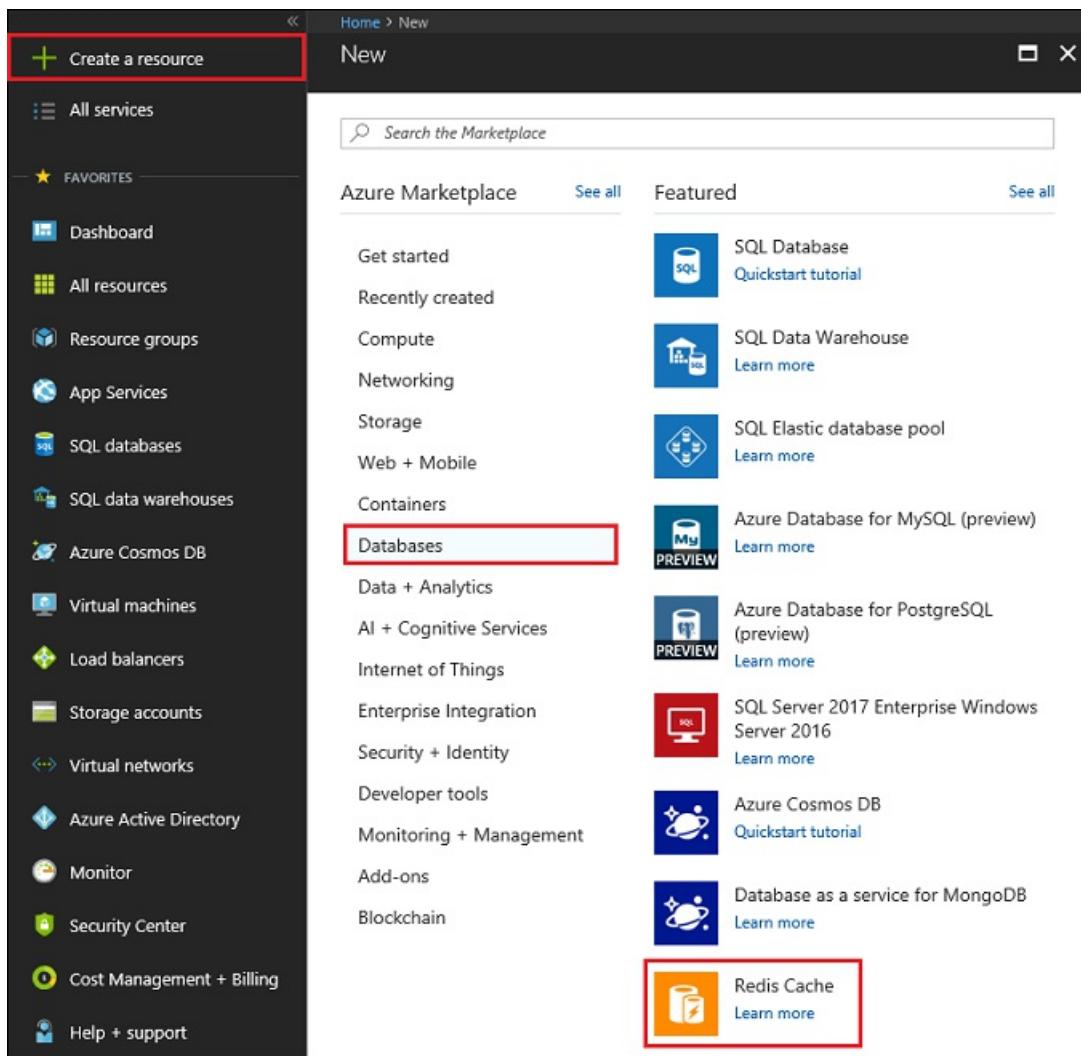
Clustering does not increase the number of connections available for a clustered cache. For more information about size, throughput, and bandwidth with premium caches, see [What Azure Cache for Redis offering and size should I use?](#)

In Azure, Redis cluster is offered as a primary/replica model where each shard has a primary/replica pair with replication where the replication is managed by Azure Cache for Redis service.

Clustering

Clustering is enabled on the **New Azure Cache for Redis** blade during cache creation.

To create a premium cache, sign in to the [Azure portal](#) and click **Create a resource > Databases > Azure Cache for Redis**.



NOTE

In addition to creating caches in the Azure portal, you can also create them using Resource Manager templates, PowerShell, or Azure CLI. For more information about creating an Azure Cache for Redis, see [Create a cache](#).

To configure premium features, first select one of the premium pricing tiers in the **Pricing tier** drop-down list. For more information about each pricing tier, click **View full pricing details** and select a pricing tier from the **Choose your pricing tier** blade.

The screenshot shows two windows side-by-side. On the left is the 'New Redis Cache' blade, where step 1 is completed with a red box around the 'Pricing tier' dropdown set to 'Premium P1 (6 GB Cache, Replication)'. On the right is the 'Choose your pricing tier' blade, which lists various Redis cache plans across different tiers (Premium, Standard). A red box highlights the P3 Premium plan, and a red circle with the number 2 is placed above it. A red box also surrounds the 'Select' button at the bottom of the blade, with a red circle containing the number 3 placed above it.

New Redis Cache

- * DNS name: contoso5premium (.redis.cache.windows.net)
- * Subscription: Prototype3
- * Resource group: Create new / Use existing
- * Location: Central US
- * Pricing tier (View full pricing details): Premium P1 (6 GB Cache, Replication) (1)
- Redis Cluster: Not configured
- Redis data persistence: Not configured
- Virtual Network: Not configured
- Unblock port 6379 (not SSL encrypted)

Choose your pricing tier

Browse the available plans and their features

P1 Premium	P2 Premium	P3 Premium
6 GB Cache	13 GB Cache	26 GB Cache
Replication	Replication	Replication
Moderate network ba...	Moderate network ba...	High network bandwi...
All Standard features	All Standard features	All Standard features
Data Persistence	Data Persistence	Data Persistence
Virtual Network	Virtual Network	Virtual Network
Redis Cluster	Redis Cluster	Redis Cluster
99.9% SLA	99.9% SLA	99.9% SLA
412.92 USD/MONTH (ESTIMATED) PER SH...	825.84 USD/MONTH (ESTIMATED) PER SH...	1,650.94 USD/MONTH (ESTIMATED) PER SH...

P4 Premium	C0 Standard	C1 Standard
53 GB Cache	250 MB Cache	1 GB Cache
Replication	Replication	Replication
Highest network ban...	Low network bandwi...	Low network bandwi...
All Standard features	Shared infrastructure	Dedicated service
Data Persistence	SSL	SSL
Virtual Network	Up to 256 connections	Up to 1,000 connectio...
Redis Cluster	Configure Redis (key...)	Configure Redis (key...
99.9% SLA	99.9% SLA	99.9% SLA
3,303.36 USD/MONTH (ESTIMATED) PER SH...	40.92 USD/MONTH (ESTIMATED)	102.67 USD/MONTH (ESTIMATED)

C2 Standard	C3 Standard	C4 Standard
2.5 GB Cache	6 GB Cache	13 GB Cache

Select (3)

Clustering is configured on the **Redis Cluster** blade.

The screenshot shows the 'New Redis Cache' blade. On the left, there are several configuration sections: 'Redis name' (with placeholder 'myredis'), 'Subscription' (selected 'Standard'), 'Resource group' (radio buttons for 'Create new' and 'Use existing'), 'Location' (selected 'East US'), and 'Pricing tier' (selected 'Standard 6 GB'). Below these is a large red box highlighting the 'Redis Cluster' section, which contains the message 'Not configured'. Underneath are three collapsed sections: 'Data durability', 'Virtual Network', and 'Advanced settings'. At the bottom left is a checkbox for 'Pin to dashboard', and at the bottom right are 'Create' and 'OK' buttons.

New Redis Cache

Redis Cluster

Clustering

Enabled **Disabled**

Shard count ⓘ

1 Total size: 6 GB

Estimating cost...

Redis Cluster ⓘ

Not configured

Redis data durability ⓘ

Not configured

Virtual Network ⓘ

Not configured

Pin to dashboard

Create OK

You can have up to 10 shards in the cluster. Click **Enabled** and slide the slider or type a number between 1 and 10 for **Shard count** and click **OK**.

Each shard is a primary/replica cache pair managed by Azure, and the total size of the cache is calculated by multiplying the number of shards by the cache size selected in the pricing tier.

The screenshot shows the 'Redis Cluster' configuration page. It features two main sections: 'Clustering' and 'Shard count'. In the 'Clustering' section, the 'Enabled' button is highlighted with a red box and a circled '1'. In the 'Shard count' section, a slider is set to '3', indicated by a red box and a circled '2'. Below the slider, the text 'Total size: 18 GB' is visible. At the bottom right is an 'OK' button, which is also highlighted with a red box and a circled '3'.

Redis Cluster

Clustering

Enabled **Disabled**

Shard count ⓘ

3 Total size: 18 GB

OK

Once the cache is created you connect to it and use it just like a non-clustered cache, and Redis distributes the data throughout the Cache shards. If diagnostics is [enabled](#), metrics are captured separately for each shard and can be [viewed](#) in the Azure Cache for Redis blade.

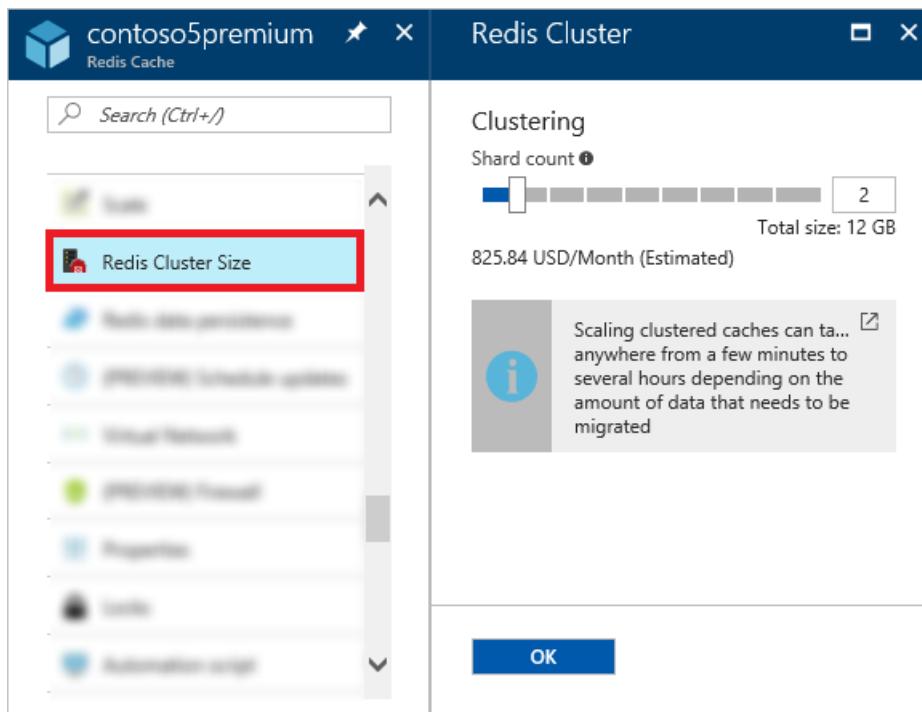
NOTE

There are some minor differences required in your client application when clustering is configured. For more information, see [Do I need to make any changes to my client application to use clustering?](#)

For sample code on working with clustering with the StackExchange.Redis client, see the [clustering.cs](#) portion of the [Hello World](#) sample.

Change the cluster size on a running premium cache

To change the cluster size on a running premium cache with clustering enabled, click **Cluster Size** from the **Resource menu**.



To change the cluster size, use the slider or type a number between 1 and 10 in the **Shard count** text box and click **OK** to save.

Increasing the cluster size increases max throughput and cache size. Increasing the cluster size doesn't increase the max. connections available to clients.

NOTE

Scaling a cluster runs the [MIGRATE](#) command, which is an expensive command, so for minimal impact, consider running this operation during non-peak hours. During the migration process, you will see a spike in server load. Scaling a cluster is a long running process and the amount of time taken depends on the number of keys and size of the values associated with those keys.

Clustering FAQ

The following list contains answers to commonly asked questions about Azure Cache for Redis clustering.

- [Do I need to make any changes to my client application to use clustering?](#)
- [How are keys distributed in a cluster?](#)
- [What is the largest cache size I can create?](#)
- [Do all Redis clients support clustering?](#)

- [How do I connect to my cache when clustering is enabled?](#)
- [Can I directly connect to the individual shards of my cache?](#)
- [Can I configure clustering for a previously created cache?](#)
- [Can I configure clustering for a basic or standard cache?](#)
- [Can I use clustering with the Redis ASP.NET Session State and Output Caching providers?](#)
- [I am getting MOVE exceptions when using StackExchange.Redis and clustering, what should I do?](#)

Do I need to make any changes to my client application to use clustering?

- When clustering is enabled, only database 0 is available. If your client application uses multiple databases and it tries to read or write to a database other than 0, the following exception is thrown.

```
Unhandled Exception: StackExchange.Redis.RedisConnectionException: ProtocolFailure on GET --->
```

```
StackExchange.Redis.RedisCommandException: Multiple databases are not supported on this server; cannot switch to database: 6
```

For more information, see [Redis Cluster Specification - Implemented subset](#).

- If you are using [StackExchange.Redis](#), you must use 1.0.481 or later. You connect to the cache using the same [endpoints, ports, and keys](#) that you use when connecting to a cache that does not have clustering enabled. The only difference is that all reads and writes must be done to database 0.
 - Other clients may have different requirements. See [Do all Redis clients support clustering?](#)
- If your application uses multiple key operations batched into a single command, all keys must be located in the same shard. To locate keys in the same shard, see [How are keys distributed in a cluster?](#)
- If you are using Redis ASP.NET Session State provider you must use 2.0.1 or higher. See [Can I use clustering with the Redis ASP.NET Session State and Output Caching providers?](#)

How are keys distributed in a cluster?

Per the Redis [Keys distribution model](#) documentation: The key space is split into 16384 slots. Each key is hashed and assigned to one of these slots, which are distributed across the nodes of the cluster. You can configure which part of the key is hashed to ensure that multiple keys are located in the same shard using hash tags.

- Keys with a hash tag - if any part of the key is enclosed in `{` and `}`, only that part of the key is hashed for the purposes of determining the hash slot of a key. For example, the following 3 keys would be located in the same shard: `{key}1`, `{key}2`, and `{key}3` since only the `key` part of the name is hashed. For a complete list of keys hash tag specifications, see [Keys hash tags](#).
- Keys without a hash tag - the entire key name is used for hashing. This results in a statistically even distribution across the shards of the cache.

For best performance and throughput, we recommend distributing the keys evenly. If you are using keys with a hash tag it is the application's responsibility to ensure the keys are distributed evenly.

For more information, see [Keys distribution model](#), [Redis Cluster data sharding](#), and [Keys hash tags](#).

For sample code on working with clustering and locating keys in the same shard with the StackExchange.Redis client, see the [clustering.cs](#) portion of the [Hello World](#) sample.

What is the largest cache size I can create?

The largest premium cache size is 120 GB. You can create up to 10 shards giving you a maximum size of 1.2TB GB. If you need a larger size you can [request more](#). For more information, see [Azure Cache for Redis Pricing](#).

Do all Redis clients support clustering?

Not all clients support Redis clustering! Please check the documentation for the library you are using, to verify you are using a library and version which support clustering. StackExchange.Redis is one library that does support clustering, in its newer versions. For more information on other clients, see the [Playing with the cluster](#) section of the [Redis cluster tutorial](#).

The Redis clustering protocol requires each client to connect to each shard directly in clustering mode, and also defines new error responses such as 'MOVED' na 'CROSSSLOTS'. Attempting to use a client that doesn't support clustering with a cluster mode cache can result in a lot of [MOVED redirection exceptions](#), or just break your application, if you are doing cross-slot multi-key requests.

NOTE

If you are using StackExchange.Redis as your client, ensure you are using the latest version of [StackExchange.Redis](#) 1.0.481 or later for clustering to work correctly. If you have any issues with move exceptions, see [move exceptions](#) for more information.

How do I connect to my cache when clustering is enabled?

You can connect to your cache using the same [endpoints](#), [ports](#), and [keys](#) that you use when connecting to a cache that does not have clustering enabled. Redis manages the clustering on the backend so you don't have to manage it from your client.

Can I directly connect to the individual shards of my cache?

The clustering protocol requires that the client make the correct shard connections. So the client should do this correctly for you. With that said, each shard consists of a primary/replica cache pair, collectively known as a cache instance. You can connect to these cache instances using the redis-cli utility in the [unstable](#) branch of the Redis repository at GitHub. This version implements basic support when started with the `-c` switch. For more information, see [Playing with the cluster](#) on <https://redis.io> in the [Redis cluster tutorial](#).

For non-ssl, use the following commands.

```
Redis-cli.exe -h <<cachename>> -p 13000 (to connect to instance 0)
Redis-cli.exe -h <<cachename>> -p 13001 (to connect to instance 1)
Redis-cli.exe -h <<cachename>> -p 13002 (to connect to instance 2)
...
Redis-cli.exe -h <<cachename>> -p 1300N (to connect to instance N)
```

For ssl, replace `1300N` with `1500N`.

Can I configure clustering for a previously created cache?

Yes. First ensure that your cache is premium, by scaling if is not. Next, you should be able to see the cluster configuration options, including an option to enable cluster. You can change the cluster size after the cache is created, or after you have enabled clustering for the first time.

IMPORTANT

You can't undo enabling clustering. And a cache with clustering enabled and only one shard behaves *differently* than a cache of the same size with *no* clustering.

Can I configure clustering for a basic or standard cache?

Clustering is only available for premium caches.

Can I use clustering with the Redis ASP.NET Session State and Output Caching providers?

- **Redis Output Cache provider** - no changes required.
- **Redis Session State provider** - to use clustering, you must use [RedisSessionStateProvider](#) 2.0.1 or higher or an exception is thrown. This is a breaking change; for more information, see [v2.0.0 Breaking Change Details](#).

I am getting MOVE exceptions when using StackExchange.Redis and clustering, what should I do?

If you are using StackExchange.Redis and receive `MOVE` exceptions when using clustering, ensure that you are using [StackExchange.Redis 1.1.603](#) or later. For instructions on configuring your .NET applications to use

StackExchange.Redis, see [Configure the cache clients](#).

Next steps

Learn how to use more premium cache features.

- [Introduction to the Azure Cache for Redis Premium tier](#)

How to configure data persistence for a Premium Azure Cache for Redis

1/23/2020 • 9 minutes to read • [Edit Online](#)

Azure Cache for Redis has different cache offerings which provide flexibility in the choice of cache size and features, including Premium tier features such as clustering, persistence, and virtual network support. This article describes how to configure persistence in a premium Azure Cache for Redis instance.

For information on other premium cache features, see [Introduction to the Azure Cache for Redis Premium tier](#).

What is data persistence?

[Redis persistence](#) allows you to persist data stored in Redis. You can also take snapshots and back up the data, which you can load in case of a hardware failure. This is a huge advantage over Basic or Standard tier where all the data is stored in memory and there can be potential data loss in case of a failure where Cache nodes are down.

Azure Cache for Redis offers Redis persistence using the following models:

- **RDB persistence** - When RDB (Redis database) persistence is configured, Azure Cache for Redis persists a snapshot of the Azure Cache for Redis in a Redis binary format to disk based on a configurable backup frequency. If a catastrophic event occurs that disables both the primary and replica cache, the cache is reconstructed using the most recent snapshot. Learn more about the [advantages](#) and [disadvantages](#) of RDB persistence.
- **AOF persistence** - When AOF (Append only file) persistence is configured, Azure Cache for Redis saves every write operation to a log that is saved at least once per second into an Azure Storage account. If a catastrophic event occurs that disables both the primary and replica cache, the cache is reconstructed using the stored write operations. Learn more about the [advantages](#) and [disadvantages](#) of AOF persistence.

Persistence writes Redis data into an Azure Storage account that you own and manage. You can configure from the **New Azure Cache for Redis** blade during cache creation and on the **Resource menu** for existing premium caches.

NOTE

Azure Storage automatically encrypts data when it is persisted. You can use your own keys for the encryption. For more information, see [Customer-managed keys with Azure Key Vault](#).

To create a premium cache, sign in to the [Azure portal](#) and click **Create a resource > Databases > Azure Cache for Redis**.

The screenshot shows the Azure portal's 'New' blade for creating a new resource. On the left, there's a sidebar with various service categories like All services, Favorites, and specific services like Dashboard, App Services, and Storage accounts. The 'Create a resource' button at the top left is highlighted with a red box. In the main area, there's a search bar labeled 'Search the Marketplace'. Below it, the 'Azure Marketplace' section has tabs for 'Featured' (which is selected) and 'See all'. Under 'Featured', there are several service cards: 'SQL Database' (Quickstart tutorial), 'SQL Data Warehouse' (Learn more), 'SQL Elastic database pool' (Learn more), 'Azure Database for MySQL (preview)' (Learn more), 'Azure Database for PostgreSQL (preview)' (Learn more), 'SQL Server 2017 Enterprise Windows Server 2016' (Learn more), 'Azure Cosmos DB' (Quickstart tutorial), 'Database as a service for MongoDB' (Learn more), and 'Redis Cache' (Learn more). The 'Databases' category in the center-left is also highlighted with a red box. The 'Redis Cache' card is specifically highlighted with a red box.

NOTE

In addition to creating caches in the Azure portal, you can also create them using Resource Manager templates, PowerShell, or Azure CLI. For more information about creating an Azure Cache for Redis, see [Create a cache](#).

To configure premium features, first select one of the premium pricing tiers in the **Pricing tier** drop-down list. For more information about each pricing tier, click **View full pricing details** and select a pricing tier from the **Choose your pricing tier** blade.

The screenshot shows two overlapping windows. On the left is the 'New Redis Cache' creation window, and on the right is the 'Choose your pricing tier' window.

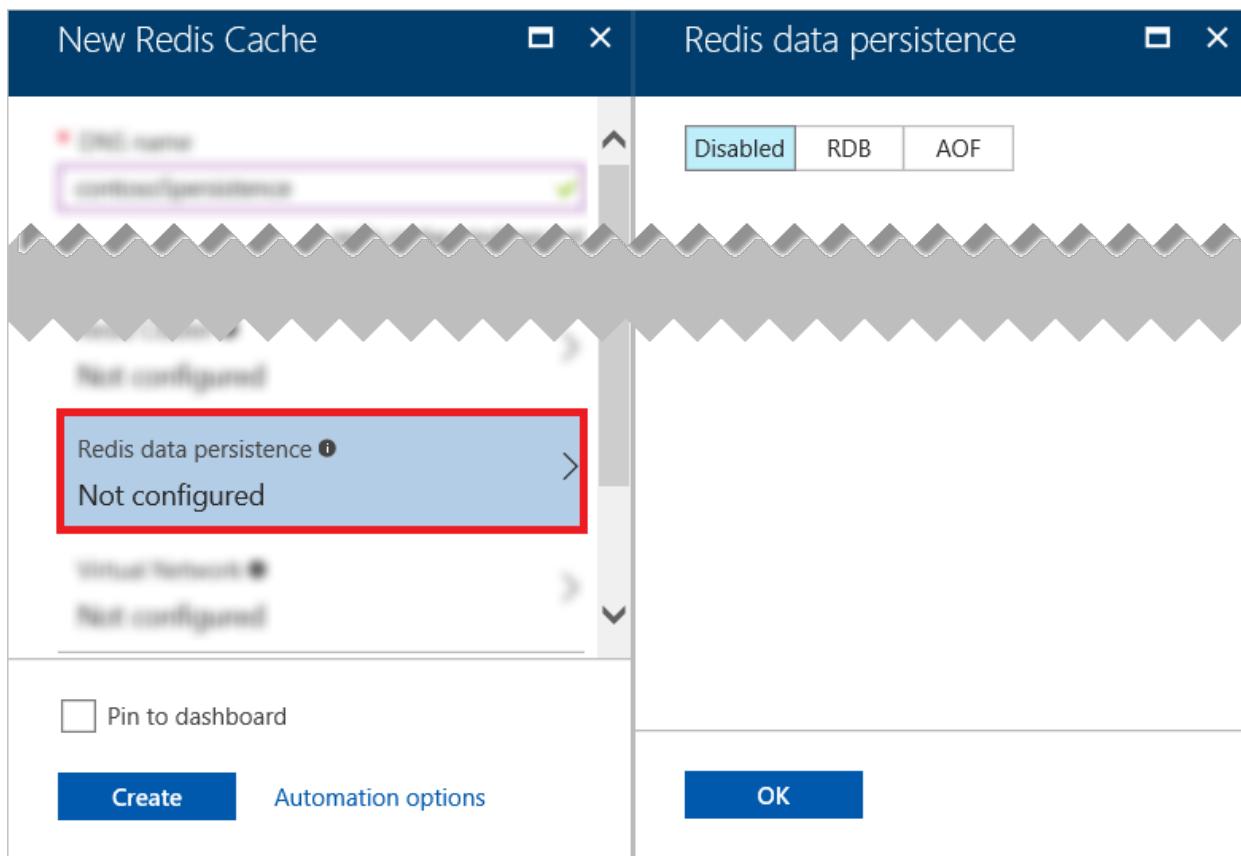
New Redis Cache (Left Window):

- Pricing tier:** Premium P1 (6 GB Cache, Replication) is selected. A red circle with '1' highlights this selection.
- Redis Cluster Configuration:** Redis Cluster is set to 'Not configured' for all three sections: Redis Cluster, Redis data persistence, and Virtual Network. These sections are highlighted with a red box.
- Ports:** The 'Unblock port 6379 (not SSL encrypted)' checkbox is unchecked.
- Buttons:** 'Create' and 'Automation options' buttons are at the bottom.

Choose your pricing tier (Right Window):

- Recommended Pricing Tiers:** P1 Premium, P2 Premium, and P3 Premium are highlighted with a red box. Each tier includes details like GB Cache, replication, network bandwidth, features, and SLA, along with its estimated monthly cost.
- Standard Pricing Tiers:** C0 Standard, C1 Standard, C2 Standard, C3 Standard, and C4 Standard are listed below.
- Select Button:** A red circle with '3' highlights the 'Select' button at the bottom of the pricing tier table.

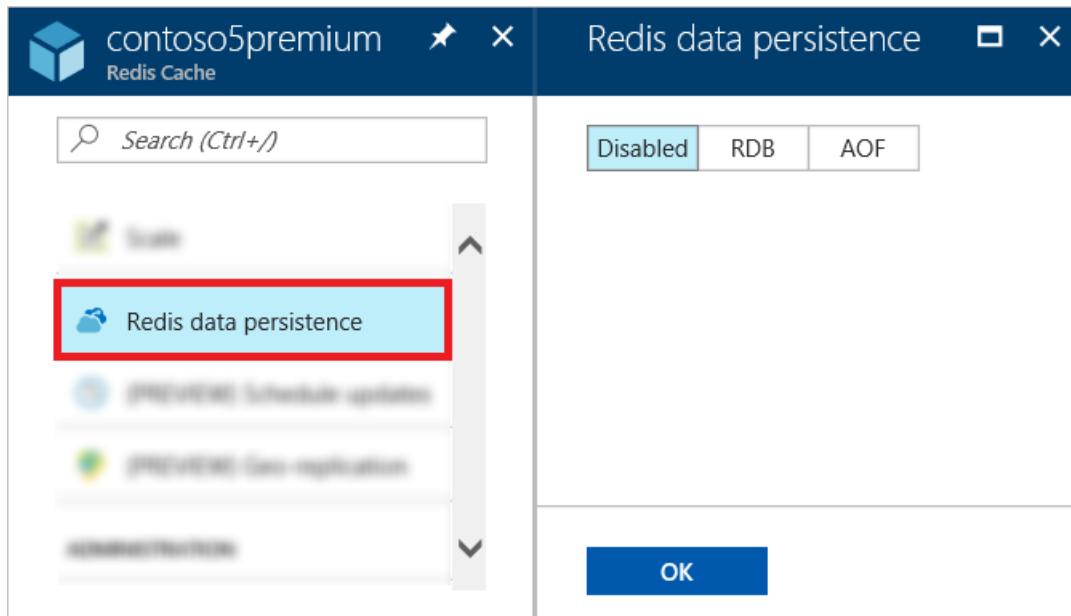
Once a premium pricing tier is selected, click **Redis persistence**.



The steps in the next section describe how to configure Redis persistence on your new premium cache. Once Redis persistence is configured, click **Create** to create your new premium cache with Redis persistence.

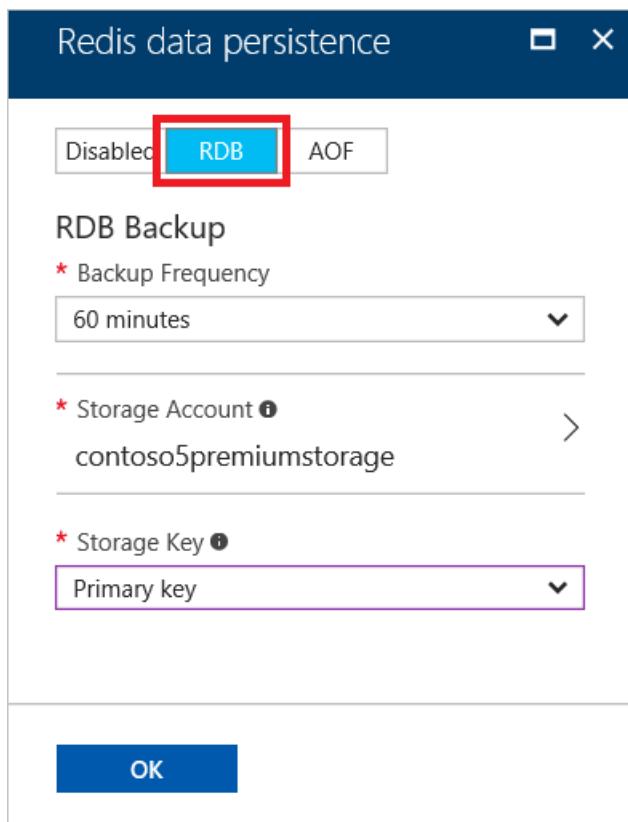
Enable Redis persistence

Redis persistence is enabled on the **Data persistence** blade by choosing either **RDB** or **AOF** persistence. For new caches, this blade is accessed during the cache creation process, as described in the previous section. For existing caches, the **Data persistence** blade is accessed from the **Resource menu** for your cache.



Configure RDB persistence

To enable RDB persistence, click **RDB**. To disable RDB persistence on a previously enabled premium cache, click **Disabled**.



To configure the backup interval, select a **Backup Frequency** from the drop-down list. Choices include **15 Minutes**, **30 minutes**, **60 minutes**, **6 hours**, **12 hours**, and **24 hours**. This interval starts counting down after the previous backup operation successfully completes and when it elapses a new backup is initiated.

Click **Storage Account** to select the storage account to use, and choose either the **Primary key** or **Secondary key** to use from the **Storage Key** drop-down. You must choose a storage account in the same region as the cache, and a **Premium Storage** account is recommended because premium storage has higher throughput.

IMPORTANT

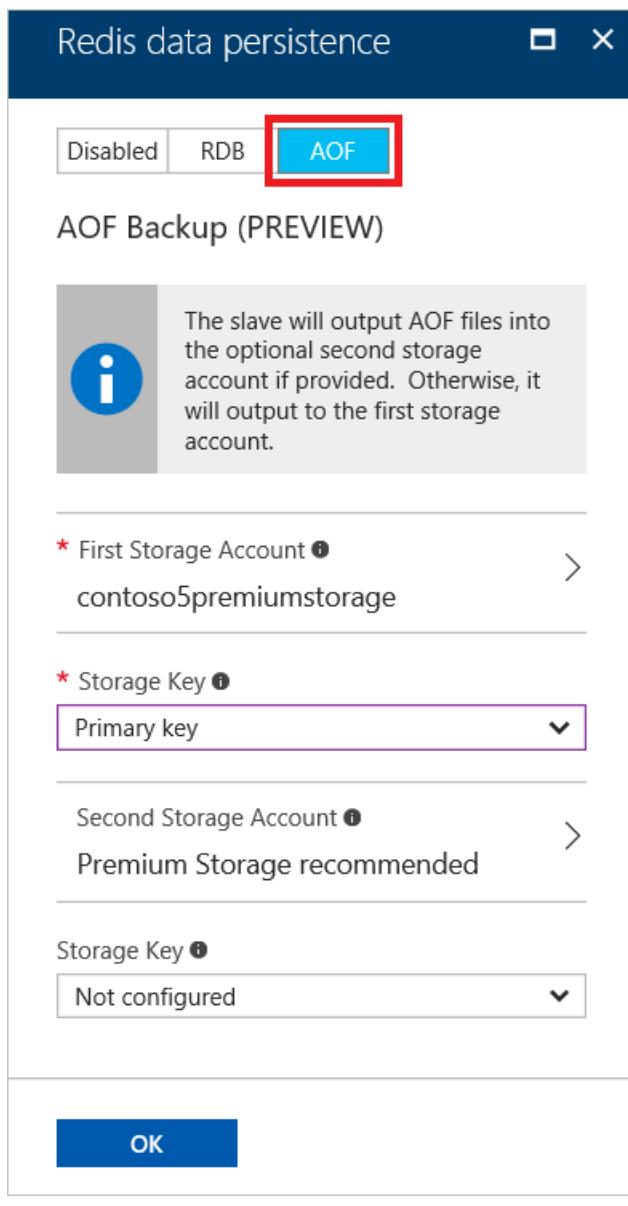
If the storage key for your persistence account is regenerated, you must reconfigure the desired key from the **Storage Key** drop-down.

Click **OK** to save the persistence configuration.

The next backup (or first backup for new caches) is initiated once the backup frequency interval elapses.

Configure AOF persistence

To enable AOF persistence, click **AOF**. To disable AOF persistence on a previously enabled premium cache, click **Disabled**.



To configure AOF persistence, specify a **First Storage Account**. This storage account must be in the same region as the cache, and a **Premium Storage** account is recommended because premium storage has higher throughput. You can optionally configure an additional storage account named **Second Storage Account**. If a second storage account is configured, the writes to the replica cache are written to this second storage account. For each configured storage account, choose either the **Primary key** or **Secondary key** to use from the **Storage Key** drop-down.

IMPORTANT

If the storage key for your persistence account is regenerated, you must reconfigure the desired key from the **Storage Key** drop-down.

When AOF persistence is enabled, write operations to the cache are saved to the designated storage account (or accounts if you have configured a second storage account). In the event of a catastrophic failure that takes down both the primary and replica cache, the stored AOF log is used to rebuild the cache.

Persistence FAQ

The following list contains answers to commonly asked questions about Azure Cache for Redis persistence.

- [Can I enable persistence on a previously created cache?](#)
- [Can I enable AOF and RDB persistence at the same time?](#)

- Which persistence model should I choose?
- What happens if I have scaled to a different size and a backup is restored that was made before the scaling operation?

RDB persistence

- Can I change the RDB backup frequency after I create the cache?
- Why if I have an RDB backup frequency of 60 minutes there is more than 60 minutes between backups?
- What happens to the old RDB backups when a new backup is made?

AOF persistence

- When should I use a second storage account?
- Does AOF persistence affect throughout, latency, or performance of my cache?
- How can I remove the second storage account?
- What is a rewrite and how does it affect my cache?
- What should I expect when scaling a cache with AOF enabled?
- How is my AOF data organized in storage?

Can I enable persistence on a previously created cache?

Yes, Redis persistence can be configured both at cache creation and on existing premium caches.

Can I enable AOF and RDB persistence at the same time?

No, you can enable only RDB or AOF, but not both at the same time.

Which persistence model should I choose?

AOF persistence saves every write to a log, which has some impact on throughput, compared with RDB persistence which saves backups based on the configured backup interval, with minimal impact on performance. Choose AOF persistence if your primary goal is to minimize data loss, and you can handle a decrease in throughput for your cache. Choose RDB persistence if you wish to maintain optimal throughput on your cache, but still want a mechanism for data recovery.

- Learn more about the [advantages](#) and [disadvantages](#) of RDB persistence.
- Learn more about the [advantages](#) and [disadvantages](#) of AOF persistence.

For more information on performance when using AOF persistence, see [Does AOF persistence affect throughout, latency, or performance of my cache?](#)

What happens if I have scaled to a different size and a backup is restored that was made before the scaling operation?

For both RDB and AOF persistence:

- If you have scaled to a larger size, there is no impact.
- If you have scaled to a smaller size, and you have a custom [databases](#) setting that is greater than the [databases limit](#) for your new size, data in those databases isn't restored. For more information, see [Is my custom databases setting affected during scaling?](#)
- If you have scaled to a smaller size, and there isn't enough room in the smaller size to hold all of the data from the last backup, keys will be evicted during the restore process, typically using the [allkeys-lru](#) eviction policy.

Can I change the RDB backup frequency after I create the cache?

Yes, you can change the backup frequency for RDB persistence on the **Data persistence** blade. For instructions, see [Configure Redis persistence](#).

Why if I have an RDB backup frequency of 60 minutes there is more than 60 minutes between backups?

The RDB persistence backup frequency interval does not start until the previous backup process has completed

successfully. If the backup frequency is 60 minutes and it takes a backup process 15 minutes to successfully complete, the next backup won't start until 75 minutes after the start time of the previous backup.

What happens to the old RDB backups when a new backup is made?

All RDB persistence backups except for the most recent one are automatically deleted. This deletion may not happen immediately but older backups are not persisted indefinitely.

When should I use a second storage account?

You should use a second storage account for AOF persistence when you believe you have higher than expected set operations on the cache. Setting up the secondary storage account helps ensure your cache doesn't reach storage bandwidth limits.

Does AOF persistence affect throughout, latency, or performance of my cache?

AOF persistence affects throughput by about 15% – 20% when the cache is below maximum load (CPU and Server Load both under 90%). There should not be latency issues when the cache is within these limits. However, the cache will reach these limits sooner with AOF enabled.

How can I remove the second storage account?

You can remove the AOF persistence secondary storage account by setting the second storage account to be the same as the first storage account. For instructions, see [Configure AOF persistence](#).

What is a rewrite and how does it affect my cache?

When the AOF file becomes large enough, a rewrite is automatically queued on the cache. The rewrite resizes the AOF file with the minimal set of operations needed to create the current data set. During rewrites, expect to reach performance limits sooner especially when dealing with large datasets. Rewrites occur less often as the AOF file becomes larger, but will take a significant amount of time when it happens.

What should I expect when scaling a cache with AOF enabled?

If the AOF file at the time of scaling is significantly large, then expect the scale operation to take longer than expected since it will be reloading the file after scaling has finished.

For more information on scaling, see [What happens if I have scaled to a different size and a backup is restored that was made before the scaling operation?](#)

How is my AOF data organized in storage?

Data stored in AOF files is divided into multiple page blobs per node to increase performance of saving the data to storage. The following table displays how many page blobs are used for each pricing tier:

PREMIUM TIER	BLOBS
P1	4 per shard
P2	8 per shard
P3	16 per shard
P4	20 per shard

When clustering is enabled, each shard in the cache has its own set of page blobs, as indicated in the previous table. For example, a P2 cache with three shards distributes its AOF file across 24 page blobs (8 blobs per shard, with 3 shards).

After a rewrite, two sets of AOF files exist in storage. Rewrites occur in the background and append to the first set of files, while set operations that are sent to the cache during the rewrite append to the second set. A backup is temporarily stored during rewrites in case of failure, but is promptly deleted after a rewrite finishes.

Next steps

Learn how to use more premium cache features.

- [Introduction to the Azure Cache for Redis Premium tier](#)

How to configure Virtual Network Support for a Premium Azure Cache for Redis

1/28/2020 • 13 minutes to read • [Edit Online](#)

Azure Cache for Redis has different cache offerings, which provide flexibility in the choice of cache size and features, including Premium tier features such as clustering, persistence, and virtual network support. A VNet is a private network in the cloud. When an Azure Cache for Redis instance is configured with a VNet, it is not publicly addressable and can only be accessed from virtual machines and applications within the VNet. This article describes how to configure virtual network support for a premium Azure Cache for Redis instance.

NOTE

Azure Cache for Redis supports both classic and Resource Manager VNets.

For information on other premium cache features, see [Introduction to the Azure Cache for Redis Premium tier](#).

Why VNet?

[Azure Virtual Network \(VNet\)](#) deployment provides enhanced security and isolation for your Azure Cache for Redis, as well as subnets, access control policies, and other features to further restrict access.

Virtual network support

Virtual Network (VNet) support is configured on the **New Azure Cache for Redis** blade during cache creation.

To create a premium cache, sign in to the [Azure portal](#) and click **Create a resource > Databases > Azure Cache for Redis**.

The screenshot shows the Azure Marketplace 'New' page. On the left, there's a sidebar with various service categories like All services, Favorites, and specific services like Dashboard, App Services, etc. A red box highlights the 'Create a resource' button at the top of the sidebar. The main area has tabs for 'Azure Marketplace' and 'See all'. Under 'Featured', there are several service cards. One card for 'Databases' is highlighted with a red box. Another card for 'Redis Cache' is also highlighted with a red box. Other cards include SQL Database, SQL Data Warehouse, SQL Elastic database pool, Azure Database for MySQL (preview), Azure Database for PostgreSQL (preview), SQL Server 2017 Enterprise Windows Server 2016, Azure Cosmos DB, Database as a service for MongoDB, and Redis Cache.

NOTE

In addition to creating caches in the Azure portal, you can also create them using Resource Manager templates, PowerShell, or Azure CLI. For more information about creating an Azure Cache for Redis, see [Create a cache](#).

To configure premium features, first select one of the premium pricing tiers in the **Pricing tier** drop-down list. For more information about each pricing tier, click **View full pricing details** and select a pricing tier from the **Choose your pricing tier** blade.

The screenshot shows two blades side-by-side. The left blade is titled 'New Redis Cache' and contains fields for DNS name (contoso5premium), Subscription (Prototype3), Resource group (Create new), Location (Central US), and Pricing tier (Premium P1 (6 GB Cache, Replication)). A red box labeled '1' highlights the 'Pricing tier' dropdown. The right blade is titled 'Choose your pricing tier' and lists various Redis cache plans across three categories: Premium, Standard, and Basic. The Premium section is highlighted with a red box labeled '2'. The P1 Premium plan is selected, showing 6 GB Cache, Replication, and a monthly cost of 412.92 USD/MONTH (ESTIMATED) PER SH... The bottom of the right blade has a red box labeled '3' around the 'Select' button.

Pricing Tier	Plan	Configuration	Cost
Premium	P1 Premium	6 GB Cache, Replication, Moderate network bandwidth, All Standard features, Data Persistence, Virtual Network, Redis Cluster, 99.9% SLA	412.92 USD/MONTH (ESTIMATED) PER SH...
	P2 Premium	13 GB Cache, Replication, Moderate network bandwidth, All Standard features, Data Persistence, Virtual Network, Redis Cluster, 99.9% SLA	825.84 USD/MONTH (ESTIMATED) PER SH...
	P3 Premium	26 GB Cache, Replication, High network bandwidth, All Standard features, Data Persistence, Virtual Network, Redis Cluster, 99.9% SLA	1,650.94 USD/MONTH (ESTIMATED) PER SH...
	P4 Premium	53 GB Cache, Replication, Highest network bandwidth, All Standard features, Data Persistence, Virtual Network, Redis Cluster, 99.9% SLA	3,303.36 USD/MONTH (ESTIMATED) PER SH...
	C0 Standard	250 MB Cache, Replication, Low network bandwidth, Shared infrastructure, SSL, Up to 256 connections, Configure Redis (key...), 99.9% SLA	40.92 USD/MONTH (ESTIMATED)
	C1 Standard	1 GB Cache, Replication, Low network bandwidth, Dedicated service, SSL, Up to 1,000 connections, Configure Redis (key...), 99.9% SLA	102.67 USD/MONTH (ESTIMATED)
	C2 Standard	2.5 GB Cache	
	C3 Standard	6 GB Cache	
	C4 Standard	13 GB Cache	

Once you have selected a premium pricing tier, you can configure Redis VNet integration by selecting a VNet that is in the same subscription and location as your cache. To use a new VNet, create it first by following the steps in [Create a virtual network using the Azure portal](#) or [Create a virtual network \(classic\) by using the Azure portal](#) and then return to the **New Azure Cache for Redis** blade to create and configure your premium cache.

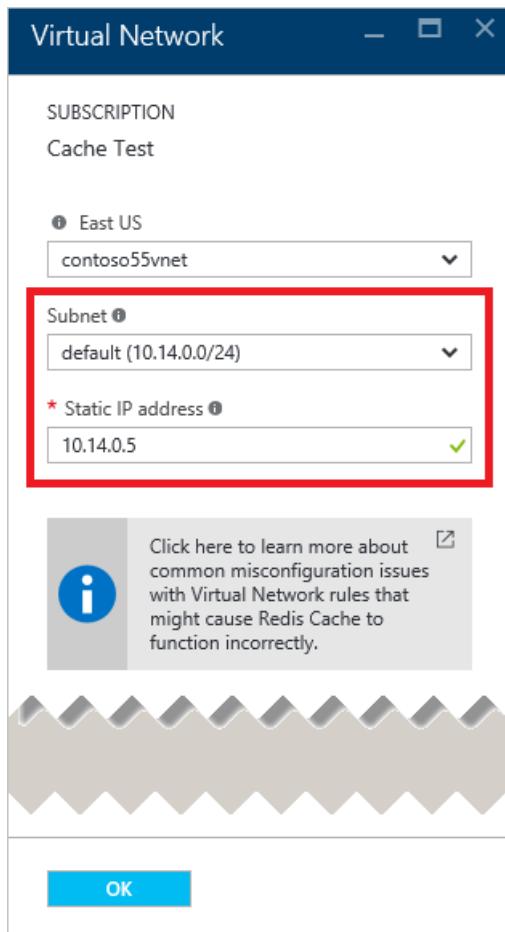
To configure the VNet for your new cache, click **Virtual Network** on the **New Azure Cache for Redis** blade, and select the desired VNet from the drop-down list.

The screenshot shows two overlapping Azure portal windows. The left window is titled "New Redis Cache" and contains fields for "Name" (contoso55redis), "Subscription" (Cache Test), "Resource group" (Create new, selected), "Location" (East US), "Size tier" (Premium 6 GB), "Redis Cluster" (Not configured), and "Redis data persistence" (Not configured). A red box highlights the "Virtual Network" section, which shows "Not configured". The right window is titled "Virtual Network" and shows the "SUBSCRIPTION" section with "Cache Test". It displays a list of virtual networks under "East US": "None" and "contoso55vnet" (selected). A tooltip below the list states: "with Virtual Network rules that might cause Redis Cache to function incorrectly." The "OK" button at the bottom right of the right window is also highlighted with a red box.

Select the desired subnet from the **Subnet** drop-down list. If desired, specify a **Static IP address**. The **Static IP address** field is optional, and if none is specified, one is chosen from the selected subnet.

IMPORTANT

When deploying an Azure Cache for Redis to a Resource Manager VNet, the cache must be in a dedicated subnet that contains no other resources except for Azure Cache for Redis instances. If an attempt is made to deploy an Azure Cache for Redis to a Resource Manager VNet to a subnet that contains other resources, the deployment fails.



IMPORTANT

Azure reserves some IP addresses within each subnet, and these addresses can't be used. The first and last IP addresses of the subnets are reserved for protocol conformance, along with three more addresses used for Azure services. For more information, see [Are there any restrictions on using IP addresses within these subnets?](#)

In addition to the IP addresses used by the Azure VNET infrastructure, each Redis instance in the subnet uses two IP addresses per shard and one additional IP address for the load balancer. A non-clustered cache is considered to have one shard.

After the cache is created, you can view the configuration for the VNet by clicking **Virtual Network** from the **Resource menu**.

The screenshot shows the Azure portal interface for managing a Redis Cache instance named 'contoso5premium'. The left sidebar lists several resources, and the main pane displays configuration details. A red box highlights the 'Virtual Network' section, which includes fields for 'STATIC IP ADDRESS' (set to '10.14.0.5') and 'HOST NAME' ('contoso5premium.redis.cache.windows.net'). Other visible settings include 'SUBNET' (set to 'default') and a link to learn more about recommended virtual network configuration.

To connect to your Azure Cache for Redis instance when using a VNet, specify the host name of your cache in the connection string as shown in the following example:

```
private static Lazy<ConnectionMultiplexer> lazyConnection = new Lazy<ConnectionMultiplexer>(() =>
{
    return
        ConnectionMultiplexer.Connect("contoso5premium.redis.cache.windows.net,abortConnect=false,ssl=true,password=pa
        ssword");
});

public static ConnectionMultiplexer Connection
{
    get
    {
        return lazyConnection.Value;
    }
}
```

Azure Cache for Redis VNet FAQ

The following list contains answers to commonly asked questions about the Azure Cache for Redis scaling.

- What are some common misconfiguration issues with Azure Cache for Redis and VNets?
- [How can I verify that my cache is working in a VNET?](#)
- When trying to connect to my Azure Cache for Redis in a VNET, why am I getting an error stating the remote certificate is invalid?
- [Can I use VNets with a standard or basic cache?](#)
- Why does creating an Azure Cache for Redis fail in some subnets but not others?
- [What are the subnet address space requirements?](#)
- [Do all cache features work when hosting a cache in a VNET?](#)

What are some common misconfiguration issues with Azure Cache for Redis and VNets?

When Azure Cache for Redis is hosted in a VNet, the ports in the following tables are used.

IMPORTANT

If the ports in the following tables are blocked, the cache may not function correctly. Having one or more of these ports blocked is the most common misconfiguration issue when using Azure Cache for Redis in a VNet.

- [Outbound port requirements](#)
- [Inbound port requirements](#)

Outbound port requirements

There are nine outbound port requirements. Outbound requests in these ranges are either outbound to other services necessary for the cache to function or internal to the Redis subnet for internode communication. For geo-replication, additional outbound requirements exist for communication between subnets of the primary and secondary cache.

PORT(S)	DIRECTION	TRANSPORT PROTOCOL	PURPOSE	LOCAL IP	REMOTE IP
80, 443	Outbound	TCP	Redis dependencies on Azure Storage/PKI (Internet)	(Redis subnet)	*
443	Outbound	TCP	Redis dependency on Azure Key Vault	(Redis subnet)	AzureKeyVault ¹
53	Outbound	TCP/UDP	Redis dependencies on DNS (Internet/VNet)	(Redis subnet)	168.63.129.16 and 169.254.169.254 ² and any custom DNS server for the subnet ³
8443	Outbound	TCP	Internal communications for Redis	(Redis subnet)	(Redis subnet)
10221-10231	Outbound	TCP	Internal communications for Redis	(Redis subnet)	(Redis subnet)
20226	Outbound	TCP	Internal communications for Redis	(Redis subnet)	(Redis subnet)
13000-13999	Outbound	TCP	Internal communications for Redis	(Redis subnet)	(Redis subnet)
15000-15999	Outbound	TCP	Internal communications for Redis and Geo-Replication	(Redis subnet)	(Redis subnet) (Geo-replica peer subnet)

PORT(S)	DIRECTION	TRANSPORT PROTOCOL	PURPOSE	LOCAL IP	REMOTE IP
6379-6380	Outbound	TCP	Internal communications for Redis	(Redis subnet)	(Redis subnet)

¹ You can use the service tag 'AzureKeyVault' with Resource Manager Network Security Groups.

² These IP addresses owned by Microsoft are used to address the Host VM which serves Azure DNS.

³ Not needed for subnets with no custom DNS server, or newer redis caches that ignore custom DNS.

Geo-replication peer port requirements

If you are using georeplication between caches in Azure Virtual Networks, please note that the recommended configuration is to unblock ports 15000-15999 for the whole subnet in both inbound AND outbound directions to both caches, so that all the replica components in the subnet can communicate directly with each other even in the event of a future geo-failover.

Inbound port requirements

There are eight inbound port range requirements. Inbound requests in these ranges are either inbound from other services hosted in the same VNET or internal to the Redis subnet communications.

PORT(S)	DIRECTION	TRANSPORT PROTOCOL	PURPOSE	LOCAL IP	REMOTE IP
6379, 6380	Inbound	TCP	Client communication to Redis, Azure load balancing	(Redis subnet)	(Redis subnet), Virtual Network, Azure Load Balancer ¹
8443	Inbound	TCP	Internal communications for Redis	(Redis subnet)	(Redis subnet)
8500	Inbound	TCP/UDP	Azure load balancing	(Redis subnet)	Azure Load Balancer
10221-10231	Inbound	TCP	Internal communications for Redis	(Redis subnet)	(Redis subnet), Azure Load Balancer
13000-13999	Inbound	TCP	Client communication to Redis Clusters, Azure load balancing	(Redis subnet)	Virtual Network, Azure Load Balancer
15000-15999	Inbound	TCP	Client communication to Redis Clusters, Azure load Balancing, and Geo-Replication	(Redis subnet)	Virtual Network, Azure Load Balancer, (Geo-replica peer subnet)
16001	Inbound	TCP/UDP	Azure load balancing	(Redis subnet)	Azure Load Balancer

PORT(S)	DIRECTION	TRANSPORT PROTOCOL	PURPOSE	LOCAL IP	REMOTE IP
20226	Inbound	TCP	Internal communications for Redis	(Redis subnet)	(Redis subnet)

¹ You can use the Service Tag 'AzureLoadBalancer' (Resource Manager) (or 'AZURE_LOADBALANCER' for classic) for authoring the NSG rules.

Additional VNET network connectivity requirements

There are network connectivity requirements for Azure Cache for Redis that may not be initially met in a virtual network. Azure Cache for Redis requires all the following items to function properly when used within a virtual network.

- Outbound network connectivity to Azure Storage endpoints worldwide. This includes endpoints located in the same region as the Azure Cache for Redis instance, as well as storage endpoints located in **other** Azure regions. Azure Storage endpoints resolve under the following DNS domains: *table.core.windows.net*, *blob.core.windows.net*, *queue.core.windows.net*, and *file.core.windows.net*.
- Outbound network connectivity to *ocsp.msocsp.com*, *mscrl.microsoft.com*, and *crl.microsoft.com*. This connectivity is needed to support SSL functionality.
- The DNS configuration for the virtual network must be capable of resolving all of the endpoints and domains mentioned in the earlier points. These DNS requirements can be met by ensuring a valid DNS infrastructure is configured and maintained for the virtual network.
- Outbound network connectivity to the following Azure Monitoring endpoints, which resolve under the following DNS domains: *shoebox2-black.shoebox2.metrics.nsatc.net*, *north-prod2.prod2.metrics.nsatc.net*, *azglobal-black.azglobal.metrics.nsatc.net*, *shoebox2-red.shoebox2.metrics.nsatc.net*, *east-prod2.prod2.metrics.nsatc.net*, *azglobal-red.azglobal.metrics.nsatc.net*.

How can I verify that my cache is working in a VNET?

IMPORTANT

When connecting to an Azure Cache for Redis instance that is hosted in a VNET, your cache clients must be in the same VNET or in a VNET with VNET peering enabled within the same Azure region. Global VNET Peering isn't currently supported. This includes any test applications or diagnostic pinging tools. Regardless of where the client application is hosted, Network security groups must be configured such that the client's network traffic is allowed to reach the Redis instance.

Once the port requirements are configured as described in the previous section, you can verify that your cache is working by performing the following steps.

- Reboot** all of the cache nodes. If all of the required cache dependencies can't be reached (as documented in [Inbound port requirements](#) and [Outbound port requirements](#)), the cache won't be able to restart successfully.
- Once the cache nodes have restarted (as reported by the cache status in the Azure portal), you can perform the following tests:
 - ping the cache endpoint (using port 6380) from a machine that is within the same VNET as the cache, using **tcping**. For example:

```
tcping.exe contosocache.redis.cache.windows.net 6380
```

If the **tcping** tool reports that the port is open, the cache is available for connection from clients in the VNET.

- Another way to test is to create a test cache client (which could be a simple console application using StackExchange.Redis) that connects to the cache and adds and retrieves some items from the cache.

Install the sample client application onto a VM that is in the same VNET as the cache and run it to verify connectivity to the cache.

When trying to connect to my Azure Cache for Redis in a VNET, why am I getting an error stating the remote certificate is invalid?

When trying to connect to an Azure Cache for Redis in a VNET, you see a certificate validation error such as this:

```
{"No connection is available to service this operation: SET mykey; The remote certificate is invalid according to the validation procedure.; ..."}
```

The cause could be you are connecting to the host by the IP address. We recommend using the hostname. In other words, use the following:

```
[mycachename].redis.windows.net:6380,password=xxxxxxxxxxxxxxxxxxxx,ssl=True,abortConnect=False
```

Avoid using the IP address similar to the following connection string:

```
10.128.2.84:6380,password=xxxxxxxxxxxxxxxxxxxx,ssl=True,abortConnect=False
```

If you are unable to resolve the DNS name, some client libraries include configuration options like `sslHost` which is provided by the StackExchange.Redis client. This allows you to override the hostname used for certificate validation. For example:

```
10.128.2.84:6380,password=xxxxxxxxxxxxxxxxxxxx,ssl=True,abortConnect=False;sslHost=[mycachename].redis.windows.net
```

Can I use VNets with a standard or basic cache?

VNets can only be used with premium caches.

Why does creating an Azure Cache for Redis fail in some subnets but not others?

If you are deploying an Azure Cache for Redis to a Resource Manager VNet, the cache must be in a dedicated subnet that contains no other resource type. If an attempt is made to deploy an Azure Cache for Redis to a Resource Manager VNet subnet that contains other resources, the deployment fails. You must delete the existing resources inside the subnet before you can create a new Azure Cache for Redis.

You can deploy multiple types of resources to a classic VNet as long as you have enough IP addresses available.

What are the subnet address space requirements?

Azure reserves some IP addresses within each subnet, and these addresses can't be used. The first and last IP addresses of the subnets are reserved for protocol conformance, along with three more addresses used for Azure services. For more information, see [Are there any restrictions on using IP addresses within these subnets?](#)

In addition to the IP addresses used by the Azure VNET infrastructure, each Redis instance in the subnet uses two IP addresses per shard and one additional IP address for the load balancer. A non-clustered cache is considered to have one shard.

Do all cache features work when hosting a cache in a VNET?

When your cache is part of a VNET, only clients in the VNET can access the cache. As a result, the following cache management features don't work at this time.

- Redis Console - Because Redis Console runs in your local browser, which is outside the VNET, it can't connect to your cache.

Use ExpressRoute with Azure Cache for Redis

Customers can connect an [Azure ExpressRoute](#) circuit to their virtual network infrastructure, thus extending their on-premises network to Azure.

By default, a newly created ExpressRoute circuit does not perform forced tunneling (advertisement of a default

route, 0.0.0.0/0) on a VNET. As a result, outbound Internet connectivity is allowed directly from the VNET and client applications are able to connect to other Azure endpoints including Azure Cache for Redis.

However a common customer configuration is to use forced tunneling (advertise a default route) which forces outbound Internet traffic to instead flow on-premises. This traffic flow breaks connectivity with Azure Cache for Redis if the outbound traffic is then blocked on-premises such that the Azure Cache for Redis instance is not able to communicate with its dependencies.

The solution is to define one (or more) user-defined routes (UDRs) on the subnet that contains the Azure Cache for Redis. A UDR defines subnet-specific routes that will be honored instead of the default route.

If possible, it is recommended to use the following configuration:

- The ExpressRoute configuration advertises 0.0.0.0/0 and by default force tunnels all outbound traffic on-premises.
- The UDR applied to the subnet containing the Azure Cache for Redis defines 0.0.0.0/0 with a working route for TCP/IP traffic to the public internet; for example by setting the next hop type to 'Internet'.

The combined effect of these steps is that the subnet level UDR takes precedence over the ExpressRoute forced tunneling, thus ensuring outbound Internet access from the Azure Cache for Redis.

Connecting to an Azure Cache for Redis instance from an on-premises application using ExpressRoute is not a typical usage scenario due to performance reasons (for best performance Azure Cache for Redis clients should be in the same region as the Azure Cache for Redis).

IMPORTANT

The routes defined in a UDR **must** be specific enough to take precedence over any routes advertised by the ExpressRoute configuration. The following example uses the broad 0.0.0.0/0 address range, and as such can potentially be accidentally overridden by route advertisements using more specific address ranges.

WARNING

Azure Cache for Redis is not supported with ExpressRoute configurations that **incorrectly cross-advertise routes from the public peering path to the private peering path**. ExpressRoute configurations that have public peering configured, receive route advertisements from Microsoft for a large set of Microsoft Azure IP address ranges. If these address ranges are incorrectly cross-advertised on the private peering path, the result is that all outbound network packets from the Azure Cache for Redis instance's subnet are incorrectly force-tunneled to a customer's on-premises network infrastructure. This network flow breaks Azure Cache for Redis. The solution to this problem is to stop cross-advertising routes from the public peering path to the private peering path.

Background information on user-defined routes is available in this [overview](#).

For more information about ExpressRoute, see [ExpressRoute technical overview](#).

Next steps

Learn how to use more premium cache features.

- [Introduction to the Azure Cache for Redis Premium tier](#)

How to set up geo-replication for Azure Cache for Redis

11/15/2019 • 8 minutes to read • [Edit Online](#)

Geo-replication provides a mechanism for linking two Premium tier Azure Cache for Redis instances. One cache is chosen as the primary linked cache, and the other as the secondary linked cache. The secondary linked cache becomes read-only, and data written to the primary cache is replicated to the secondary linked cache. This functionality can be used to replicate a cache across Azure regions. This article provides a guide to configuring geo-replication for your Premium tier Azure Cache for Redis instances.

Geo-replication prerequisites

To configure geo-replication between two caches, the following prerequisites must be met:

- Both caches are [Premium tier](#) caches.
- Both caches are in the same Azure subscription.
- The secondary linked cache is either the same cache size or a larger cache size than the primary linked cache.
- Both caches are created and in a running state.

Some features aren't supported with geo-replication:

- Persistence isn't supported with geo-replication.
- Clustering is supported if both caches have clustering enabled and have the same number of shards.
- Caches in the same VNET are supported.
- Caches in different VNETs are supported with caveats. See [Can I use geo-replication with my caches in a VNET?](#) for more information.

After geo-replication is configured, the following restrictions apply to your linked cache pair:

- The secondary linked cache is read-only; you can read from it, but you can't write any data to it.
- Any data that was in the secondary linked cache before the link was added is removed. If the geo-replication is later removed however, the replicated data remains in the secondary linked cache.
- You can't [scale](#) either cache while the caches are linked.
- You can't [change the number of shards](#) if the cache has clustering enabled.
- You can't enable persistence on either cache.
- You can [Export](#) from either cache.
- You can't [Import](#) into the secondary linked cache.
- You can't delete either linked cache, or the resource group that contains them, until you unlink the caches. For more information, see [Why did the operation fail when I tried to delete my linked cache?](#)
- If the caches are in different regions, network egress costs apply to the data moved across regions. For more information, see [How much does it cost to replicate my data across Azure regions?](#)
- Automatic failover doesn't occur between the primary and secondary linked cache. For more information and information on how to failover a client application, see [How does failing over to the secondary linked cache work?](#)

Add a geo-replication link

1. To link two caches together for geo-replication, first click **Geo-replication** from the Resource menu of the

cache that you intend to be the primary linked cache. Next, click **Add cache replication link** from the **Geo-replication** blade.

The screenshot shows the 'Geo-replication' blade for a Redis Cache instance named 'contoso-geo1'. The left sidebar has a 'SETTINGS' section with various options like 'General settings', 'Redis Cache settings', 'Logs', etc., and the 'Geo-replication' option is highlighted with a red box and a red number '1'. At the top right, there's a search bar, an 'Add cache replication link' button with a red box and a red number '2', and an 'Unlink caches' button. Below the search bar is a world map with a green dot representing the primary cache's location. At the bottom, there's a table with columns 'LINKED CACHE', 'ROLE', 'LINKED CACHE LOCATION', and 'LINK STATUS', showing 'No results'.

- Click the name of your intended secondary cache from the **Compatible caches** list. If your secondary cache isn't displayed in the list, verify that the [Geo-replication prerequisites](#) for the secondary cache are met. To filter the caches by region, click the region in the map to display only those caches in the **Compatible caches** list.

This screenshot shows the 'COMPATIBLE CACHES' list. It has two columns: 'COMPATIBLE CACHES' and 'LOCATION'. The first row shows 'contoso-geo2' in the 'COMPATIBLE CACHES' column and 'Central US' in the 'LOCATION' column. A red box highlights 'contoso-geo2'.

You can also start the linking process or view details about the secondary cache by using the context menu.

This screenshot shows the context menu for 'contoso-geo2'. It has two items: 'View cache' and 'Link as secondary'. A red box highlights the '...' button next to 'contoso-geo2'.

- Click **Link** to link the two caches together and begin the replication process.

contoso-geo2

PRIMARY CACHE
contoso-geo1

PRIMARY REGION
East US

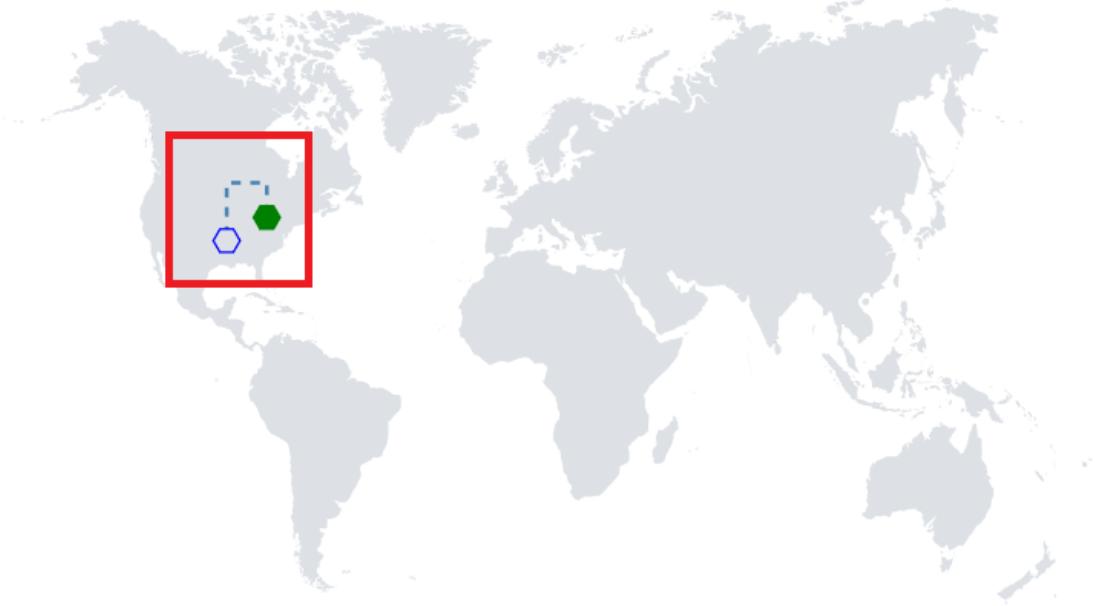
SECONDARY CACHE
contoso-geo2

SECONDARY REGION
Central US

Link

4. You can view the progress of the replication process on the **Geo-replication** blade.

Add cache replication link Unlink caches



LINKED CACHE	ROLE	LINKED CACHE LOCATION	LINK STATUS
contoso-geo2	Secondary	Central US	Creating

You can also view the linking status on the **Overview** blade for both the primary and secondary caches.

The screenshot shows the Azure portal interface for a Redis Cache named 'contoso-geo1'. On the left, there's a search bar and a sidebar with options like 'Overview', 'Metrics', 'Logs', and 'Logs (Preview)'. On the right, under 'Essentials', it shows the resource group 'contosogroup', status 'Status', linking information, location 'East US', and subscription details. A red box labeled '1' is around the 'Overview' tab, and another red box labeled '2' is around the 'Linking' section.

Once the replication process is complete, the **Link status** changes to **Succeeded**.

LINKED CACHE	ROLE	LINKED CACHE LOCATION	LINK STATUS
contoso-geo2	Secondary	Central US	Succeeded

The primary linked cache remains available for use during the linking process. The secondary linked cache isn't available until the linking process completes.

Remove a geo-replication link

- To remove the link between two caches and stop geo-replication, click **Unlink caches** from the **Geo-replication** blade.

The screenshot shows the 'Geo-replication' blade in the Azure portal. At the top, there are buttons for 'Add cache replication link' and 'Unlink caches' (which is highlighted with a red box labeled '1'). Below is a world map with two locations marked: one in North America (blue hexagon) and one in Europe (green hexagon), connected by a line. A table below the map shows the replication status:

LINKED CACHE	ROLE	LINKED CACHE LOCATION	LINK STATUS
contoso-geo2	Secondary	Central US	Succeeded

When the unlinking process completes, the secondary cache is available for both reads and writes.

NOTE

When the geo-replication link is removed, the replicated data from the primary linked cache remains in the secondary cache.

Geo-replication FAQ

- [Can I use geo-replication with a Standard or Basic tier cache?](#)
- [Is my cache available for use during the linking or unlinking process?](#)
- [Can I link more than two caches together?](#)
- [Can I link two caches from different Azure subscriptions?](#)
- [Can I link two caches with different sizes?](#)
- [Can I use geo-replication with clustering enabled?](#)
- [Can I use geo-replication with my caches in a VNET?](#)
- [What is the replication schedule for Redis geo-replication?](#)
- [How long does geo-replication replication take?](#)
- [Is the replication recovery point guaranteed?](#)
- [Can I use PowerShell or Azure CLI to manage geo-replication?](#)
- [How much does it cost to replicate my data across Azure regions?](#)
- [Why did the operation fail when I tried to delete my linked cache?](#)
- [What region should I use for my secondary linked cache?](#)
- [How does failing over to the secondary linked cache work?](#)

Can I use geo-replication with a Standard or Basic tier cache?

No, geo-replication is only available for Premium tier caches.

Is my cache available for use during the linking or unlinking process?

- When linking, the primary linked cache remains available while the linking process completes.
- When linking, the secondary linked cache isn't available until the linking process completes.
- When unlinking, both caches remain available while the unlinking process completes.

Can I link more than two caches together?

No, you can only link two caches together.

Can I link two caches from different Azure subscriptions?

No, both caches must be in the same Azure subscription.

Can I link two caches with different sizes?

Yes, as long as the secondary linked cache is larger than the primary linked cache.

Can I use geo-replication with clustering enabled?

Yes, as long as both caches have the same number of shards.

Can I use geo-replication with my caches in a VNET?

Yes, geo-replication of caches in VNETs is supported with caveats:

- Geo-replication between caches in the same VNET is supported.
- Geo-replication between caches in different VNETs is also supported.
 - If the VNETs are in the same region, you can connect them using [VNET peering](#) or a [VPN Gateway VNET-to-VNET connection](#).
 - If the VNETs are in different regions, geo-replication using VNET peering isn't supported because of a constraint with Basic internal load balancers. For more information about VNET peering constraints, see

[Virtual Network - Peering - Requirements and constraints](#). The recommended solution is to use a VPN Gateway VNET-to-VNET connection.

Using [this Azure template](#), you can quickly deploy two geo-replicated caches into a VNET connected with a VPN Gateway VNET-to-VNET connection.

What is the replication schedule for Redis geo-replication?

Replication is continuous and asynchronous and doesn't happen on a specific schedule. All the writes done to the primary are instantaneously and asynchronously replicated on the secondary.

How long does geo-replication replication take?

Replication is incremental, asynchronous, and continuous and the time taken isn't much different from the latency across regions. Under certain circumstances, the secondary cache may be required to do a full sync of the data from the primary. The replication time in this case is dependent on number of factors like: load on the primary cache, available network bandwidth, and inter-region latency. We have found replication time for a full 53-GB geo-replicated pair can be anywhere between 5 to 10 minutes.

Is the replication recovery point guaranteed?

For caches in a geo-replicated mode, persistence is disabled. If a geo-replicated pair is unlinked, such as a customer-initiated failover, the secondary linked cache keeps its synced data up to that point of time. No recovery point is guaranteed in such situations.

To obtain a recovery point, [Export](#) from either cache. You can later [Import](#) into the primary linked cache.

Can I use PowerShell or Azure CLI to manage geo-replication?

Yes, geo-replication can be managed using the Azure portal, PowerShell, or Azure CLI. For more information, see the [PowerShell docs](#) or [Azure CLI docs](#).

How much does it cost to replicate my data across Azure regions?

When using geo-replication, data from the primary linked cache is replicated to the secondary linked cache. There's no charge for the data transfer if the two linked caches are in the same region. If the two linked caches are in different regions, the data transfer charge is the network egress cost of data moving across either region. For more information, see [Bandwidth Pricing Details](#).

Why did the operation fail when I tried to delete my linked cache?

Geo-replicated caches and their resource groups can't be deleted while linked until you remove the geo-replication link. If you attempt to delete the resource group that contains one or both of the linked caches, the other resources in the resource group are deleted, but the resource group stays in the `deleting` state and any linked caches in the resource group remain in the `running` state. To completely delete the resource group and the linked caches within it, unlink the caches as described in [Remove a geo-replication link](#).

What region should I use for my secondary linked cache?

In general, it's recommended for your cache to exist in the same Azure region as the application that accesses it. For applications with separate primary and fallback regions, it's recommended your primary and secondary caches exist in those same regions. For more information about paired regions, see [Best Practices – Azure Paired regions](#).

How does failing over to the secondary linked cache work?

Automatic failover across Azure regions isn't supported for geo-replicated caches. In a disaster-recovery scenario, customers should bring up the entire application stack in a coordinated manner in their backup region. Letting individual application components decide when to switch to their backups on their own can negatively impact performance. One of the key benefits of Redis is that it's a very low-latency store. If the customer's main application is in a different region than its cache, the added round-trip time would have a noticeable impact on performance. For this reason, we avoid failing over automatically because of transient availability issues.

To start a customer-initiated failover, first unlink the caches. Then, change your Redis client to use the connection endpoint of the (formerly linked) secondary cache. When the two caches are unlinked, the secondary cache becomes a regular read-write cache again and accepts requests directly from Redis clients.

Next steps

Learn more about the [Azure Cache for Redis Premium tier](#).

Prepay for Azure Cache for Redis compute resources with reserved capacity

2/21/2020 • 5 minutes to read • [Edit Online](#)

Azure Cache for Redis now helps you save money by prepaying for compute resources compared to pay-as-you-go prices. With Azure Cache for Redis reserved capacity, you make an upfront commitment on cache for a one or three year period to get a significant discount on the compute costs. To purchase Azure Cache for Redis reserved capacity, you need to specify the Azure region, service tier, and term.

You do not need to assign the reservation to specific Azure Cache for Redis instances. An already running Azure Cache for Redis or ones that are newly deployed will automatically get the benefit of reserved pricing, up to the reserved cache size. By purchasing a reservation, you are pre-paying for the compute costs for a period of one or three years. As soon as you buy a reservation, the Azure Cache for Redis compute charges that match the reservation attributes are no longer charged at the pay-as-you go rates. A reservation does not cover networking or storage charges associated with the cache. At the end of the reservation term, the billing benefit expires and the Azure Cache for Redis is billed at the pay-as-you go price. Reservations do not auto-renew. For pricing information, see the [Azure Cache for Redis reserved capacity offering](#).

You can buy Azure Cache for Redis reserved capacity in the [Azure portal](#). To buy the reserved capacity:

- You must be in the owner role for at least one Enterprise or individual subscription with pay-as-you-go rates.
- For Enterprise subscriptions, **Add Reserved Instances** must be enabled in the [EA portal](#). Or, if that setting is disabled, you must be an EA Admin on the subscription.
- For Cloud Solution Provider (CSP) program, only the admin agents or sales agents can purchase Azure Cache for Redis reserved capacity.

For the details on how enterprise customers and Pay-As-You-Go customers are charged for reservation purchases, see [understand Azure reservation usage for your Enterprise enrollment](#) and [understand Azure reservation usage for your Pay-As-You-Go subscription](#).

Determine the right cache size before purchase

The size of reservation should be based on the total amount of compute used by the existing or soon-to-be-deployed cache within a specific region and using the same service tier.

For example, let's suppose that you are running one general purpose, Gen5 – 32 vCore cache, and two memory optimized, Gen5 – 16 vCore caches. Further, let's suppose that you plan to deploy within the next month an additional general purpose, Gen5 – 32 vCore database server, and one memory optimized, Gen5 – 16 vCore database server. Let's suppose that you know that you will need these resources for at least 1 year. In this case, you should purchase a 64 (2x32) vCores, 1 year reservation for single database general purpose - Gen5 and a 48 (2x16 + 16) vCore 1 year reservation for single database memory optimized - Gen5

Buy Azure Cache for Redis reserved capacity

1. Sign in to the [Azure portal](#).
2. Select **All services > Reservations**.
3. Select **Add** and then in the Purchase reservations pane, select **Azure Cache for Redis** to purchase a new reservation for your caches.
4. Fill-in the required fields. Existing or new databases that match the attributes you select qualify to get the reserved capacity discount. The actual number of your Azure Cache for Redis instances that get the discount

depend on the scope and quantity selected.

Select the product you want to purchase

Reserved Instances for Azure Redis Cache provide significant discounts (up to 55%) over pay-as-you-go prices, by allowing you to pre purchase instances for 1 or 3 year terms. [Learn More](#)

Scope * Shared Billing subscription * Finance App - Test

Region : West US Billing frequency : Monthly Term : Select a value Reset filters

Name	Cache name	Region	Term	Billing frequency
Azure Redis Cache Premium	P1	West US	One Year	Monthly
Azure Redis Cache Premium	P1	West US	Three Years	Monthly
Azure Redis Cache Premium	P2	West US	One Year	Monthly
Azure Redis Cache Premium	P2	West US	Three Years	Monthly
Azure Redis Cache Premium	P3	West US	One Year	Monthly
Azure Redis Cache Premium	P3	West US	Three Years	Monthly
Azure Redis Cache Premium	P4	West US	One Year	Monthly
Azure Redis Cache Premium	P4	West US	Three Years	Monthly
Azure Redis Cache Premium	P5	West US	One Year	Monthly
Azure Redis Cache Premium	P5	West US	Three Years	Monthly

Select Cancel Monthly price per node: <Price>
55% Estimated savings

The following table describes required fields.

FIELD	DESCRIPTION
Subscription	<p>The subscription used to pay for the Azure Cache for Redis reserved capacity reservation. The payment method on the subscription is charged the upfront costs for the Azure Cache for Redis reserved capacity reservation. The subscription type must be an enterprise agreement (offer numbers: MS-AZR-0017P or MS-AZR-0148P) or an individual agreement with pay-as-you-go pricing (offer numbers: MS-AZR-0003P or MS-AZR-0023P). For an enterprise subscription, the charges are deducted from the enrollment's monetary commitment balance or charged as overage. For an individual subscription with pay-as-you-go pricing, the charges are billed to the credit card or invoice payment method on the subscription.</p>
Scope	<p>The reservation's scope can cover one subscription or multiple subscriptions (shared scope). If you select:</p> <p>Shared, the reservation discount is applied to Azure Cache for Redis instances running in any subscriptions within your billing context. For enterprise customers, the shared scope is the enrollment and includes all subscriptions within the enrollment. For Pay-As-You-Go customers, the shared scope is all Pay-As-You-Go subscriptions created by the account administrator.</p> <p>Single subscription, the reservation discount is applied to Azure Cache for Redis instances in this subscription.</p> <p>Single resource group, the reservation discount is applied to Azure Cache for Redis instances in the selected subscription and the selected resource group within that subscription.</p>
Region	The Azure region that's covered by the Azure Cache for Redis reserved capacity reservation.

FIELD	DESCRIPTION
Pricing tier	The service tier for the Azure Cache for Redis servers.
Term	One year or three years
Quantity	The amount of compute resources being purchased within the Azure Cache for Redis reserved capacity reservation. The quantity is a number of caches in the selected Azure region and service tier that are being reserved and will get the billing discount. For example, if you are running or planning to run an Azure Cache for Redis servers with the total cache capacity of 26 GB in the East US region, then you would specify quantity as 26 to maximize the benefit for all caches.

Cancel, exchange, or refund reservations

You can cancel, exchange, or refund reservations with certain limitations. For more information, see [Self-service exchanges and refunds for Azure Reservations](#).

Cache size flexibility

Cache size flexibility helps you scale up or down within a service tier and region, without losing the reserved capacity benefit.

Need help? Contact us

If you have questions or need help, [create a support request](#).

Next steps

The reservation discount is applied automatically to the Azure Cache for Redis instances that match the reservation scope and attributes. You can update the scope of the reservation through the Azure portal, PowerShell, Azure CLI, or the API.

- To learn how reserved capacity discounts are applied to Azure Cache for Redis, see [Understand the Azure reservation discount](#)
- To learn more about Azure Reservations, see the following articles:
 - [What are Azure Reservations?](#)
 - [Manage Azure Reservations](#)
 - [Understand Azure Reservations discount](#)
 - [Understand reservation usage for your Pay-As-You-Go subscription](#)
 - [Understand reservation usage for your Enterprise enrollment](#)
 - [Azure Reservations in Partner Center Cloud Solution Provider \(CSP\) program](#)

Migrate from Managed Cache Service to Azure Cache for Redis

11/15/2019 • 13 minutes to read • [Edit Online](#)

Migrating your applications that use Azure Managed Cache Service to Azure Cache for Redis can be accomplished with minimal changes to your application, depending on the Managed Cache Service features used by your caching application. While the APIs are not exactly the same they are similar, and much of your existing code that uses Managed Cache Service to access a cache can be reused with minimal changes. This article shows how to make the necessary configuration and application changes to migrate your Managed Cache Service applications to use Azure Cache for Redis, and shows how some of the features of Azure Cache for Redis can be used to implement the functionality of a Managed Cache Service cache.

NOTE

Managed Cache Service and In-Role Cache were [retired](#) November 30, 2016. If you have any In-Role Cache deployments that you want to migrate to Azure Cache for Redis, you can follow the steps in this article.

Migration Steps

The following steps are required to migrate a Managed Cache Service application to use Azure Cache for Redis.

- Map Managed Cache Service features to Azure Cache for Redis
- Choose a Cache Offering
- Create a Cache
- Configure the Cache Clients
 - Remove the Managed Cache Service Configuration
 - Configure a cache client using the StackExchange.Redis NuGet Package
- Migrate Managed Cache Service code
 - Connect to the cache using the ConnectionMultiplexer class
 - Access primitive data types in the cache
 - Work with .NET objects in the cache
- Migrate ASP.NET Session State and Output caching to Azure Cache for Redis

Map Managed Cache Service features to Azure Cache for Redis

Azure Managed Cache Service and Azure Cache for Redis are similar but implement some of their features in different ways. This section describes some of the differences and provides guidance on implementing the features of Managed Cache Service in Azure Cache for Redis.

MANAGED CACHE SERVICE FEATURE	MANAGED CACHE SERVICE SUPPORT	AZURE CACHE FOR REDIS SUPPORT
Named caches	A default cache is configured, and in the Standard and Premium cache offerings, up to nine additional named caches can be configured if desired.	Azure Cache for Redis has a configurable number of databases (default of 16) that can be used to implement a similar functionality to named caches. For more information, see What are Redis databases? and Default Redis server configuration .

MANAGED CACHE SERVICE FEATURE	MANAGED CACHE SERVICE SUPPORT	AZURE CACHE FOR REDIS SUPPORT
High Availability	Provides high availability for items in the cache in the Standard and Premium cache offerings. If items are lost due to a failure, backup copies of the items in the cache are still available. Writes to the secondary cache are made synchronously.	High availability is available in the Standard and Premium cache offerings, which have a two node Primary/Replica configuration (each shard in a Premium cache has a primary/replica pair). Writes to the replica are made asynchronously. For more information, see Azure Cache for Redis pricing .
Notifications	Allows clients to receive asynchronous notifications when a variety of cache operations occur on a named cache.	Client applications can use Redis pub/sub or Keyspace notifications to achieve a similar functionality to notifications.
Local cache	Stores a copy of cached objects locally on the client for extra-fast access.	Client applications would need to implement this functionality using a dictionary or similar data structure.
Eviction Policy	None or LRU. The default policy is LRU.	Azure Cache for Redis supports the following eviction policies: volatile-lru, allkeys-lru, volatile-random, allkeys-random, volatile-ttl, noeviction. The default policy is volatile-lru. For more information, see Default Redis server configuration .
Expiration Policy	The default expiration policy is Absolute and the default expiration interval is 10 minutes. Sliding and Never policies are also available.	By default items in the cache do not expire, but an expiration can be configured on a per write basis using cache set overloads.
Regions and Tagging	Regions are subgroups for cached items. Regions also support the annotation of cached items with additional descriptive strings called tags. Regions support the ability to perform search operations on any tagged items in that region. All items within a region are located within a single node of the cache cluster.	An Azure Cache for Redis consists of a single node (unless Redis cluster is enabled) so the concept of Managed Cache Service regions does not apply. Redis supports searching and wildcard operations when retrieving keys so descriptive tags can be embedded within the key names and used to retrieve the items later. For an example of implementing a tagging solution using Redis, see Implementing cache tagging with Redis .
Serialization	Managed Cache supports NetDataContractSerializer, BinaryFormatter, and the use of custom serializers. The default is NetDataContractSerializer.	It is the responsibility of the client application to serialize .NET objects before placing them into the cache, with the choice of the serializer up to the client application developer. For more information and sample code, see Work with .NET objects in the cache .
Cache emulator	Managed Cache provides a local cache emulator.	Azure Cache for Redis does not have an emulator, but you can run the MSOpenTech build of redis-server.exe locally to provide an emulator experience.

Choose a Cache Offering

Microsoft Azure Cache for Redis is available in the following tiers:

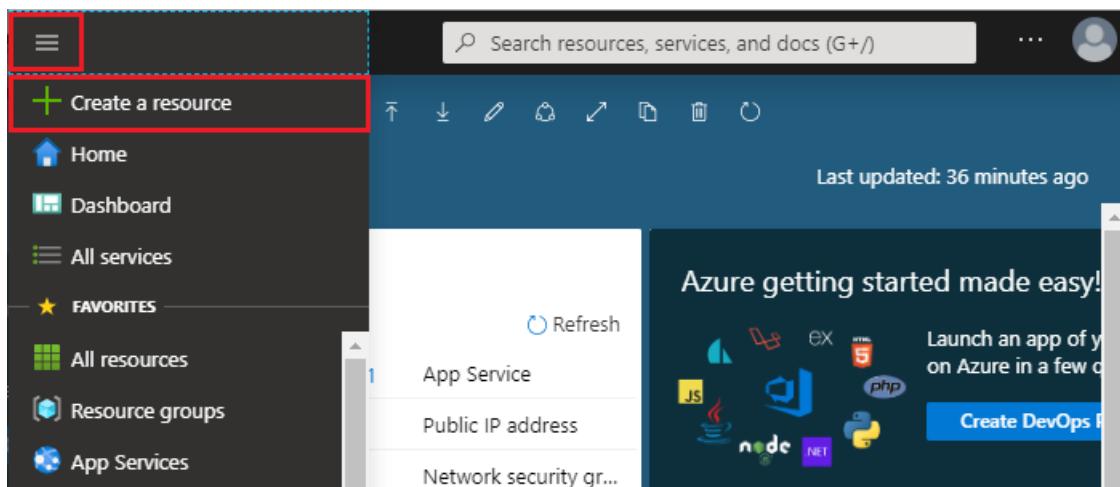
- **Basic** – Single node. Multiple sizes up to 53 GB.
- **Standard** – Two-node Primary/Replica. Multiple sizes up to 53 GB. 99.9% SLA.
- **Premium** – Two-node Primary/Replica with up to 10 shards. Multiple sizes from 6 GB to 1.2 TB. All Standard tier features and more including support for [Redis cluster](#), [Redis persistence](#), and [Azure Virtual Network](#). 99.9% SLA.

Each tier differs in terms of features and pricing. The features are covered later in this guide, and for more information on pricing, see [Cache Pricing Details](#).

A starting point for migration is to pick the size that matches the size of your previous Managed Cache Service cache, and then scale up or down depending on the requirements of your application. For more information on choosing the right Azure Cache for Redis offering, see [What Azure Cache for Redis offering and size should I use](#).

Create a Cache

1. To create a cache, sign in to the [Azure portal](#) and select **Create a resource**.



2. On the **New** page, select **Databases** and then select **Azure Cache for Redis**.

New

Search the Marketplace

Azure Marketplace [See all](#)

Get started
Recently created
AI + Machine Learning
Analytics
Blockchain
Compute
Containers
Databases
Developer Tools
DevOps
Identity
Integration
Internet of Things
Media
Mixed Reality
IT & Management Tools
Networking
Software as a Service (SaaS)
Security
Storage
Web

Featured [See all](#)

	Azure SQL Managed Instance Quickstart tutorial
	SQL Database Quickstart tutorial
	SQL Data Warehouse Quickstart tutorial
	Azure Database for MariaDB Learn more
	Couchbase Enterprise Edition (Hourly Pricing) (preview) Learn more
	Azure Database for MySQL Quickstart tutorial
	Azure Database for PostgreSQL Quickstart tutorial
	Azure Cosmos DB Quickstart tutorial
	SQL Server 2017 Enterprise Windows Server 2016 Learn more
	Azure Cache for Redis Quickstart tutorial

- On the **New Redis Cache** page, configure the settings for your new cache.

SETTING	SUGGESTED VALUE	DESCRIPTION
DNS name	Enter a globally unique name.	The cache name must be a string between 1 and 63 characters that contains only numbers, letters, or hyphens. The name must start and end with a number or letter, and can't contain consecutive hyphens. Your cache instance's <i>host name</i> will be < <i>DNS name</i> >.redis.cache.windows.net.
Subscription	Drop down and select your subscription.	The subscription under which to create this new Azure Cache for Redis instance.

SETTING	SUGGESTED VALUE	DESCRIPTION
Resource group	Drop down and select a resource group, or select Create new and enter a new resource group name.	Name for the resource group in which to create your cache and other resources. By putting all your app resources in one resource group, you can easily manage or delete them together.
Location	Drop down and select a location.	Select a region near other services that will use your cache.
Pricing tier	Drop down and select a Pricing tier .	The pricing tier determines the size, performance, and features that are available for the cache. For more information, see Azure Cache for Redis Overview .

4. Select **Create**.

New Redis Cache □ X

DNS name *
 ✓
.redis.cache.windows.net

Subscription *

Resource group * ⓘ
 ▼
[Create new](#)

Location *
 ▼

Pricing tier ([View full pricing details](#)) *
 ▼

(PREVIEW) Availability zone
 ▼

Redis Cluster ⓘ >

Data persistence ⓘ >

Virtual Network ⓘ >

Unblock port 6379 (not SSL encrypted)

Create [Automation options](#)

It takes a while for the cache to create. You can monitor progress on the Azure Cache for Redis [Overview](#) page. When **Status** shows as **Running**, the cache is ready to use.

The screenshot shows the Azure Cache for Redis blade for a resource named 'contoso-tmp'. The left sidebar lists navigation options: Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Settings, and Access keys. The main content area displays resource details. A red box highlights the 'Status' section, which shows 'Running - Basic 250 MB'. Other visible details include the host name 'contoso-tmp.redis.cache.windows.net', ports (Non-SSL port (6379) disabled), location 'Central US', subscription information ('A Subscription'), and a subscription ID '12345678-1234-1234-1234-123456781234'.

Configure the Cache Clients

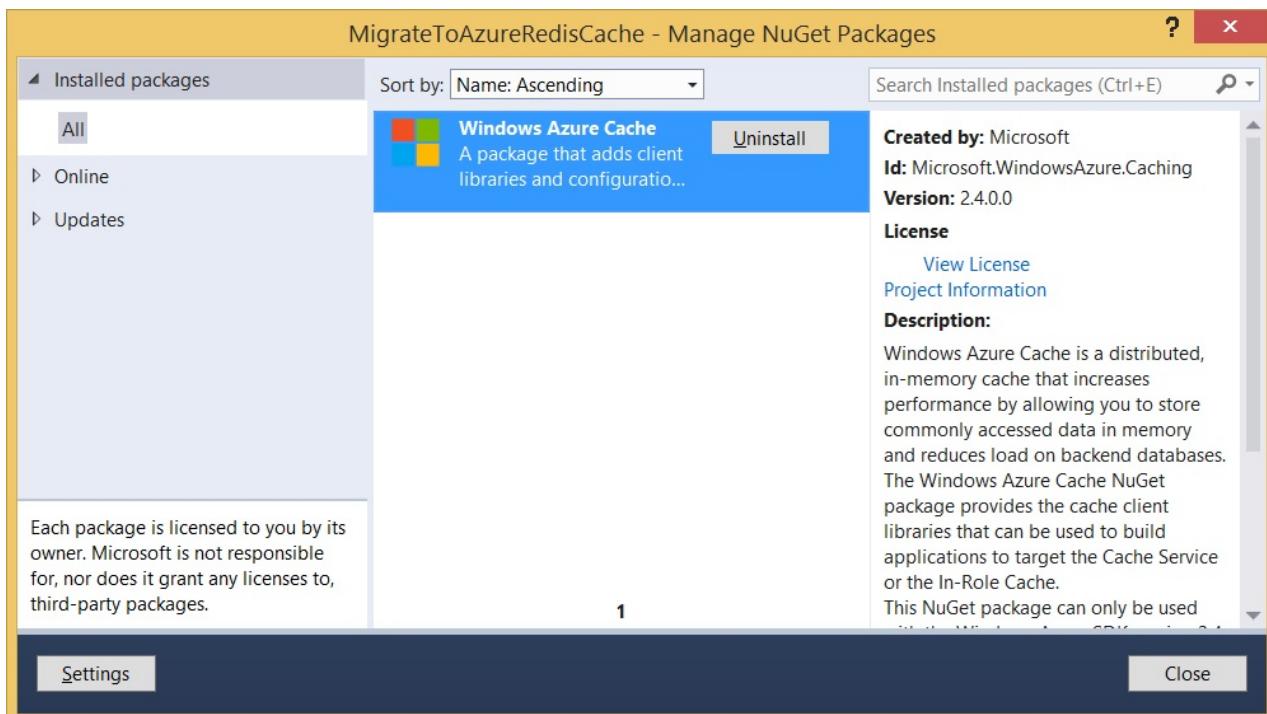
Once the cache is created and configured, the next step is to remove the Managed Cache Service configuration, and add the Azure Cache for Redis configuration and references so that cache clients can access the cache.

- Remove the Managed Cache Service Configuration
- Configure a cache client using the StackExchange.Redis NuGet Package

Remove the Managed Cache Service Configuration

Before the client applications can be configured for Azure Cache for Redis, the existing Managed Cache Service configuration and assembly references must be removed by uninstalling the Managed Cache Service NuGet package.

To uninstall the Managed Cache Service NuGet package, right-click the client project in **Solution Explorer** and choose **Manage NuGet Packages**. Select the **Installed packages** node, and type **WindowsAzure.Caching** into the Search installed packages box. Select **Windows Azure Cache** (or **Windows Azure Caching** depending on the version of the NuGet package), click **Uninstall**, and then click **Close**.



Uninstalling the Managed Cache Service NuGet package removes the Managed Cache Service assemblies and the Managed Cache Service entries in the app.config or web.config of the client application. Because some customized settings may not be removed when uninstalling the NuGet package, open web.config or app.config and ensure that the following elements are removed.

Ensure that the `dataCacheClients` entry is removed from the `configSections` element. Do not remove the entire `configSections` element; just remove the `dataCacheClients` entry, if it is present.

```
<configSections>
    <!-- Existing sections omitted for clarity. -->
    <section name="dataCacheClients" type="Microsoft.ApplicationServer.Caching.DataCacheClientsSection,
    Microsoft.ApplicationServer.Caching.Core" allowLocation="true" allowDefinition="Everywhere"/>
</configSections>
```

Ensure that the `dataCacheClients` section is removed. The `dataCacheClients` section will be similar to the following example.

```
<dataCacheClients>
    <dataCacheClientname="default">
        <!--To use the in-role flavor of Azure Cache, set identifier to be the cache cluster role name -->
        <!--To use the Azure Managed Cache Service, set identifier to be the endpoint of the cache cluster -->
        <autoDiscoverisEnabled="true" identifier="[Cache role name or Service Endpoint]"/>

        <!--<localCache isEnabled="true" sync="TimeoutBased" objectCount="100000" ttlValue="300" />-->
        <!--Use this section to specify security settings for connecting to your cache. This section is not
        required if your cache is hosted on a role that is a part of your cloud service. -->
        <!--<securityProperties mode="Message" sslEnabled="true">
            <messageSecurity authorizationInfo="[Authentication Key]" />
        </securityProperties>-->
    </dataCacheClient>
</dataCacheClients>
```

Once the Managed Cache Service configuration is removed, you can configure the cache client as described in the following section.

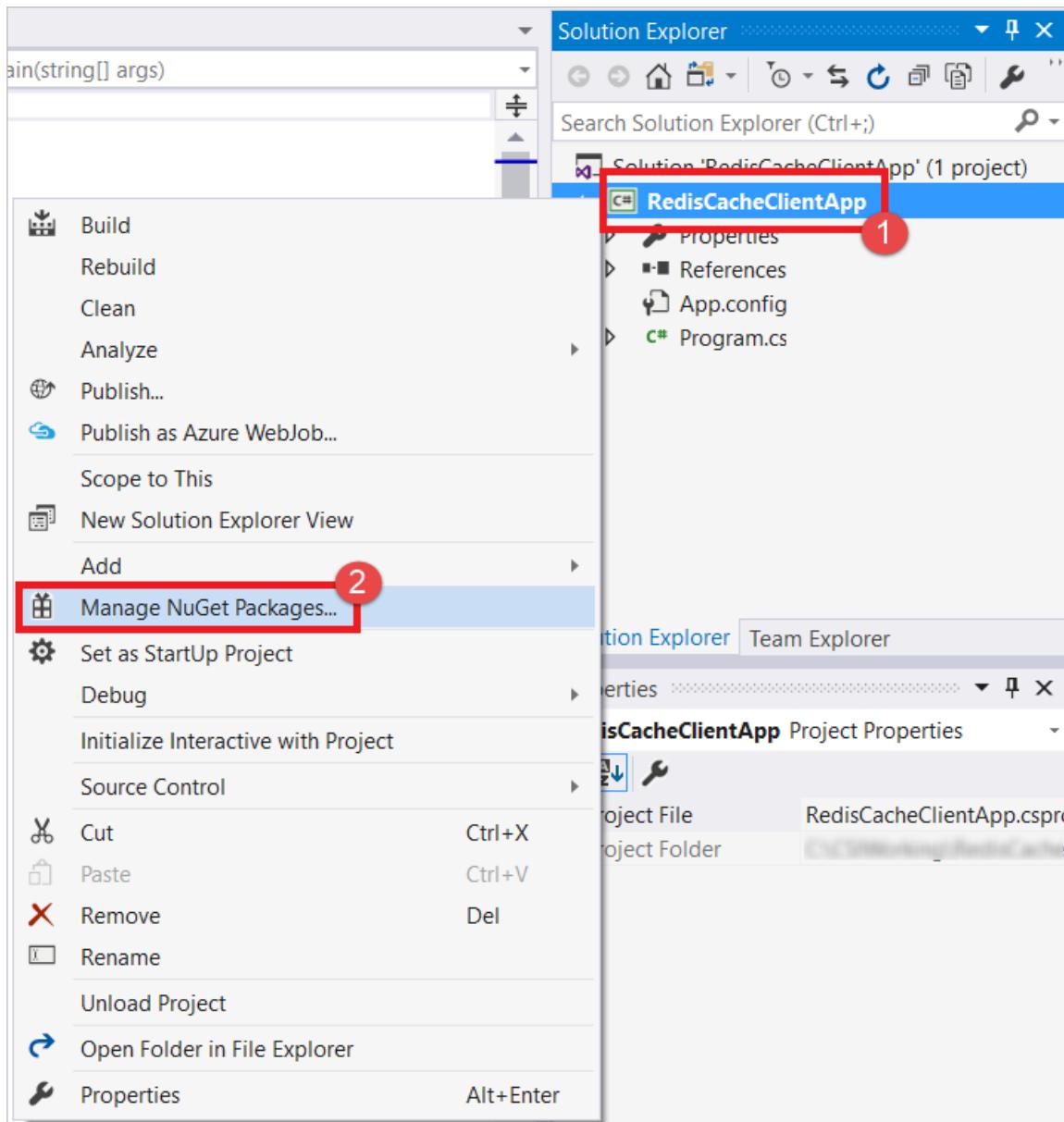
Configure a cache client using the StackExchange.Redis NuGet Package

.NET applications can use the **StackExchange.Redis** cache client, which can be configured in Visual Studio using a NuGet package that simplifies the configuration of cache client applications.

NOTE

For more information, see the [StackExchange.Redis GitHub page](#) and the [StackExchange.Azure Cache for Redis client documentation](#).

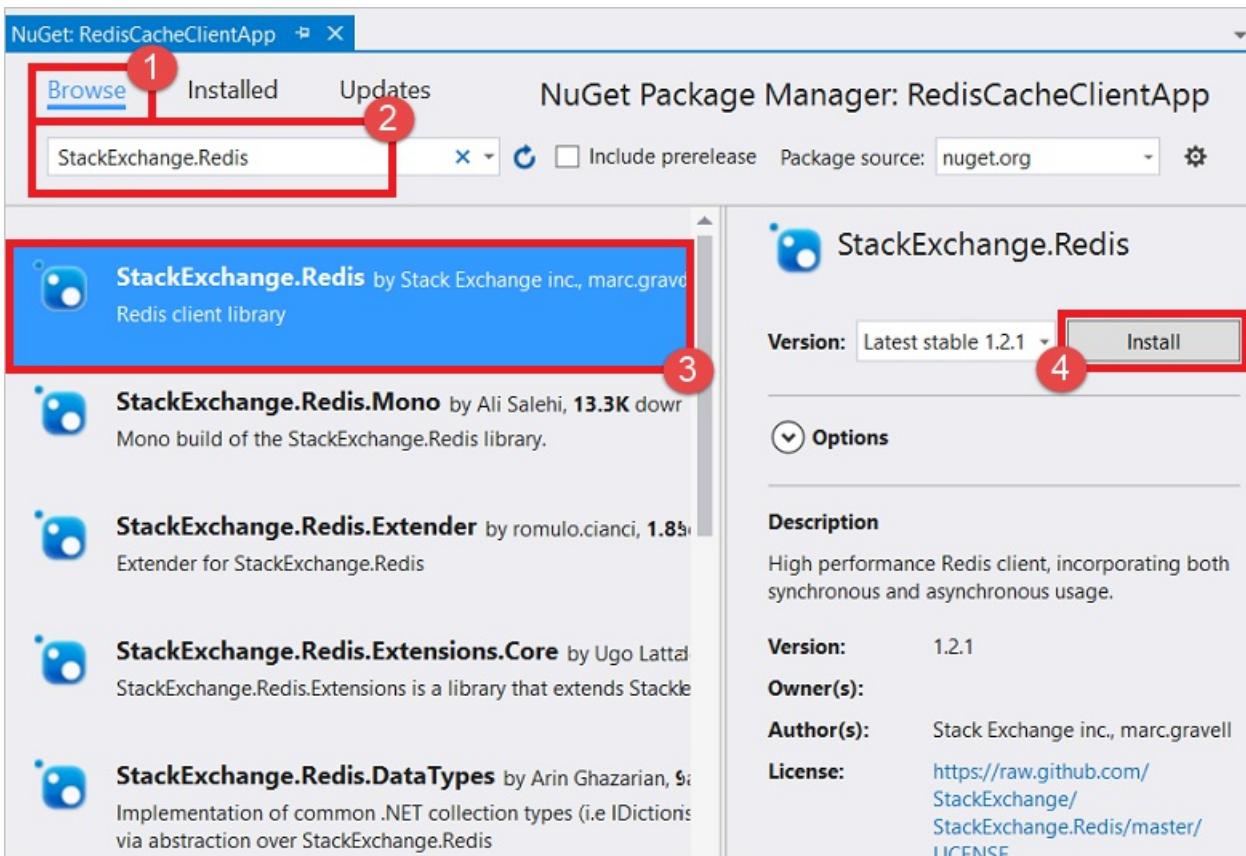
To configure a client application in Visual Studio using the StackExchange.Redis NuGet package, right-click the project in **Solution Explorer** and choose **Manage NuGet Packages**.



Type **StackExchange.Redis** or **StackExchange.Redis.StrongName** into the search text box, select the desired version from the results, and click **Install**.

NOTE

If you prefer to use a strong-named version of the **StackExchange.Redis** client library, choose **StackExchange.Redis.StrongName**; otherwise choose **StackExchange.Redis**.



The NuGet package downloads and adds the required assembly references for your client application to access Azure Cache for Redis with the StackExchange.Azure Cache for Redis client.

NOTE

If you have previously configured your project to use StackExchange.Redis, you can check for updates to the package from the **NuGet Package Manager**. To check for and install updated versions of the StackExchange.Redis NuGet package, click **Updates** in the **NuGet Package Manager** window. If an update to the StackExchange.Redis NuGet package is available, you can update your project to use the updated version.

You can also install the StackExchange.Redis NuGet package by clicking **NuGet Package Manager, Package Manager Console** from the **Tools** menu, and running the following command from the **Package Manager Console** window.

```
Install-Package StackExchange.Redis
```

Migrate Managed Cache Service code

The API for the StackExchange.Azure Cache for Redis client is similar to the Managed Cache Service. This section provides an overview of the differences.

Connect to the cache using the `ConnectionMultiplexer` class

In Managed Cache Service, connections to the cache were handled by the `DataCacheFactory` and `DataCache` classes. In Azure Cache for Redis, these connections are managed by the `ConnectionMultiplexer` class.

Add the following using statement to the top of any file from which you want to access the cache.

```
using StackExchange.Redis
```

If this namespace doesn't resolve, be sure that you have added the StackExchange.Redis NuGet package as described in [Quickstart: Use Azure Cache for Redis with a .NET application](#).

NOTE

Note that the StackExchange.Redis client requires .NET Framework 4 or higher.

To connect to an Azure Cache for Redis instance, call the static `ConnectionMultiplexer.Connect` method and pass in the endpoint and key. One approach to sharing a `ConnectionMultiplexer` instance in your application is to have a static property that returns a connected instance, similar to the following example. This approach provides a thread-safe way to initialize a single connected `ConnectionMultiplexer` instance. In this example `abortConnect` is set to false, which means that the call will succeed even if a connection to the cache is not established. One key feature of `ConnectionMultiplexer` is that it will automatically restore connectivity to the cache once the network issue or other causes are resolved.

```
private static Lazy<ConnectionMultiplexer> lazyConnection = new Lazy<ConnectionMultiplexer>(() =>
{
    return
        ConnectionMultiplexer.Connect("contoso5.redis.cache.windows.net,abortConnect=false,ssl=true,password=..."); });

public static ConnectionMultiplexer Connection
{
    get
    {
        return lazyConnection.Value;
    }
}
```

The cache endpoint, keys, and ports can be obtained from the **Azure Cache for Redis** blade for your cache instance. For more information, see [Azure Cache for Redis properties](#).

Once the connection is established, return a reference to the Azure Cache for Redis database by calling the `ConnectionMultiplexer.GetDatabase` method. The object returned from the `GetDatabase` method is a lightweight pass-through object and does not need to be stored.

```
IDatabase cache = Connection.GetDatabase();

// Perform cache operations using the cache object...
// Simple put of integral data types into the cache
cache.StringSet("key1", "value");
cache.StringSet("key2", 25);

// Simple get of data types from the cache
string key1 = cache.StringGet("key1");
int key2 = (int)cache.StringGet("key2");
```

The StackExchange.Redis client uses the `RedisKey` and `RedisValue` types for accessing and storing items in the cache. These types map onto most primitive language types, including string, and often are not used directly. Redis Strings are the most basic kind of Redis value, and can contain many types of data, including serialized binary streams, and while you may not use the type directly, you will use methods that contain `String` in the name. For most primitive data types, you store and retrieve items from the cache using the `StringSet` and `StringGet` methods, unless you are storing collections or other Redis data types in the cache.

`StringSet` and `StringGet` are similar to the Managed Cache Service `Put` and `Get` methods, with one major difference being that before you set and get a .NET object into the cache you must serialize it first.

When calling `StringGet`, if the object exists, it is returned, and if it does not, null is returned. In this case, you can retrieve the value from the desired data source and store it in the cache for subsequent use. This pattern is known as the cache-aside pattern.

To specify the expiration of an item in the cache, use the `TimeSpan` parameter of `StringSet`.

```
cache.StringSet("key1", "value1", TimeSpan.FromMinutes(90));
```

Azure Cache for Redis can work with .NET objects as well as primitive data types, but before a .NET object can be cached it must be serialized. This serialization is the responsibility of the application developer, and gives the developer flexibility in the choice of the serializer. For more information and sample code, see [Work with .NET objects in the cache](#).

Migrate ASP.NET Session State and Output caching to Azure Cache for Redis

Azure Cache for Redis has providers for both ASP.NET Session State and Page Output caching. To migrate your application that uses the Managed Cache Service versions of these providers, first remove the existing sections from your web.config, and then configure the Azure Cache for Redis versions of the providers. For instructions on using the Azure Cache for Redis ASP.NET providers, see [ASP.NET Session State Provider for Azure Cache for Redis](#) and [ASP.NET Output Cache Provider for Azure Cache for Redis](#).

Next steps

Explore the [Azure Cache for Redis documentation](#) for tutorials, samples, videos, and more.

Manage Azure Cache for Redis with Azure PowerShell

1/14/2020 • 21 minutes to read • [Edit Online](#)

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

This topic shows you how to perform common tasks such as create, update, and scale your Azure Cache for Redis instances, how to regenerate access keys, and how to view information about your caches. For a complete list of Azure Cache for Redis PowerShell cmdlets, see [Azure Cache for Redis cmdlets](#).

NOTE

Azure has two different deployment models you can use to create and work with resources: [Azure Resource Manager and classic](#). This article covers the use of the Resource Manager deployment model. We recommend the Resource Manager deployment model for new deployments instead of the classic deployment model.

For more information about the classic deployment model, see [Azure Resource Manager vs. classic deployment: Understand deployment models and the state of your resources](#).

Prerequisites

If you have already installed Azure PowerShell, you must have Azure PowerShell version 1.0.0 or later. You can check the version of Azure PowerShell that you have installed with this command at the Azure PowerShell command prompt.

```
Get-Module Az | format-table version
```

First, you must log in to Azure with this command.

```
Connect-AzAccount
```

Specify the email address of your Azure account and its password in the Microsoft Azure sign-in dialog.

Next, if you have multiple Azure subscriptions, you need to set your Azure subscription. To see a list of your current subscriptions, run this command.

```
Get-AzSubscription | sort SubscriptionName | Select SubscriptionName
```

To specify the subscription, run the following command. In the following example, the subscription name is `ContosoSubscription`.

```
Select-AzSubscription -SubscriptionName ContosoSubscription
```

Before you can use Windows PowerShell with Azure Resource Manager, you need the following:

- Windows PowerShell, Version 3.0 or 4.0. To find the version of Windows PowerShell, type: `$PSVersionTable` and verify the value of `PSVersion` is 3.0 or 4.0. To install a compatible version, see [Windows Management Framework 3.0](#) or [Windows Management Framework 4.0](#).

To get detailed help for any cmdlet you see in this tutorial, use the `Get-Help` cmdlet.

```
Get-Help <cmdlet-name> -Detailed
```

For example, to get help for the `New-AzRedisCache` cmdlet, type:

```
Get-Help New-AzRedisCache -Detailed
```

How to connect to other clouds

By default the Azure environment is `AzureCloud`, which represents the global Azure cloud instance. To connect to a different instance, use the `Connect-AzAccount` command with the `-Environment` or `-EnvironmentName` command line switch with the desired environment or environment name.

To see the list of available environments, run the `Get-AzEnvironment` cmdlet.

To connect to the Azure Government Cloud

To connect to the Azure Government Cloud, use one of the following commands.

```
Connect-AzAccount -EnvironmentName AzureUSGovernment
```

or

```
Connect-AzAccount -Environment (Get-AzEnvironment -Name AzureUSGovernment)
```

To create a cache in the Azure Government Cloud, use one of the following locations.

- USGov Virginia
- USGov Iowa

For more information about the Azure Government Cloud, see [Microsoft Azure Government](#) and [Microsoft Azure Government Developer Guide](#).

To connect to the Azure China Cloud

To connect to the Azure China Cloud, use one of the following commands.

```
Connect-AzAccount -EnvironmentName AzureChinaCloud
```

or

```
Connect-AzAccount -Environment (Get-AzEnvironment -Name AzureChinaCloud)
```

To create a cache in the Azure China Cloud, use one of the following locations.

- China East
- China North

For more information about the Azure China Cloud, see [AzureChinaCloud for Azure operated by 21Vianet in China](#).

To connect to Microsoft Azure Germany

To connect to Microsoft Azure Germany, use one of the following commands.

```
Connect-AzAccount -EnvironmentName AzureGermanCloud
```

or

```
Connect-AzAccount -Environment (Get-AzEnvironment -Name AzureGermanCloud)
```

To create a cache in Microsoft Azure Germany, use one of the following locations.

- Germany Central
- Germany Northeast

For more information about Microsoft Azure Germany, see [Microsoft Azure Germany](#).

Properties used for Azure Cache for Redis PowerShell

The following table contains properties and descriptions for commonly used parameters when creating and managing your Azure Cache for Redis instances using Azure PowerShell.

PARAMETER	DESCRIPTION	DEFAULT
Name	Name of the cache	
Location	Location of the cache	
ResourceGroupName	Resource group name in which to create the cache	
Size	The size of the cache. Valid values are: P1, P2, P3, P4, C0, C1, C2, C3, C4, C5, C6, 250MB, 1GB, 2.5GB, 6GB, 13GB, 26GB, 53GB	1GB
ShardCount	The number of shards to create when creating a premium cache with clustering enabled. Valid values are: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10	
SKU	Specifies the SKU of the cache. Valid values are: Basic, Standard, Premium	Standard
RedisConfiguration	Specifies Redis configuration settings. For details on each setting, see the following RedisConfiguration properties table.	
EnableNonSslPort	Indicates whether the non-SSL port is enabled.	False

PARAMETER	DESCRIPTION	DEFAULT
MaxMemoryPolicy	This parameter has been deprecated - use RedisConfiguration instead.	
StaticIP	When hosting your cache in a VNET, specifies a unique IP address in the subnet for the cache. If not provided, one is chosen for you from the subnet.	
Subnet	When hosting your cache in a VNET, specifies the name of the subnet in which to deploy the cache.	
VirtualNetwork	When hosting your cache in a VNET, specifies the resource ID of the VNET in which to deploy the cache.	
KeyType	Specifies which access key to regenerate when renewing access keys. Valid values are: Primary, Secondary	

RedisConfiguration properties

PROPERTY	DESCRIPTION	PRICING TIERS
rdb-backup-enabled	Whether Redis data persistence is enabled	Premium only
rdb-storage-connection-string	The connection string to the storage account for Redis data persistence	Premium only
rdb-backup-frequency	The backup frequency for Redis data persistence	Premium only
maxmemory-reserved	Configures the memory reserved for non-cache processes	Standard and Premium
maxmemory-policy	Configures the eviction policy for the cache	All pricing tiers
notify-keyspace-events	Configures keyspace notifications	Standard and Premium
hash-max-ziplist-entries	Configures memory optimization for small aggregate data types	Standard and Premium
hash-max-ziplist-value	Configures memory optimization for small aggregate data types	Standard and Premium
set-max-intset-entries	Configures memory optimization for small aggregate data types	Standard and Premium
zset-max-ziplist-entries	Configures memory optimization for small aggregate data types	Standard and Premium

PROPERTY	DESCRIPTION	PRICING TIERS
zset-max-ziplist-value	Configures memory optimization for small aggregate data types	Standard and Premium
databases	Configures the number of databases. This property can be configured only at cache creation.	Standard and Premium

To create an Azure Cache for Redis

New Azure Cache for Redis instances are created using the [New-AzRedisCache](#) cmdlet.

IMPORTANT

The first time you create an Azure Cache for Redis in a subscription using the Azure portal, the portal registers the `Microsoft.Cache` namespace for that subscription. If you attempt to create the first Azure Cache for Redis in a subscription using PowerShell, you must first register that namespace using the following command; otherwise cmdlets such as `New-AzRedisCache` and `Get-AzRedisCache` fail.

```
Register-AzResourceProvider -ProviderNamespace "Microsoft.Cache"
```

To see a list of available parameters and their descriptions for `New-AzRedisCache`, run the following command.

```
PS C:\> Get-Help New-AzRedisCache -detailed

NAME
    New-AzRedisCache

SYNOPSIS
    Creates a new Azure Cache for Redis.

SYNTAX
    New-AzRedisCache -Name <String> -ResourceGroupName <String> -Location <String> [-RedisVersion <String>]
        [-Size <String>] [-Sku <String>] [-MaxMemoryPolicy <String>] [-RedisConfiguration <Hashtable>] [-
        EnableNonSslPort <Boolean>] [-ShardCount <Integer>] [-VirtualNetwork <String>] [-Subnet <String>] [-StaticIP <String>]
        [<CommonParameters>]

DESCRIPTION
    The New-AzRedisCache cmdlet creates a new Azure Cache for Redis.

PARAMETERS
    -Name <String>
        Name of the Azure Cache for Redis to create.

    -ResourceGroupName <String>
        Name of resource group in which to create the Azure Cache for Redis.

    -Location <String>
        Location in which to create the Azure Cache for Redis.

    -RedisVersion <String>
        RedisVersion is deprecated and will be removed in future release.

    -Size <String>
        Size of the Azure Cache for Redis. The default value is 1GB or C1. Possible values are P1, P2, P3, P4,
        C0, C1, C2, C3,
        C4, C5, C6, 250MB, 1GB, 2.5GB, 6GB, 13GB, 26GB, 53GB.
```

```

-Sku <String>
    Sku of Azure Cache for Redis. The default value is Standard. Possible values are Basic, Standard and Premium.

-MaxMemoryPolicy <String>
    The 'MaxMemoryPolicy' setting has been deprecated. Please use 'RedisConfiguration' setting to set MaxMemoryPolicy. e.g. -RedisConfiguration @{"maxmemory-policy" = "allkeys-lru"}

-RedisConfiguration <Hashtable>
    All Redis Configuration Settings. Few possible keys: rdb-backup-enabled, rdb-storage-connection-string,
    rdb-backup-frequency, maxmemory-reserved, maxmemory-policy, notify-keyspace-events, hash-max-ziplist-entries,
    hash-max-ziplist-value, set-max-intset-entries, zset-max-ziplist-entries, zset-max-ziplist-value,
    databases.

-EnableNonSslPort <Boolean>
    EnableNonSslPort is used by Azure Cache for Redis. If no value is provided, the default value is false and the non-SSL port will be disabled. Possible values are true and false.

-ShardCount <Integer>
    The number of shards to create on a Premium Cluster Cache.

-VirtualNetwork <String>
    The exact ARM resource ID of the virtual network to deploy the Azure Cache for Redis in. Example format: /subscriptions/{subid}/resourceGroups/{resourceGroupName}/providers/Microsoft.ClassicNetwork/VirtualNetworks/{vnetName}

-Subnet <String>
    Required when deploying an Azure Cache for Redis inside an existing Azure Virtual Network.

-StaticIP <String>
    Required when deploying an Azure Cache for Redis inside an existing Azure Virtual Network.

<CommonParameters>
    This cmdlet supports the common parameters: Verbose, Debug,
    ErrorAction, ErrorVariable, WarningAction, WarningVariable,
    OutBuffer, PipelineVariable, and OutVariable. For more information, see about_CommonParameters (https://go.microsoft.com/fwlink/?LinkID=113216).

```

To create a cache with default parameters, run the following command.

```
New-AzRedisCache -ResourceGroupName myGroup -Name mycache -Location "North Central US"
```

`ResourceGroupName`, `Name`, and `Location` are required parameters, but the rest are optional and have default values. Running the previous command creates a Standard SKU Azure Cache for Redis instance with the specified name, location, and resource group, that is 1 GB in size with the non-SSL port disabled.

To create a premium cache, specify a size of P1 (6 GB - 60 GB), P2 (13 GB - 130 GB), P3 (26 GB - 260 GB), or P4 (53 GB - 530 GB). To enable clustering, specify a shard count using the `ShardCount` parameter. The following example creates a P1 premium cache with 3 shards. A P1 premium cache is 6 GB in size, and since we specified three shards the total size is 18 GB (3 x 6 GB).

```
New-AzRedisCache -ResourceGroupName myGroup -Name mycache -Location "North Central US" -Sku Premium -Size P1 -
    ShardCount 3
```

To specify values for the `RedisConfiguration` parameter, enclose the values inside `{}` as key/value pairs like `@{"maxmemory-policy" = "allkeys-random", "notify-keyspace-events" = "KEA"}`. The following example creates a

standard 1 GB cache with `allkeys-random` maxmemory policy and keyspace notifications configured with `KEA`. For more information, see [Keyspace notifications \(advanced settings\)](#) and [Memory policies](#).

```
New-AzRedisCache -ResourceGroupName myGroup -Name mycache -Location "North Central US" -RedisConfiguration @{"maxmemory-policy" = "allkeys-random", "notify-keyspace-events" = "KEA"}
```

To configure the databases setting during cache creation

The `databases` setting can be configured only during cache creation. The following example creates a premium P3 (26 GB) cache with 48 databases using the [New-AzRedisCache](#) cmdlet.

```
New-AzRedisCache -ResourceGroupName myGroup -Name mycache -Location "North Central US" -Sku Premium -Size P3 -RedisConfiguration @{"databases" = "48"}
```

For more information on the `databases` property, see [Default Azure Cache for Redis server configuration](#). For more information on creating a cache using the [New-AzRedisCache](#) cmdlet, see the previous To create an Azure Cache for Redis section.

To update an Azure Cache for Redis

Azure Cache for Redis instances are updated using the [Set-AzRedisCache](#) cmdlet.

To see a list of available parameters and their descriptions for `Set-AzRedisCache`, run the following command.

```

PS C:\> Get-Help Set-AzRedisCache -detailed

NAME
Set-AzRedisCache

SYNOPSIS
Set Azure Cache for Redis updatable parameters.

SYNTAX
Set-AzRedisCache -Name <String> -ResourceGroupName <String> [-Size <String>] [-Sku <String>]
[-MaxMemoryPolicy <String>] [-RedisConfiguration <Hashtable>] [-EnableNonSslPort <Boolean>] [-ShardCount
<Integer>] [<CommonParameters>]

DESCRIPTION
The Set-AzRedisCache cmdlet sets Azure Cache for Redis parameters.

PARAMETERS
-Name <String>
    Name of the Azure Cache for Redis to update.

-ResourceGroupName <String>
    Name of the resource group for the cache.

-Size <String>
    Size of the Azure Cache for Redis. The default value is 1GB or C1. Possible values are P1, P2, P3, P4,
C0, C1, C2, C3,
C4, C5, C6, 250MB, 1GB, 2.5GB, 6GB, 13GB, 26GB, 53GB.

-Sku <String>
    Sku of Azure Cache for Redis. The default value is Standard. Possible values are Basic, Standard and
Premium.

-MaxMemoryPolicy <String>
    The 'MaxMemoryPolicy' setting has been deprecated. Please use 'RedisConfiguration' setting to set
    MaxMemoryPolicy. e.g. -RedisConfiguration @{"maxmemory-policy" = "allkeys-lru"}

-RedisConfiguration <Hashtable>
    All Redis Configuration Settings. Few possible keys: rdb-backup-enabled, rdb-storage-connection-
    string,
    rdb-backup-frequency, maxmemory-reserved, maxmemory-policy, notify-keyspace-events, hash-max-ziplist-
    entries,
    hash-max-ziplist-value, set-max-intset-entries, zset-max-ziplist-entries, zset-max-ziplist-value.

-EnableNonSslPort <Boolean>
    EnableNonSslPort is used by Azure Cache for Redis. The default value is null and no change will be
    made to the
    currently configured value. Possible values are true and false.

-ShardCount <Integer>
    The number of shards to create on a Premium Cluster Cache.

<CommonParameters>
    This cmdlet supports the common parameters: Verbose, Debug,
    ErrorAction, ErrorVariable, WarningAction, WarningVariable,
    OutBuffer, PipelineVariable, and OutVariable. For more information, see
    about_CommonParameters (https://go.microsoft.com/fwlink/?LinkID=113216).

```

The `Set-AzRedisCache` cmdlet can be used to update properties such as `Size`, `Sku`, `EnableNonSslPort`, and the `RedisConfiguration` values.

The following command updates the maxmemory-policy for the Azure Cache for Redis named myCache.

```
Set-AzRedisCache -ResourceGroupName "myGroup" -Name "myCache" -RedisConfiguration @{"maxmemory-policy" =
"allkeys-random"}
```

To scale an Azure Cache for Redis

`Set-AzRedisCache` can be used to scale an Azure Cache for Redis instance when the `Size`, `Sku`, or `ShardCount` properties are modified.

NOTE

Scaling a cache using PowerShell is subject to the same limits and guidelines as scaling a cache from the Azure portal. You can scale to a different pricing tier with the following restrictions.

- You can't scale from a higher pricing tier to a lower pricing tier.
- You can't scale from a **Premium** cache down to a **Standard** or a **Basic** cache.
- You can't scale from a **Standard** cache down to a **Basic** cache.
- You can scale from a **Basic** cache to a **Standard** cache but you can't change the size at the same time. If you need a different size, you can do a subsequent scaling operation to the desired size.
- You can't scale from a **Basic** cache directly to a **Premium** cache. You must scale from **Basic** to **Standard** in one scaling operation, and then from **Standard** to **Premium** in a subsequent scaling operation.
- You can't scale from a larger size down to the **CO (250 MB)** size.

For more information, see [How to Scale Azure Cache for Redis](#).

The following example shows how to scale a cache named `myCache` to a 2.5 GB cache. Note that this command works for both a Basic or a Standard cache.

```
Set-AzRedisCache -ResourceGroupName myGroup -Name myCache -Size 2.5GB
```

After this command is issued, the status of the cache is returned (similar to calling `Get-AzRedisCache`). Note that the `ProvisioningState` is `Scaling`.

```
PS C:\> Set-AzRedisCache -Name myCache -ResourceGroupName myGroup -Size 2.5GB
```

```
Name          : mycache
Id           : /subscriptions/12ad12bd-abdc-2231-a2ed-a2b8b246bbad4/resourceGroups/mygroup/providers/Microsoft.Cache/Redis/mycache
Location     : South Central US
Type         : Microsoft.Cache/Redis
HostName     : mycache.redis.cache.windows.net
Port         : 6379
ProvisioningState : Scaling
SslPort      : 6380
RedisConfiguration : {[maxmemory-policy, volatile-lru], [maxmemory-reserved, 150], [notify-keyspace-events, KEA],
                     [maxmemory-delta, 150]...}
EnableNonSslPort : False
RedisVersion  : 3.0
Size         : 1GB
Sku          : Standard
ResourceGroupName : mygroup
PrimaryKey    : ....
SecondaryKey   :
VirtualNetwork  :
Subnet        :
StaticIP      :
TenantSettings : {}
ShardCount    :
```

When the scaling operation is complete, the `ProvisioningState` changes to `succeeded`. If you need to make a subsequent scaling operation, such as changing from Basic to Standard and then changing the size, you must wait

until the previous operation is complete or you receive an error similar to the following.

```
Set-AzRedisCache : Conflict: The resource '...' is not in a stable state, and is currently unable to accept the update request.
```

To get information about an Azure Cache for Redis

You can retrieve information about a cache using the [Get-AzRedisCache](#) cmdlet.

To see a list of available parameters and their descriptions for [Get-AzRedisCache](#), run the following command.

```
PS C:\> Get-Help Get-AzRedisCache -detailed

NAME
    Get-AzRedisCache

SYNOPSIS
    Gets details about a single cache or all caches in the specified resource group or all caches in the current subscription.

SYNTAX
    Get-AzRedisCache [-Name <String>] [-ResourceGroupName <String>] [<CommonParameters>]

DESCRIPTION
    The Get-AzRedisCache cmdlet gets the details about a cache or caches depending on input parameters. If both ResourceGroupName and Name parameters are provided then Get-AzRedisCache will return details about the specific cache name provided.

    If only ResourceGroupName is provided than it will return details about all caches in the specified resource group.

    If no parameters are given than it will return details about all caches the current subscription.

PARAMETERS
    -Name <String>
        The name of the cache. When this parameter is provided along with ResourceGroupName, Get-AzRedisCache returns the details for the cache.

    -ResourceGroupName <String>
        The name of the resource group that contains the cache or caches. If ResourceGroupName is provided with Name then Get-AzRedisCache returns the details of the cache specified by Name. If only the ResourceGroup parameter is provided, then details for all caches in the resource group are returned.

    <CommonParameters>
        This cmdlet supports the common parameters: Verbose, Debug, ErrorAction, ErrorVariable, WarningAction, WarningVariable, OutBuffer, PipelineVariable, and OutVariable. For more information, see about_CommonParameters (https://go.microsoft.com/fwlink/?LinkID=113216).
```

To return information about all caches in the current subscription, run [Get-AzRedisCache](#) without any parameters.

```
Get-AzRedisCache
```

To return information about all caches in a specific resource group, run [Get-AzRedisCache](#) with the [ResourceGroupName](#) parameter.

```
Get-AzRedisCache -ResourceGroupName myGroup
```

To return information about a specific cache, run `Get-AzRedisCache` with the `Name` parameter containing the name of the cache, and the `ResourceGroupName` parameter with the resource group containing that cache.

```
PS C:\> Get-AzRedisCache -Name myCache -ResourceGroupName myGroup

Name          : mycache
Id            : /subscriptions/12ad12bd-abdc-2231-a2ed-a2b8b246bbad4/resourceGroups/myGroup/providers/Microsoft.Cache/Redis/mycache
Location      : South Central US
Type          : Microsoft.Cache/Redis
HostName      : mycache.redis.cache.windows.net
Port          : 6379
ProvisioningState : Succeeded
SslPort        : 6380
RedisConfiguration : {[maxmemory-policy, volatile-lru], [maxmemory-reserved, 62], [notify-keyspace-events, KEA],
                      [maxclients, 1000]...}
EnableNonSslPort : False
RedisVersion   : 3.0
Size          : 1GB
Sku           : Standard
ResourceGroupName : myGroup
VirtualNetwork  :
Subnet         :
StaticIP       :
TenantSettings : {}
ShardCount     :
```

To retrieve the access keys for an Azure Cache for Redis

To retrieve the access keys for your cache, you can use the `Get-AzRedisCacheKey` cmdlet.

To see a list of available parameters and their descriptions for `Get-AzRedisCacheKey`, run the following command.

```
PS C:\> Get-Help Get-AzRedisCacheKey -detailed

NAME
    Get-AzRedisCacheKey

SYNOPSIS
    Gets the access keys for the specified Azure Cache for Redis.

SYNTAX
    Get-AzRedisCacheKey -Name <String> -ResourceGroupName <String> [<CommonParameters>]

DESCRIPTION
    The Get-AzRedisCacheKey cmdlet gets the access keys for the specified cache.

PARAMETERS
    -Name <String>
        Name of the Azure Cache for Redis.

    -ResourceGroupName <String>
        Name of the resource group for the cache.

    <CommonParameters>
        This cmdlet supports the common parameters: Verbose, Debug,
        ErrorAction, ErrorVariable, WarningAction, WarningVariable,
        OutBuffer, PipelineVariable, and OutVariable. For more information, see
        about_CommonParameters (https://go.microsoft.com/fwlink/?LinkID=113216).
```

To retrieve the keys for your cache, call the `Get-AzRedisCacheKey` cmdlet and pass in the name of your cache the name of the resource group that contains the cache.

```
PS C:\> Get-AzRedisCacheKey -Name myCache -ResourceGroupName myGroup

PrimaryKey   : b2wdt43sfet1ju4hfbrfnregrd9wgIcc6IA3zA01lY=
SecondaryKey : ABhfB757JgjIgt785JgKH9865eifmekfnn649303JKL=
```

To regenerate access keys for your Azure Cache for Redis

To regenerate the access keys for your cache, you can use the [New-AzRedisCacheKey](#) cmdlet.

To see a list of available parameters and their descriptions for `New-AzRedisCacheKey`, run the following command.

```
PS C:\> Get-Help New-AzRedisCacheKey -detailed
```

NAME
New-AzRedisCacheKey

SYNOPSIS
Regenerates the access key of an Azure Cache for Redis.

SYNTAX
New-AzRedisCacheKey -Name <String> -ResourceGroupName <String> -KeyType <String> [-Force] [<CommonParameters>]

DESCRIPTION
The New-AzRedisCacheKey cmdlet regenerate the access key of an Azure Cache for Redis.

PARAMETERS

- Name <String>
Name of the Azure Cache for Redis.
- ResourceGroupName <String>
Name of the resource group for the cache.
- KeyType <String>
Specifies whether to regenerate the primary or secondary access key. Possible values are Primary or Secondary.
- Force
When the Force parameter is provided, the specified access key is regenerated without any confirmation prompts.
- <CommonParameters>
This cmdlet supports the common parameters: Verbose, Debug, ErrorAction, ErrorVariable, WarningAction, WarningVariable, OutBuffer, PipelineVariable, and OutVariable. For more information, see about_CommonParameters (<https://go.microsoft.com/fwlink/?LinkID=113216>).

To regenerate the primary or secondary key for your cache, call the `New-AzRedisCacheKey` cmdlet and pass in the name, resource group, and specify either `Primary` or `Secondary` for the `KeyType` parameter. In the following example, the secondary access key for a cache is regenerated.

```
PS C:\> New-AzRedisCacheKey -Name myCache -ResourceGroupName myGroup -KeyType Secondary
```

Confirm
Are you sure you want to regenerate Secondary key for Azure Cache for Redis 'myCache'?
[Y] Yes [N] No [S] Suspend [?] Help (default is "Y"): Y

```
PrimaryKey : b2wdt43sfet1ju4hfbrfynregrd9wgIcc6IA3zA01lY=
SecondaryKey : c53hj3kh4jhHjPJk810jji785JgKH9865eifmekfnn6=
```

To delete an Azure Cache for Redis

To delete an Azure Cache for Redis, use the `Remove-AzRedisCache` cmdlet.

To see a list of available parameters and their descriptions for `Remove-AzRedisCache`, run the following command.

```
PS C:\> Get-Help Remove-AzRedisCache -detailed
```

NAME
Remove-AzRedisCache

SYNOPSIS
Remove Azure Cache for Redis if exists.

SYNTAX
Remove-AzRedisCache -Name <String> -ResourceGroupName <String> [-Force] [-PassThru] [<CommonParameters>]

DESCRIPTION
The Remove-AzRedisCache cmdlet removes an Azure Cache for Redis if it exists.

PARAMETERS

- Name <String>
Name of the Azure Cache for Redis to remove.
- ResourceGroupName <String>
Name of the resource group of the cache to remove.
- Force
When the Force parameter is provided, the cache is removed without any confirmation prompts.
- PassThru
By default Remove-AzRedisCache removes the cache and does not return any value. If the PassThru parameter is provided then Remove-AzRedisCache returns a boolean value indicating the success of the operation.
- <CommonParameters>
This cmdlet supports the common parameters: Verbose, Debug, ErrorAction, ErrorVariable, WarningAction, WarningVariable, OutBuffer, PipelineVariable, and OutVariable. For more information, see about_CommonParameters (<https://go.microsoft.com/fwlink/?LinkID=113216>).

In the following example, the cache named `myCache` is removed.

```
PS C:\> Remove-AzRedisCache -Name myCache -ResourceGroupName myGroup
```

Confirm
Are you sure you want to remove Azure Cache for Redis 'myCache'?
[Y] Yes [N] No [S] Suspend [?] Help (default is "Y"):

To import an Azure Cache for Redis

You can import data into an Azure Cache for Redis instance using the `Import-AzRedisCache` cmdlet.

IMPORTANT

Import/Export is only available for [premium tier](#) caches. For more information about Import/Export, see [Import and Export data in Azure Cache for Redis](#).

To see a list of available parameters and their descriptions for `Import-AzRedisCache`, run the following command.

```
PS C:\> Get-Help Import-AzRedisCache -detailed
```

NAME
`Import-AzRedisCache`

SYNOPSIS
Import data from blobs to Azure Cache for Redis.

SYNTAX
`Import-AzRedisCache -Name <String> -ResourceGroupName <String> -Files <String[]> [-Format <String>] [-Force] [-PassThru] [<CommonParameters>]`

DESCRIPTION
The `Import-AzRedisCache` cmdlet imports data from the specified blobs into Azure Cache for Redis.

PARAMETERS

- Name <String>
The name of the cache.
- ResourceGroupName <String>
The name of the resource group that contains the cache.
- Files <String[]>
SAS urls of blobs whose content should be imported into the cache.
- Format <String>
Format for the blob. Currently "rdb" is the only supported, with other formats expected in the future.
- Force
When the Force parameter is provided, import will be performed without any confirmation prompts.
- PassThru
By default `Import-AzRedisCache` imports data in cache and does not return any value. If the PassThru parameter is provided then `Import-AzRedisCache` returns a boolean value indicating the success of the operation.
- <CommonParameters>
This cmdlet supports the common parameters: Verbose, Debug, ErrorAction, ErrorVariable, WarningAction, WarningVariable, OutBuffer, PipelineVariable, and OutVariable. For more information, see [about_CommonParameters](https://go.microsoft.com/fwlink/?LinkID=113216) (<https://go.microsoft.com/fwlink/?LinkID=113216>).

The following command imports data from the blob specified by the SAS uri into Azure Cache for Redis.

```
PS C:\>Import-AzRedisCache -ResourceGroupName "resourceGroupName" -Name "cacheName" -Files @("https://mystorageaccount.blob.core.windows.net/mycontainername/blobname?sv=2015-04-05&sr=b&sig=caIwutG2uDa0NZ8mjdnJdgOY8%2F8mhwRuGNdICU%2B0pI4%3D&st=2016-05-27T00%3A00%3A00Z&se=2016-05-28T00%3A00%3A00Z&sp=rwd") -Force
```

To export an Azure Cache for Redis

You can export data from an Azure Cache for Redis instance using the `Export-AzRedisCache` cmdlet.

IMPORTANT

Import/Export is only available for [premium tier](#) caches. For more information about Import/Export, see [Import and Export data in Azure Cache for Redis](#).

To see a list of available parameters and their descriptions for `Export-AzRedisCache`, run the following command.

```
PS C:\> Get-Help Export-AzRedisCache -detailed
```

NAME
`Export-AzRedisCache`

SYNOPSIS
Exports data from Azure Cache for Redis to a specified container.

SYNTAX
`Export-AzRedisCache -Name <String> -ResourceGroupName <String> -Prefix <String> -Container <String> [-Format <String>] [-PassThru] [<CommonParameters>]`

DESCRIPTION
The `Export-AzRedisCache` cmdlet exports data from Azure Cache for Redis to a specified container.

PARAMETERS

- `-Name <String>`
The name of the cache.
- `-ResourceGroupName <String>`
The name of the resource group that contains the cache.
- `-Prefix <String>`
Prefix to use for blob names.
- `-Container <String>`
SAS url of container where data should be exported.
- `-Format <String>`
Format for the blob. Currently "rdb" is the only supported, with other formats expected in the future.
- `-PassThru`
By default `Export-AzRedisCache` does not return any value. If the `PassThru` parameter is provided then `Export-AzRedisCache` returns a boolean value indicating the success of the operation.
- `<CommonParameters>`
This cmdlet supports the common parameters: `Verbose`, `Debug`, `ErrorAction`, `ErrorVariable`, `WarningAction`, `WarningVariable`, `OutBuffer`, `PipelineVariable`, and `OutVariable`. For more information, see [about_CommonParameters](#) (<https://go.microsoft.com/fwlink/?LinkID=113216>).

The following command exports data from an Azure Cache for Redis instance into the container specified by the SAS uri.

```
PS C:\>Export-AzRedisCache -ResourceGroupName "resourceGroupName" -Name "cacheName" -Prefix "blobprefix" -Container "https://mystorageaccount.blob.core.windows.net/mycontainer?sv=2015-04-05&sr=c&sig=HezZtBZ3DURmEGDduauE7pvETY4kqlPI8JCNa8ATmaw%3D&st=2016-05-27T00%3A00%3A00Z&se=2016-05-28T00%3A00%3A00Z&sp=rwdl"
```

To reboot an Azure Cache for Redis

You can reboot your Azure Cache for Redis instance using the `Reset-AzRedisCache` cmdlet.

IMPORTANT

Reboot is only available for [premium tier](#) caches. For more information about rebooting your cache, see [Cache administration - reboot](#).

To see a list of available parameters and their descriptions for `Reset-AzRedisCache`, run the following command.

```
PS C:\> Get-Help Reset-AzRedisCache -detailed

NAME
    Reset-AzRedisCache

SYNOPSIS
    Reboot specified node(s) of an Azure Cache for Redis instance.

SYNTAX
    Reset-AzRedisCache -Name <String> -ResourceGroupName <String> -RebootType <String> [-ShardId <Integer>]
    [-Force] [-PassThru] [<CommonParameters>]

DESCRIPTION
    The Reset-AzRedisCache cmdlet reboots the specified node(s) of an Azure Cache for Redis instance.

PARAMETERS
    -Name <String>
        The name of the cache.

    -ResourceGroupName <String>
        The name of the resource group that contains the cache.

    -RebootType <String>
        Which node to reboot. Possible values are "PrimaryNode", "SecondaryNode", "AllNodes".

    -ShardId <Integer>
        Which shard to reboot when rebooting a premium cache with clustering enabled.

    -Force
        When the Force parameter is provided, reset will be performed without any confirmation prompts.

    -PassThru
        By default Reset-AzRedisCache does not return any value. If the PassThru parameter is provided
        then Reset-AzRedisCache returns a boolean value indicating the success of the operation.

    <CommonParameters>
        This cmdlet supports the common parameters: Verbose, Debug,
        ErrorAction, ErrorVariable, WarningAction, WarningVariable,
        OutBuffer, PipelineVariable, and OutVariable. For more information, see
        about_CommonParameters (https://go.microsoft.com/fwlink/?LinkID=113216).
```

The following command reboots both nodes of the specified cache.

```
PS C:\>Reset-AzRedisCache -ResourceGroupName "resourceGroupName" -Name "cacheName" -RebootType "AllNodes"
-Force
```

Next steps

To learn more about using Windows PowerShell with Azure, see the following resources:

- [Azure Cache for Redis cmdlet documentation on MSDN](#)
- [Azure Resource Manager Cmdlets](#): Learn to use the cmdlets in the Azure Resource Manager module.
- [Using Resource groups to manage your Azure resources](#): Learn how to create and manage resource groups in the Azure portal.
- [Azure blog](#): Learn about new features in Azure.
- [Windows PowerShell blog](#): Learn about new features in Windows PowerShell.
- ["Hey, Scripting Guy!" Blog](#): Get real-world tips and tricks from the Windows PowerShell community.

Azure CLI Samples for Azure Cache for Redis

12/13/2019 • 2 minutes to read • [Edit Online](#)

The following table includes links to bash scripts built using the Azure CLI.

Create cache	
Create a cache	Creates a resource group and a basic tier Azure Cache for Redis.
Create a premium cache with clustering	Creates a resource group and a premium tier cache with clustering enabled.
Get cache details	Gets details of an Azure Cache for Redis instance, including provisioning status.
Get the hostname, ports, and keys	Gets the hostname, ports, and keys for an Azure Cache for Redis instance.
Web app plus cache	
Connect a web app to an Azure Cache for Redis	Creates an Azure web app and an Azure Cache for Redis, then adds the redis connection details to the app settings.
Delete cache	
Delete a cache	Deletes an Azure Cache for Redis instance

For more information about the Azure CLI, see [Install the Azure CLI](#) and [Get started with Azure CLI](#).

Create an Azure Cache for Redis using a template

12/23/2019 • 4 minutes to read • [Edit Online](#)

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

In this topic, you learn how to create an Azure Resource Manager template that deploys an Azure Cache for Redis. The cache can be used with an existing storage account to keep diagnostic data. You also learn how to define which resources are deployed and how to define parameters that are specified when the deployment is executed. You can use this template for your own deployments, or customize it to meet your requirements.

Currently, diagnostic settings are shared for all caches in the same region for a subscription. Updating one cache in the region affects all other caches in the region.

For more information about creating templates, see [Authoring Azure Resource Manager Templates](#). To learn about the JSON syntax and properties for cache resource types, see [Microsoft.Cache resource types](#).

For the complete template, see [Azure Cache for Redis template](#).

NOTE

Resource Manager templates for the new [Premium tier](#) are available.

- [Create a Premium Azure Cache for Redis with clustering](#)
- [Create Premium Azure Cache for Redis with data persistence](#)
- [Create Premium Redis Cache deployed into a Virtual Network](#)

To check for the latest templates, see [Azure Quickstart Templates](#) and search for `Azure Cache for Redis`.

What you will deploy

In this template, you will deploy an Azure Cache for Redis that uses an existing storage account for diagnostic data.

To run the deployment automatically, click the following button:



Parameters

With Azure Resource Manager, you define parameters for values you want to specify when the template is deployed. The template includes a section called Parameters that contains all of the parameter values. You should define a parameter for those values that vary based on the project you are deploying or based on the environment you are deploying to. Do not define parameters for values that always stay the same. Each parameter value is used in the template to define the resources that are deployed.

cacheSKUName

The pricing tier of the new Azure Cache for Redis.

```

"cacheSKUName": {
    "type": "string",
    "allowedValues": [
        "Basic",
        "Standard",
        "Premium"
    ],
    "defaultValue": "Basic",
    "metadata": {
        "description": "The pricing tier of the new Azure Cache for Redis."
    }
},

```

The template defines the values that are permitted for this parameter (Basic, Standard, or Premium), and assigns a default value (Basic) if no value is specified. Basic provides a single node with multiple sizes available up to 53 GB. Standard provides two-node Primary/Replica with multiple sizes available up to 53 GB and 99.9% SLA.

cacheSKUFamily

The family for the sku.

```

"cacheSKUFamily": {
    "type": "string",
    "allowedValue/s": [
        "C",
        "P"
    ],
    "defaultValue": "C",
    "metadata": {
        "description": "The family for the sku."
    }
},

```

cacheSKUCapacity

The size of the new Azure Cache for Redis instance.

For the Basic and Standard families:

```

"cacheSKUCapacity": {
    "type": "int",
    "allowedValues": [
        0,
        1,
        2,
        3,
        4,
        5,
        6
    ],
    "defaultValue": 0,
    "metadata": {
        "description": "The size of the new Azure Cache for Redis instance. "
    }
}

```

The Premium value cache capacity is defined the same, except the allowed values run from 1 to 5 instead of from 0 to 6.

The template defines the integer values that are permitted for this parameter (0 through 6 for the Basic and Standard families; 1 through 5 for the Premium family). If no value is specified, the template assigns a default value of 0 for Basic and Standard, 1 for Premium.

The values correspond to following cache sizes:

Value	Basic and Standard Cache Size	Premium Cache Size
0	250 MB (default)	n/a
1	1 GB	6 GB (default)
2	2.5 GB	13 GB
3	6 GB	26 GB
4	13 GB	53 GB
5	26 GB	120 GB
6	53 GB	n/a

redisCacheLocation

The location of the Azure Cache for Redis. For best performance, use the same location as the app to be used with the cache.

```
"redisCacheLocation": {  
    "type": "string"  
}
```

existingDiagnosticsStorageAccountName

The name of the existing storage account to use for diagnostics.

```
"existingDiagnosticsStorageAccountName": {  
    "type": "string"  
}
```

enableNonSslPort

A boolean value that indicates whether to allow access via non-SSL ports.

```
"enableNonSslPort": {  
    "type": "bool"  
}
```

diagnosticsStatus

A value that indicates whether diagnostics is enabled. Use ON or OFF.

```
"diagnosticsStatus": {  
    "type": "string",  
    "defaultValue": "ON",  
    "allowedValues": [  
        "ON",  
        "OFF"  
    ]  
}
```

Resources to deploy

Azure Cache for Redis

Creates the Azure Cache for Redis.

```
{  
    "apiVersion": "2015-08-01",  
    "name": "[parameters('redisCacheName')]",  
    "type": "Microsoft.Cache/Redis",  
    "location": "[parameters('redisCacheLocation')]",  
    "properties": {  
        "enableNonSslPort": "[parameters('enableNonSslPort')]",  
        "sku": {  
            "capacity": "[parameters('redisCacheCapacity')]",  
            "family": "[parameters('redisCacheFamily')]",  
            "name": "[parameters('redisCacheSKU')]"  
        }  
    },  
    "resources": [  
        {  
            "apiVersion": "2017-05-01-preview",  
            "type": "Microsoft.Cache/redis/providers/diagnosticsettings",  
            "name": "[concat(parameters('redisCacheName'), '/Microsoft.Insights/service')]",  
            "location": "[parameters('redisCacheLocation')]",  
            "dependsOn": [  
                "[concat('Microsoft.Cache/Redis/', parameters('redisCacheName'))]"  
            ],  
            "properties": {  
                "status": "[parameters('diagnosticsStatus')]",  
                "storageAccountName": "[parameters('existingDiagnosticsStorageAccountName')]"  
            }  
        }  
    ]  
}
```

Commands to run deployment

To deploy the resources to Azure, you must be signed in to your Azure account and you must use the Azure Resource Manager module. To learn about using Azure Resource Manager with either Azure PowerShell or Azure CLI, see:

- [Manage Azure resources by using Azure PowerShell](#)
- [Manage Azure resources by using Azure CLI](#).

The following examples assume you already have a resource group in your account with the specified name.

PowerShell

```
New-AzResourceGroupDeployment -TemplateUri https://raw.githubusercontent.com/Azure/azure-quickstart-templates/master/101-redis-cache/azuredeploy.json -ResourceGroupName ExampleDeployGroup -redisCacheName ExampleCache
```

Azure CLI

```
azur group deployment create --template-uri https://raw.githubusercontent.com/Azure/azure-quickstart-templates/master/101-redis-cache/azuredeploy.json -g ExampleDeployGroup
```

Create a Web App plus Azure Cache for Redis using a template

12/23/2019 • 5 minutes to read • [Edit Online](#)

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

In this topic, you will learn how to create an Azure Resource Manager template that deploys an Azure Web App with Azure Cache for Redis. You will learn how to define which resources are deployed and how to define parameters that are specified when the deployment is executed. You can use this template for your own deployments, or customize it to meet your requirements.

For more information about creating templates, see [Authoring Azure Resource Manager Templates](#). To learn about the JSON syntax and properties for cache resource types, see [Microsoft.Cache resource types](#).

For the complete template, see [Web App with Azure Cache for Redis template](#).

What you will deploy

In this template, you will deploy:

- Azure Web App
- Azure Cache for Redis

To run the deployment automatically, click the following button:



Parameters to specify

With Azure Resource Manager, you define parameters for values you want to specify when the template is deployed. The template includes a section called Parameters that contains all of the parameter values. You should define a parameter for those values that will vary based on the project you are deploying or based on the environment you are deploying to. Do not define parameters for values that will always stay the same. Each parameter value is used in the template to define the resources that are deployed.

When defining parameters, use the **allowedValues** field to specify which values a user can provide during deployment. Use the **defaultValue** field to assign a value to the parameter, if no value is provided during deployment.

We will describe each parameter in the template.

siteName

The name of the web app that you wish to create.

```
"siteName":{  
    "type":"string"  
}
```

hostingPlanName

The name of the App Service plan to use for hosting the web app.

```
"hostingPlanName":{  
    "type":"string"  
}
```

sku

The pricing tier for the hosting plan.

```
"sku": {  
    "type": "string",  
    "allowedValues": [  
        "F1",  
        "D1",  
        "B1",  
        "B2",  
        "B3",  
        "S1",  
        "S2",  
        "S3",  
        "P1",  
        "P2",  
        "P3",  
        "P4"  
    ],  
    "defaultValue": "S1",  
    "metadata": {  
        "description": "The pricing tier for the hosting plan."  
    }  
}
```

The template defines the values that are permitted for this parameter, and assigns a default value (S1) if no value is specified.

workerSize

The instance size of the hosting plan (small, medium, or large).

```
"workerSize":{  
    "type":"string",  
    "allowedValues": [  
        "0",  
        "1",  
        "2"  
    ],  
    "defaultValue":"0"  
}
```

The template defines the values that are permitted for this parameter (0, 1, or 2), and assigns a default value (0) if no value is specified. The values correspond to small, medium and large.

cacheSKUName

The pricing tier of the new Azure Cache for Redis.

```

"cacheSKUName": {
    "type": "string",
    "allowedValues": [
        "Basic",
        "Standard",
        "Premium"
    ],
    "defaultValue": "Basic",
    "metadata": {
        "description": "The pricing tier of the new Azure Cache for Redis."
    }
},

```

The template defines the values that are permitted for this parameter (Basic, Standard, or Premium), and assigns a default value (Basic) if no value is specified. Basic provides a single node with multiple sizes available up to 53 GB. Standard provides two-node Primary/Replica with multiple sizes available up to 53 GB and 99.9% SLA.

cacheSKUFamily

The family for the sku.

```

"cacheSKUFamily": {
    "type": "string",
    "allowedValue/s": [
        "C",
        "P"
    ],
    "defaultValue": "C",
    "metadata": {
        "description": "The family for the sku."
    }
},

```

cacheSKUCapacity

The size of the new Azure Cache for Redis instance.

For the Basic and Standard families:

```

"cacheSKUCapacity": {
    "type": "int",
    "allowedValues": [
        0,
        1,
        2,
        3,
        4,
        5,
        6
    ],
    "defaultValue": 0,
    "metadata": {
        "description": "The size of the new Azure Cache for Redis instance. "
    }
}

```

The Premium value cache capacity is defined the same, except the allowed values run from 1 to 5 instead of from 0 to 6.

The template defines the integer values that are permitted for this parameter (0 through 6 for the Basic and Standard families; 1 through 5 for the Premium family). If no value is specified, the template assigns a default value of 0 for Basic and Standard, 1 for Premium.

The values correspond to following cache sizes:

VALUE	BASIC AND STANDARD CACHE SIZE	PREMIUM CACHE SIZE
0	250 MB (default)	n/a
1	1 GB	6 GB (default)
2	2.5 GB	13 GB
3	6 GB	26 GB
4	13 GB	53 GB
5	26 GB	120 GB
6	53 GB	n/a

Variables for names

This template uses variables to construct names for the resources. It uses the [uniqueString](#) function to construct a value based on the resource group id.

```
"variables": {  
    "hostingPlanName": "[concat('hostingplan', uniqueString(resourceGroup().id))]",  
    "webSiteName": "[concat('webSite', uniqueString(resourceGroup().id))]",  
    "cacheName": "[concat('cache', uniqueString(resourceGroup().id))]"  
},
```

Resources to deploy

App Service plan

Creates the service plan for hosting the web app. You provide the name of the plan through the **hostingPlanName** parameter. The location of the plan is the same location used for the resource group. The pricing tier and worker size are specified in the **sku** and **workerSize** parameters

```
{  
    "apiVersion": "2015-08-01",  
    "name": "[parameters('hostingPlanName')]",  
    "type": "Microsoft.Web/serverfarms",  
    "location": "[resourceGroup().location]",  
    "sku": {  
        "name": "[parameters('sku')]",  
        "capacity": "[parameters('workerSize')]"  
    },  
    "properties": {  
        "name": "[parameters('hostingPlanName')]"  
    }  
},
```

Azure Cache for Redis

Creates the Azure Cache for Redis that is used with the web app. The name of the cache is specified in the **cacheName** variable.

The template creates the cache in the same location as the resource group.

```
{  
  "name": "[variables('cacheName')]",  
  "type": "Microsoft.Cache/Redis",  
  "location": "[resourceGroup().location]",  
  "apiVersion": "2015-08-01",  
  "dependsOn": [ ],  
  "tags": {  
    "displayName": "cache"  
  },  
  "properties": {  
    "sku": {  
      "name": "[parameters('cacheSKUName')]",  
      "family": "[parameters('cacheSKUFamily')]",  
      "capacity": "[parameters('cacheSKUCapacity')]"  
    }  
  }  
}
```

Web app

Creates the web app with name specified in the **webSiteName** variable.

Notice that the web app is configured with app setting properties that enable it to work with the Azure Cache for Redis. These app settings are dynamically created based on values provided during deployment.

```
{  
  "apiVersion": "2015-08-01",  
  "name": "[variables('webSiteName')]",  
  "type": "Microsoft.Web/sites",  
  "location": "[resourceGroup().location]",  
  "dependsOn": [  
    "[concat('Microsoft.Web/serverFarms/', variables('hostingPlanName'))]",  
    "[concat('Microsoft.Cache/Redis/', variables('cacheName'))]"  
  ],  
  "tags": {  
    "[concat('hidden-related:', resourceGroup().id, '/providers/Microsoft.Web/serverfarms/',  
    variables('hostingPlanName'))]": "empty",  
    "displayName": "Website"  
  },  
  "properties": {  
    "name": "[variables('webSiteName')]",  
    "serverFarmId": "[resourceId('Microsoft.Web/serverfarms', variables('hostingPlanName'))]"  
  },  
  "resources": [  
    {  
      "apiVersion": "2015-08-01",  
      "type": "config",  
      "name": "appsettings",  
      "dependsOn": [  
        "[concat('Microsoft.Web/Sites/', variables('webSiteName'))]",  
        "[concat('Microsoft.Cache/Redis/', variables('cacheName'))]"  
      ],  
      "properties": {  
        "CacheConnection": "  
[concat(variables('cacheName'), '.redis.cache.windows.net,abortConnect=false,ssl=true,password=',  
listKeys(resourceId('Microsoft.Cache/Redis', variables('cacheName')), '2015-08-01').primaryKey)]"  
      }  
    }  
  ]  
}
```

Commands to run deployment

To deploy the resources to Azure, you must be signed in to your Azure account and you must use the Azure Resource Manager module. To learn about using Azure Resource Manager with either Azure PowerShell or Azure CLI, see:

- [Manage Azure resources by using Azure PowerShell](#)
- [Manage Azure resources by using Azure CLI.](#)

The following examples assume you already have a resource group in your account with the specified name.

PowerShell

```
New-AzResourceGroupDeployment -TemplateUri https://raw.githubusercontent.com/Azure/azure-quickstart-templates/master/201-web-app-with-redis-cache/azuredeploy.json -ResourceGroupName ExampleDeployGroup
```

Azure CLI

```
az group deployment create --template-uri https://raw.githubusercontent.com/Azure/azure-quickstart-templates/master/201-web-app-with-redis-cache/azuredeploy.json -g ExampleDeployGroup
```

ASP.NET Session State Provider for Azure Cache for Redis

12/13/2019 • 6 minutes to read • [Edit Online](#)

Azure Cache for Redis provides a session state provider that you can use to store your session state in-memory with Azure Cache for Redis instead of a SQL Server database. To use the caching session state provider, first configure your cache, and then configure your ASP.NET application for cache using the Azure Cache for Redis Session State NuGet package.

It's often not practical in a real-world cloud app to avoid storing some form of state for a user session, but some approaches impact performance and scalability more than others. If you have to store state, the best solution is to keep the amount of state small and store it in cookies. If that isn't feasible, the next best solution is to use ASP.NET session state with a provider for distributed, in-memory cache. The worst solution from a performance and scalability standpoint is to use a database backed session state provider. This topic provides guidance on using the ASP.NET Session State Provider for Azure Cache for Redis. For information on other session state options, see [ASP.NET Session State options](#).

Store ASP.NET session state in the cache

To configure a client application in Visual Studio using the Azure Cache for Redis Session State NuGet package, click **NuGet Package Manager**, **Package Manager Console** from the **Tools** menu.

Run the following command from the `Package Manager Console` window.

```
Install-Package Microsoft.Web.RedisSessionStateProvider
```

IMPORTANT

If you are using the clustering feature from the premium tier, you must use [RedisSessionStateProvider 2.0.1](#) or higher or an exception is thrown. Moving to 2.0.1 or higher is a breaking change; for more information, see [v2.0.0 Breaking Change Details](#). At the time of this article update, the current version of this package is 2.2.3.

The Redis Session State Provider NuGet package has a dependency on the StackExchange.Redis.StrongName package. If the StackExchange.Redis.StrongName package is not present in your project, it is installed.

NOTE

In addition to the strong-named StackExchange.Redis.StrongName package, there is also the StackExchange.Redis non-strong-named version. If your project is using the non-strong-named StackExchange.Redis version you must uninstall it, otherwise you get naming conflicts in your project. For more information about these packages, see [Configure .NET cache clients](#).

The NuGet package downloads and adds the required assembly references and adds the following section into your web.config file. This section contains the required configuration for your ASP.NET application to use the Azure Cache for Redis Session State Provider.

```

<sessionState mode="Custom" customProvider="MySessionStateStore">
  <providers>
    <!-- Either use 'connectionString' OR 'settingsClassName' and 'settingsMethodName' OR use
    'host','port','accessKey','ssl','connectionTimeoutInMilliseconds' and 'operationTimeoutInMilliseconds'. -->
    <!-- 'throwOnError','retryTimeoutInMilliseconds','databaseId' and 'applicationName' can be used with both
    options. -->
    <!--
      <add name="MySessionStateStore"
        host = "127.0.0.1" [String]
        port = "" [number]
        accessKey = "" [String]
        ssl = "false" [true|false]
        throwOnError = "true" [true|false]
        retryTimeoutInMilliseconds = "5000" [number]
        databaseId = "0" [number]
        applicationName = "" [String]
        connectionTimeoutInMilliseconds = "5000" [number]
        operationTimeoutInMilliseconds = "1000" [number]
        connectionString = "<Valid StackExchange.Redis connection string>" [String]
        settingsClassName = "<Assembly qualified class name that contains settings method specified below.
        Which basically return 'connectionString' value>" [String]
        settingsMethodName = "<Settings method should be defined in settingsClass. It should be public,
        static, does not take any parameters and should have a return type of 'String', which is basically
        'connectionString' value.>" [String]
        loggingClassName = "<Assembly qualified class name that contains logging method specified below>" [String]
        loggingMethodName = "<Logging method should be defined in loggingClass. It should be public, static,
        does not take any parameters and should have a return type of System.IO.TextWriter.>" [String]
        redisSerializerType = "<Assembly qualified class name that implements
        Microsoft.Web.Redis.ISerializer>" [String]
      />
    -->
    <add name="MySessionStateStore" type="Microsoft.Web.Redis.RedisSessionStateProvider"
      host=""
      accessKey=""
      ssl="true" />
  </providers>
</sessionState>

```

The commented section provides an example of the attributes and sample settings for each attribute.

Configure the attributes with the values from your cache blade in the Microsoft Azure portal, and configure the other values as desired. For instructions on accessing your cache properties, see [Configure Azure Cache for Redis settings](#).

- **host** – specify your cache endpoint.
- **port** – use either your non-SSL port or your SSL port, depending on the **ssl** settings.
- **accessKey** – use either the primary or secondary key for your cache.
- **ssl** – true if you want to secure cache/client communications with **ssl**; otherwise false. Be sure to specify the correct port.
 - The non-SSL port is disabled by default for new caches. Specify true for this setting to use the SSL port.
For more information about enabling the non-SSL port, see the [Access Ports](#) section in the [Configure a cache](#) topic.
- **throwOnError** – true if you want an exception to be thrown if there is a failure, or false if you want the operation to fail silently. You can check for a failure by checking the static `Microsoft.Web.Redis.RedisSessionStateProvider.LastException` property. The default is true.
- **retryTimeoutInMilliseconds** – Operations that fail are retried during this interval, specified in milliseconds. The first retry occurs after 20 milliseconds, and then retries occur every second until the `retryTimeoutInMilliseconds` interval expires. Immediately after this interval, the operation is retried one final time. If the operation still fails, the exception is thrown back to the caller, depending on the `throwOnError`

setting. The default value is 0, which means no retries.

- **databaseId** – Specifies which database to use for cache output data. If not specified, the default value of 0 is used.
- **applicationName** – Keys are stored in redis as `{<Application Name>_<Session ID>}_Data`. This naming scheme enables multiple applications to share the same Redis instance. This parameter is optional and if you do not provide it a default value is used.
- **connectionTimeoutInMilliseconds** – This setting allows you to override the connectTimeout setting in the StackExchange.Redis client. If not specified, the default connectTimeout setting of 5000 is used. For more information, see [StackExchange.Redis configuration model](#).
- **operationTimeoutInMilliseconds** – This setting allows you to override the syncTimeout setting in the StackExchange.Redis client. If not specified, the default syncTimeout setting of 1000 is used. For more information, see [StackExchange.Redis configuration model](#).
- **redisSerializerType** - This setting allows you to specify custom serialization of session content that is sent to Redis. The type specified must implement `Microsoft.Web.Redis.ISerializer` and must declare public parameterless constructor. By default `System.Runtime.Serialization.Formatters.Binary.BinayFormatter` is used.

For more information about these properties, see the original blog post announcement at [Announcing ASP.NET Session State Provider for Redis](#).

Don't forget to comment out the standard InProc session state provider section in your web.config.

```
<!-- <sessionState mode="InProc"
    customProvider="DefaultSessionProvider">
<providers>
    <add name="DefaultSessionProvider"
        type="System.Web.Providers.DefaultSessionStateProvider,
        System.Web.Providers, Version=1.0.0.0, Culture=neutral,
        PublicKeyToken=31bf3856ad364e35"
        connectionStringName="DefaultConnection" />
</providers>
</sessionState> -->
```

Once these steps are performed, your application is configured to use the Azure Cache for Redis Session State Provider. When you use session state in your application, it is stored in an Azure Cache for Redis instance.

IMPORTANT

Data stored in the cache must be serializable, unlike the data that can be stored in the default in-memory ASP.NET Session State Provider. When the Session State Provider for Redis is used, be sure that the data types that are being stored in session state are serializable.

ASP.NET Session State options

- In Memory Session State Provider - This provider stores the Session State in memory. The benefit of using this provider is it is simple and fast. However you cannot scale your Web Apps if you are using in memory provider since it is not distributed.
- Sql Server Session State Provider - This provider stores the Session State in Sql Server. Use this provider if you want to store the Session state in persistent storage. You can scale your Web App but using Sql Server for Session has a performance impact on your Web App. You can also use this provider with an [In-Memory OLTP configuration](#) to help improve performance.
- Distributed In Memory Session State Provider such as Azure Cache for Redis Session State Provider - This provider gives you the best of both worlds. Your Web App can have a simple, fast, and scalable Session State Provider. Because this provider stores the Session state in a Cache, your app has to take in consideration all the

characteristics associated when talking to a Distributed In Memory Cache, such as transient network failures. For best practices on using Cache, see [Caching guidance](#) from Microsoft Patterns & Practices [Azure Cloud Application Design and Implementation Guidance](#).

For more information about session state and other best practices, see [Web Development Best Practices \(Building Real-World Cloud Apps with Azure\)](#).

Next steps

Check out the [ASP.NET Output Cache Provider for Azure Cache for Redis](#).

ASP.NET Output Cache Provider for Azure Cache for Redis

12/13/2019 • 8 minutes to read • [Edit Online](#)

The Redis Output Cache Provider is an out-of-process storage mechanism for output cache data. This data is specifically for full HTTP responses (page output caching). The provider plugs into the new output cache provider extensibility point that was introduced in ASP.NET 4.

To use the Redis Output Cache Provider, first configure your cache, and then configure your ASP.NET application using the Redis Output Cache Provider NuGet package. This topic provides guidance on configuring your application to use the Redis Output Cache Provider. For more information about creating and configuring an Azure Cache for Redis instance, see [Create a cache](#).

Store ASP.NET page output in the cache

To configure a client application in Visual Studio using the Azure Cache for Redis Session State NuGet package, click **NuGet Package Manager**, **Package Manager Console** from the **Tools** menu.

Run the following command from the **Package Manager Console** window.

```
Install-Package Microsoft.Web.RedisOutputCacheProvider
```

The Redis Output Cache Provider NuGet package has a dependency on the StackExchange.Redis.StrongName package. If the StackExchange.Redis.StrongName package is not present in your project, it is installed. For more information about the Redis Output Cache Provider NuGet package, see the [RedisOutputCacheProvider](#) NuGet page.

NOTE

In addition to the strong-named StackExchange.Redis.StrongName package, there is also the StackExchange.Redis non-strong-named version. If your project is using the non-strong-named StackExchange.Redis version you must uninstall it; otherwise, you will experience naming conflicts in your project. For more information about these packages, see [Configure .NET cache clients](#).

The NuGet package downloads and adds the required assembly references and adds the following section into your web.config file. This section contains the required configuration for your ASP.NET application to use the Redis Output Cache Provider.

```
<caching>
<outputCache defaultProvider="MyRedisOutputCache">
<providers>
    <add name="MyRedisOutputCache" type="Microsoft.Web.Redis.RedisOutputCacheProvider"
        host=""
        accessKey=""
        ssl="true" />
</providers>
</outputCache>
</caching>
```

Configure the attributes with the values from your cache blade in the Microsoft Azure portal, and configure the

other values as desired. For instructions on accessing your cache properties, see [Configure Azure Cache for Redis settings](#).

ATTRIBUTE	TYPE	DEFAULT	DESCRIPTION
<i>host</i>	string	"localhost"	The Redis server IP address or host name
<i>port</i>	positive integer	6379 (non-SSL) 6380 (SSL)	Redis server port
<i>accessKey</i>	string	""	Redis server password when Redis authorization is enabled. The value is empty string by default, which means the session state provider won't use any password when connecting to Redis server. If your Redis server is in a publicly accessible network like Azure Redis Cache, be sure to enable Redis authorization to improve security, and provide a secure password.
<i>ssl</i>	boolean	false	<p>Whether to connect to Redis server via SSL. This value is false by default because Redis doesn't support SSL out of the box. If you are using Azure Redis Cache which supports SSL out of the box, be sure to set this to true to improve security.</p> <p>The non-SSL port is disabled by default for new caches. Specify true for this setting to use the SSL port. For more information about enabling the non-SSL port, see the Access Ports section in the Configure a cache topic.</p>

ATTRIBUTE	TYPE	DEFAULT	DESCRIPTION	
<i>databaseIdNumber</i>	positive integer	0	<p><i>This attribute can be specified only through either web.config or AppSettings.</i></p> <p>Specify which Redis database to use.</p>	
<i>connectionTimeoutInMilliseconds</i>	positive integer	Provided by StackExchange.Redis	Used to set <i>ConnectTimeout</i> when creating StackExchange.Redis.ConnectionMultiplexer.	
<i>operationTimeoutInMilliseconds</i>	positive integer	Provided by StackExchange.Redis	Used to set <i>SyncTimeout</i> when creating StackExchange.Redis.ConnectionMultiplexer.	
<i>connectionString</i> (Valid StackExchange.Redis connection string)	string	<i>n/a</i>	<p>Either a parameter reference to AppSettings or web.config, or else a valid StackExchange.Redis connection string.</p> <p>This attribute can provide values for <i>host</i>, <i>port</i>, <i>accessKey</i>, <i>ssl</i>, and other StackExchange.Redis attributes. For a closer look at <i>connectionString</i>, see Setting connectionString in the Attribute notes section.</p>	

ATTRIBUTE	TYPE	DEFAULT	DESCRIPTION
<i>settingsClassName</i> <i>settingsMethodName</i>	string string	<i>n/a</i>	<p><i>These attributes can be specified only through either web.config or AppSettings.</i></p> <p>Use these attributes to provide a connection string. <i>settingsClassName</i> should be an assembly qualified class name that contains the method specified by <i>settingsMethodName</i>.</p> <p>The method specified by <i>settingsMethodName</i> should be public, static, and void (not take any parameters), with a return type of string. This method returns the actual connection string.</p>
<i>loggingClassName</i> <i>loggingMethodName</i>	string string	<i>n/a</i>	<p><i>These attributes can be specified only through either web.config or AppSettings.</i></p> <p>Use these attributes to debug your application by providing logs from Session State/Output Cache along with logs from StackExchange.Redis. <i>loggingClassName</i> should be an assembly qualified class name that contains the method specified by <i>loggingMethodName</i>.</p> <p>The method specified by <i>loggingMethodName</i> should be public, static, and void (not take any parameters), with a return type of System.IO.TextWriter.</p>

ATTRIBUTE	TYPE	DEFAULT	DESCRIPTION	
<i>applicationName</i>	string	The module name of the current process or "/"	<p><i>SessionStateProvider only</i> <i>This attribute can be specified only through either web.config or AppSettings.</i></p> <p>The app name prefix to use in Redis cache. The customer may use the same Redis cache for different purposes. To insure that the session keys do not collide, it can be prefixed with the application name.</p>	
<i>throwOnError</i>	boolean	true	<p><i>SessionStateProvider only</i> <i>This attribute can be specified only through either web.config or AppSettings.</i></p> <p>Whether to throw an exception when an error occurs.</p> <p>For more about <i>throwOnError</i>, see Notes on <i>throwOnError</i> in the Attribute notes section.</p>	> Microsoft.Web.Redis.RedisSessionStateProvider.LastException.
<i>retryTimeoutInMilliseconds</i>	positive integer	5000	<p><i>SessionStateProvider only</i> <i>This attribute can be specified only through either web.config or AppSettings.</i></p> <p>How long to retry when an operation fails. If this value is less than <i>operationTimeoutInMilliseconds</i>, the provider will not retry.</p> <p>For more about <i>retryTimeoutInMilliseconds</i>, see Notes on <i>retryTimeoutInMilliseconds</i> in the Attribute notes section.</p>	

ATTRIBUTE	TYPE	DEFAULT	DESCRIPTION
<i>redisSerializerType</i>	string	<i>n/a</i>	Specifies the assembly qualified type name of a class that implements Microsoft.Web.Redis.ISerializer and that contains the custom logic to serialize and deserialize the values. For more information, see About <i>redisSerializerType</i> in the Attribute notes section.

Attribute notes

Setting *connectionString*

The value of *connectionString* is used as key to fetch the actual connection string from AppSettings, if such a string exists in AppSettings. If not found inside AppSettings, the value of *connectionString* will be used as key to fetch actual connection string from the web.config **ConnectionString** section, if that section exists. If the connection string does not exists in AppSettings or the web.config **ConnectionString** section, the literal value of *connectionString* will be used as the connection string when creating StackExchange.Redis.ConnectionMultiplexer.

The following examples illustrate how *connectionString* is used.

Example 1

```
<connectionStrings>
  <add name="MyRedisConnectionString" connectionString="mycache.redis.cache.windows.net:6380,password=actual
access key,ssl=True,abortConnect=False" />
</connectionStrings>
```

In `web.config`, use above key as parameter value instead of actual value.

```
<sessionState mode="Custom" customProvider="MySessionStateStore">
  <providers>
    <add type = "Microsoft.Web.Redis.RedisSessionStateProvide"
        name = "MySessionStateStore"
        connectionString = "MyRedisConnectionString"/>
  </providers>
</sessionState>
```

Example 2

```
<appSettings>
  <add key="MyRedisConnectionString" value="mycache.redis.cache.windows.net:6380,password=actual access
key,ssl=True,abortConnect=False" />
</appSettings>
```

In `web.config`, use above key as parameter value instead of actual value.

```

<sessionState mode="Custom" customProvider="MySessionStateStore">
  <providers>
    <add type = "Microsoft.Web.Redis.RedisSessionStateProvide"
        name = "MySessionStateStore"
        connectionString = "MyRedisConnectionString"/>
  </providers>
</sessionState>

```

Example 3

```

<sessionState mode="Custom" customProvider="MySessionStateStore">
  <providers>
    <add type = "Microsoft.Web.Redis.RedisSessionStateProvide"
        name = "MySessionStateStore"
        connectionString = "mycache.redis.cache.windows.net:6380,password=actual access
key,ssl=True,abortConnect=False"/>
  </providers>
</sessionState>

```

Notes on *throwOnError*

Currently, if an error occurs during a session operation, the session state provider will throw an exception. This shuts down the application.

This behavior has been modified in a way that supports the expectations of existing ASP.NET session state provider users while also providing the ability to act on exceptions, if desired. The default behavior still throws an exception when an error occurs, consistent with other ASP.NET session state providers; existing code should work the same as before.

If you set *throwOnError* to **false**, then instead of throwing an exception when an error occurs, it will fail silently. To see if there was an error and, if so, discover what the exception was, check the static property *Microsoft.Web.Redis.RedisSessionStateProvider.LastException*.

Notes on *retryTimeoutInMilliseconds*

This provides some retry logic to simplify the case where some session operation should retry on failure because of things like network glitch, while also allowing you to control the retry timeout or opt out of retry entirely.

If you set *retryTimeoutInMilliseconds* to a number, for example 2000, then when a session operation fails, it will retry for 2000 milliseconds before treating it as an error. So to have the session state provider to apply this retry logic, just configure the timeout. The first retry will happen after 20 milliseconds, which is sufficient in most cases when a network glitch happens. After that, it will retry every second until it times out. Right after the time out, it will retry one more time to make sure that it won't cut off the timeout by (at most) one second.

If you don't think you need retry (for example, when you are running the Redis server on the same machine as your application) or if you want to handle the retry logic yourself, set *retryTimeoutInMilliseconds* to 0.

About *redisSerializerType*

By default, the serialization to store the values on Redis is done in a binary format provided by the **BinaryFormatter** class. Use *redisSerializerType* to specify the assembly qualified type name of a class that implements **Microsoft.Web.Redis.ISerializer** and has the custom logic to serialize and deserialize the values. For example, here is a Json serializer class using JSON.NET:

```

namespace MyCompany.Redis
{
    public class JsonSerializer : ISerializer
    {
        private static JsonSerializerSettings _settings = new JsonSerializerSettings() { TypeNameHandling = TypeNameHandling.All };

        public byte[] Serialize(object data)
        {
            return Encoding.UTF8.GetBytes(JsonConvert.SerializeObject(data, _settings));
        }

        public object Deserialize(byte[] data)
        {
            if (data == null)
            {
                return null;
            }
            return JsonConvert.DeserializeObject(Encoding.UTF8.GetString(data), _settings);
        }
    }
}

```

Assuming this class is defined in an assembly with name **MyCompanyDll**, you can set the parameter *redisSerializerType* to use it:

```

<sessionState mode="Custom" customProvider="MySessionStateStore">
    <providers>
        <add type = "Microsoft.Web.Redis.RedisSessionStateProvider"
            name = "MySessionStateStore"
            redisSerializerType = "MyCompany.Redis.JsonSerializer,MyCompanyDll"
            ... />
    </providers>
</sessionState>

```

Output cache directive

Add an OutputCache directive to each page for which you wish to cache the output.

```
<%@ OutputCache Duration="60" VaryByParam="*" %>
```

In the previous example, the cached page data remains in the cache for 60 seconds, and a different version of the page is cached for each parameter combination. For more information about the OutputCache directive, see [@OutputCache](#).

Once these steps are performed, your application is configured to use the Redis Output Cache Provider.

Next steps

Check out the [ASP.NET Session State Provider for Azure Cache for Redis](#).

2 minutes to read

How to configure Azure Cache for Redis

1/23/2020 • 18 minutes to read • [Edit Online](#)

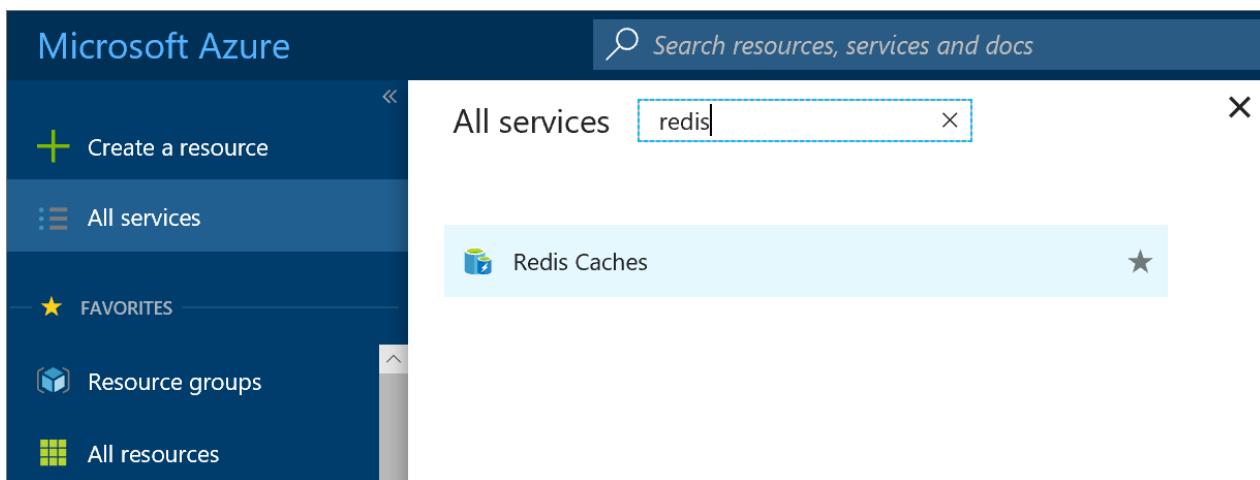
This topic describes the configurations available for your Azure Cache for Redis instances. This topic also covers the default Redis server configuration for Azure Cache for Redis instances.

NOTE

For more information on configuring and using premium cache features, see [How to configure persistence](#), [How to configure clustering](#), and [How to configure Virtual Network support](#).

Configure Azure Cache for Redis settings

If you did not pin your cache to the dashboard, find your cache in the [Azure portal](#) using **All services**.



To view your caches, click **All services** and search for **Azure Cache for Redis**.

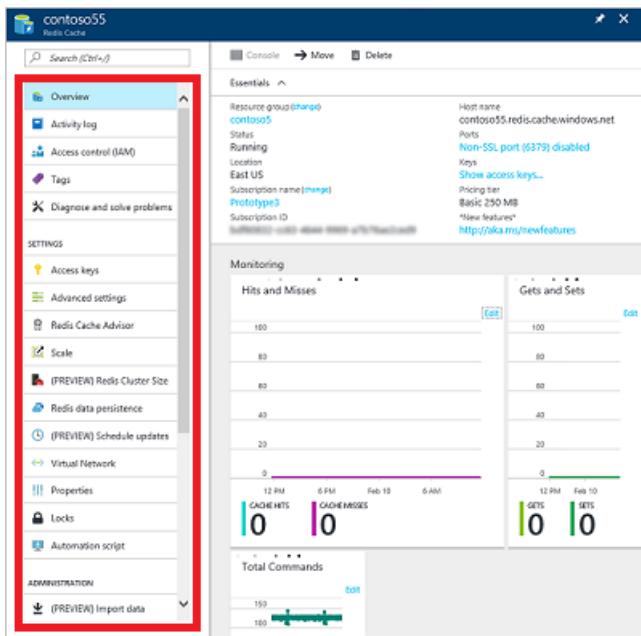
Select the desired cache to view and configure the settings for that cache.

Redis Caches		
Microsoft		
Add	Columns	Refresh
Subscriptions: All 5 selected – Don't see a subscription? Switch directories		
<input type="text" value="Filter items..."/>	All subscriptions	
NAME	LOCATION	SUBSCRIPTION
contoso55	East US	Prototype3

You can view and configure your cache from the **Azure Cache for Redis** blade.

The screenshot shows the Azure Cache for Redis blade for a resource named 'contoso5'. The left sidebar has a 'Resource Menu' with 'Overview' selected. The main area displays 'Essentials' settings including Resource group (contoso5), Status (Running), Location (East US), Subscription name (Prototype3), and Host name (contoso5.redis.cache.windows.net). It also shows Ports (Non-SSL port (6379) disabled), Keys (Show access keys...), Pricing tier (Basic 250 MB), and a link to 'http://aka.ms/newfeatures'.

Azure Cache for Redis settings are viewed and configured on the **Azure Cache for Redis** blade using the **Resource Menu**.



You can view and configure the following settings using the **Resource Menu**.

- [Overview](#)
- [Activity log](#)
- [Access control \(IAM\)](#)
- [Tags](#)
- [Diagnose and solve problems](#)
- [Settings](#)
 - [Access keys](#)
 - [Advanced settings](#)
 - [Azure Cache for Redis Advisor](#)
 - [Scale](#)
 - [Cluster size](#)
 - [Data persistence](#)
 - [Schedule updates](#)
 - [Geo-replication](#)
 - [Virtual Network](#)
 - [Firewall](#)
 - [Properties](#)
 - [Locks](#)
 - [Automation script](#)
- [Administration](#)
 - [Import data](#)
 - [Export data](#)
 - [Reboot](#)
- [Monitoring](#)
 - [Redis metrics](#)
 - [Alert rules](#)
 - [Diagnostics](#)
- [Support & troubleshooting settings](#)
 - [Resource health](#)
 - [New support request](#)

Overview

Overview provides you with basic information about your cache, such as name, ports, pricing tier, and selected cache metrics.

Activity log

Click **Activity log** to view actions performed on your cache. You can also use filtering to expand this view to include other resources. For more information on working with audit logs, see [Audit operations with Resource Manager](#). For more information on monitoring Azure Cache for Redis events, see [Operations and alerts](#).

Access control (IAM)

The **Access control (IAM)** section provides support for role-based access control (RBAC) in the Azure portal. This configuration helps organizations meet their access management requirements simply and precisely. For more information, see [Role-based access control in the Azure portal](#).

Tags

The **Tags** section helps you organize your resources. For more information, see [Using tags to organize your Azure resources](#).

Diagnose and solve problems

Click **Diagnose and solve problems** to be provided with common issues and strategies for resolving them.

Settings

The **Settings** section allows you to access and configure the following settings for your cache.

- [Access keys](#)
- [Advanced settings](#)
- [Azure Cache for Redis Advisor](#)
- [Scale](#)
- [Cluster size](#)
- [Data persistence](#)
- [Schedule updates](#)
- [Geo-replication](#)
- [Virtual Network](#)
- [Firewall](#)
- [Properties](#)
- [Locks](#)
- [Automation script](#)

Access keys

Click **Access keys** to view or regenerate the access keys for your cache. These keys are used by the clients connecting to your cache.

 Regenerate Primary  Regenerate Secondary

Primary
 

Secondary
 

Primary connection string (StackExchange.Redis)
 

Secondary connection string (StackExchange.Redis)
 

For information on other clients see:
<http://aka.ms/redisclients>

Advanced settings

The following settings are configured on the **Advanced settings** blade.

- [Access Ports](#)
- [Memory policies](#)
- [Keyspace notifications \(advanced settings\)](#)

Access Ports

By default, non-SSL access is disabled for new caches. To enable the non-SSL port, click **No** for **Allow access only via SSL** on the **Advanced settings** blade and then click **Save**.

NOTE

SSL access to Azure Cache for Redis supports TLS 1.0, 1.1 and 1.2 currently, but versions 1.0 and 1.1 are being retired soon. Please read our [Remove TLS 1.0 and 1.1 page](#) for more details.

 Save  Discard

Allow access only via SSL

Non-SSL Port

Disabled

SSL Port

6380

Minimum TLS version 

1.2

maxmemory-policy 

volatile-lru



maxmemory-reserved 



50

maxfragmentationmemory-reserved 



50

notify-keyspace-events 



Any changes to the memory reservations will take effect only
when the available memory is higher than the changed memory
reservations. For more information click [here](#).



Memory policies

The **Maxmemory policy**, **maxmemory-reserved**, and **maxfragmentationmemory-reserved** settings on the **Advanced settings** blade configure the memory policies for the cache.

Save Discard

Allow access only via SSL

Non-SSL Port

SSL Port

Minimum TLS version ⓘ

maxmemory-policy ⓘ

maxmemory-reserved ⓘ
 50

maxfragmentationmemory-reserved ⓘ
 50

notify-keyspace-events ⓘ

i Any changes to the memory reservations will take effect only when the available memory is higher than the changed memory reservations. For more information click here.

Maxmemory policy configures the eviction policy for the cache and allows you to choose from the following eviction policies:

- - This is the default eviction policy.
-
-
-
-
-

For more information about `maxmemory` policies, see [Eviction policies](#).

The **maxmemory-reserved** setting configures the amount of memory, in MB, that is reserved for non-cache operations, such as replication during failover. Setting this value allows you to have a more consistent Redis server experience when your load varies. This value should be set higher for workloads that are write heavy. When memory is reserved for such operations, it is unavailable for storage of cached data.

The **maxfragmentationmemory-reserved** setting configures the amount of memory in MB that is reserved to accommodate for memory fragmentation. Setting this value allows you to have a more consistent Redis server experience when the cache is full or close to full and the fragmentation ratio is high. When memory is reserved for such operations, it is unavailable for storage of cached data.

One thing to consider when choosing a new memory reservation value (**maxmemory-reserved** or **maxfragmentationmemory-reserved**) is how this change might affect a cache that is already running with large amounts of data in it. For instance, if you have a 53 GB cache with 49 GB of data, then change the reservation value to 8 GB, this change will drop the max available memory for the system down to 45 GB. If either your current `used_memory` or your `used_memory_rss` values are higher than the new limit of 45 GB, then the system will have to evict data until both `used_memory` and `used_memory_rss` are below 45 GB. Eviction can

increase server load and memory fragmentation. For more information on cache metrics such as `used_memory` and `used_memory_rss`, see [Available metrics and reporting intervals](#).

IMPORTANT

The **maxmemory-reserved** and **maxfragmentationmemory-reserved** settings are only available for Standard and Premium caches.

Keyspace notifications (advanced settings)

Redis keyspace notifications are configured on the **Advanced settings** blade. Keyspace notifications allow clients to receive notifications when certain events occur.

Save Discard

Allow access only via SSL
 Yes No

Non-SSL Port
Disabled

SSL Port
6380

Minimum TLS version ⓘ
Default (currently TLS 1.0)

maxmemory-policy ⓘ
volatile-lru

maxmemory-reserved ⓘ
50

maxfragmentationmemory-reserved ⓘ
50

notify-keyspace-events ⓘ

i Any changes to the memory reservations will take effect only when the available memory is higher than the changed memory reservations. For more information click here.

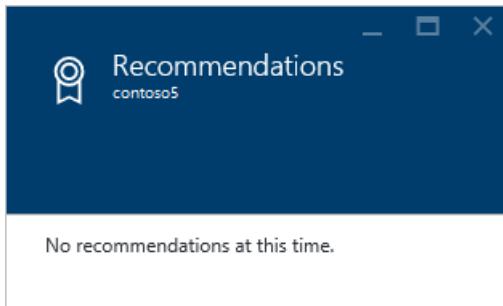
IMPORTANT

Keyspace notifications and the **notify-keyspace-events** setting are only available for Standard and Premium caches.

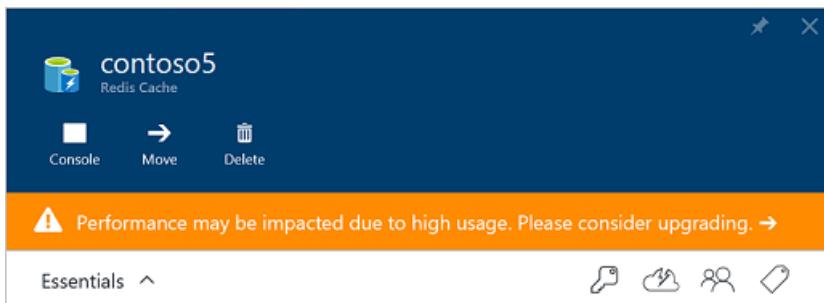
For more information, see [Redis Keyspace Notifications](#). For sample code, see the `KeySpaceNotifications.cs` file in the [Hello world](#) sample.

Azure Cache for Redis Advisor

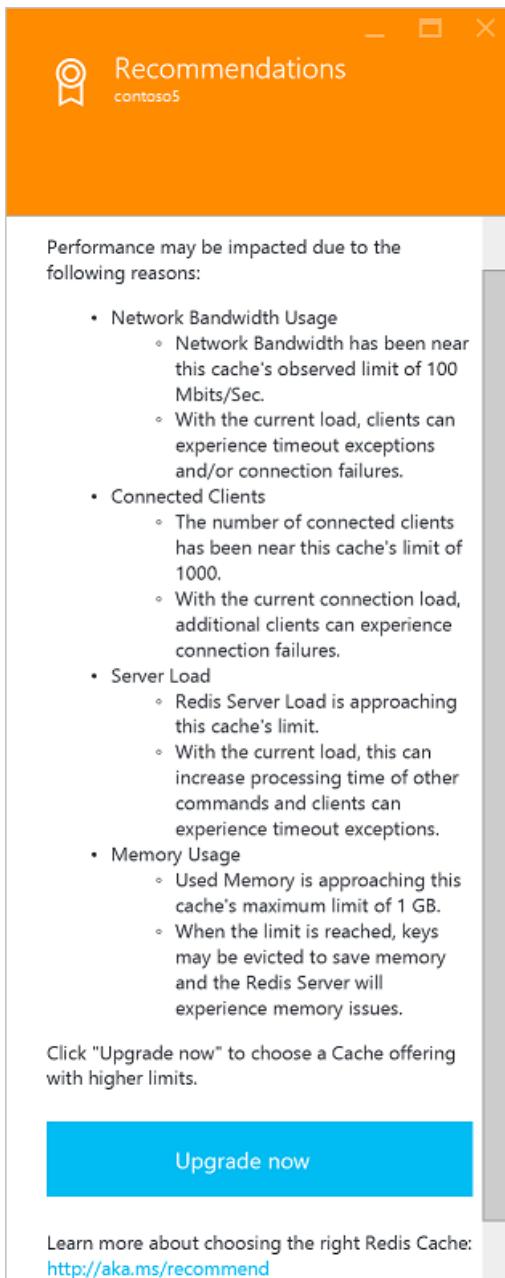
The **Azure Cache for Redis Advisor** blade displays recommendations for your cache. During normal operations, no recommendations are displayed.



If any conditions occur during the operations of your cache such as high memory usage, network bandwidth, or server load, an alert is displayed on the **Azure Cache for Redis** blade.



Further information can be found on the **Recommendations** blade.



You can monitor these metrics on the [Monitoring charts](#) and [Usage charts](#) sections of the **Azure Cache for Redis** blade.

Each pricing tier has different limits for client connections, memory, and bandwidth. If your cache approaches maximum capacity for these metrics over a sustained period of time, a recommendation is created. For more information about the metrics and limits reviewed by the **Recommendations** tool, see the following table:

AZURE CACHE FOR REDIS METRIC	MORE INFORMATION
Network bandwidth usage	Cache performance - available bandwidth
Connected clients	Default Redis server configuration - maxclients
Server load	Usage charts - Redis Server Load
Memory usage	Cache performance - size

To upgrade your cache, click **Upgrade now** to change the pricing tier and [scale](#) your cache. For more information on choosing a pricing tier, see [What Azure Cache for Redis offering and size should I use?](#)

Scale

Click **Scale** to view or change the pricing tier for your cache. For more information on scaling, see [How to Scale Azure Cache for Redis](#).

Select pricing tier

You can scale your cache instance up or down. [Learn more](#)

★ Recommended | View all

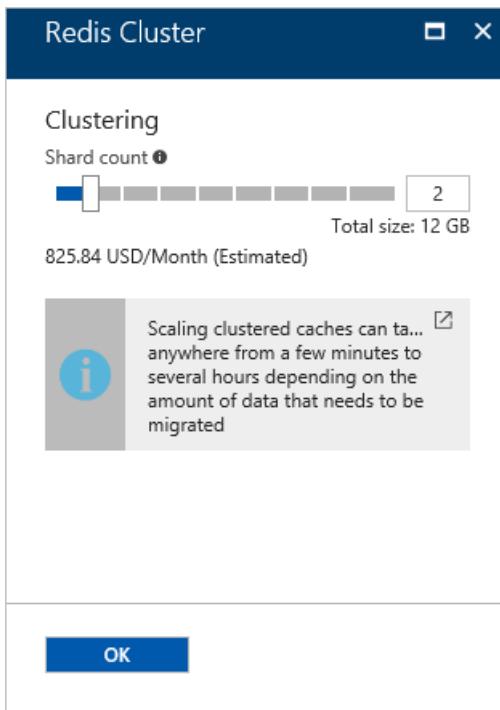
P1 Premium	P2 Premium	P3 Premium
6 GB Cache	13 GB Cache	26 GB Cache
Replication	Replication	Replication
Moderate network bandwidth	Moderate network bandwidth	High network bandwidth
All Standard features	All Standard features	All Standard features
Data Persistence	Data Persistence	Data Persistence
Virtual Network	Virtual Network	Virtual Network
Redis Cluster	Redis Cluster	Redis Cluster
99.9% SLA	99.9% SLA	99.9% SLA
412.92	825.84	1,650.94
USD/MONTH (ESTIMATED) PER SH...	USD/MONTH (ESTIMATED) PER SH...	USD/MONTH (ESTIMATED) PER SH...

P4 Premium	C1 Standard	C2 Standard
53 GB Cache	1 GB Cache	2.5 GB Cache
Replication	Replication	Replication
Highest network bandwidth	Low network bandwidth	Moderate network bandwidth
All Standard features	Dedicated service	Dedicated service
Data Persistence	SSL	SSL
Virtual Network	Up to 1,000 connections	Up to 2,000 connections
Redis Cluster	Configure Redis (keys)	Configure Redis (keys)

Select

Redis Cluster Size

Click **Cluster Size** to change the cluster size for a running premium cache with clustering enabled.



To change the cluster size, use the slider or type a number between 1 and 10 in the **Shard count** text box and click **OK** to save.

IMPORTANT

Redis clustering is only available for Premium caches. For more information, see [How to configure clustering for a Premium Azure Cache for Redis](#).

Redis data persistence

Click **Data persistence** to enable, disable, or configure data persistence for your premium cache. Azure Cache for Redis offers Redis persistence using either [RDB persistence](#) or [AOF persistence](#).

For more information, see [How to configure persistence for a Premium Azure Cache for Redis](#).

IMPORTANT

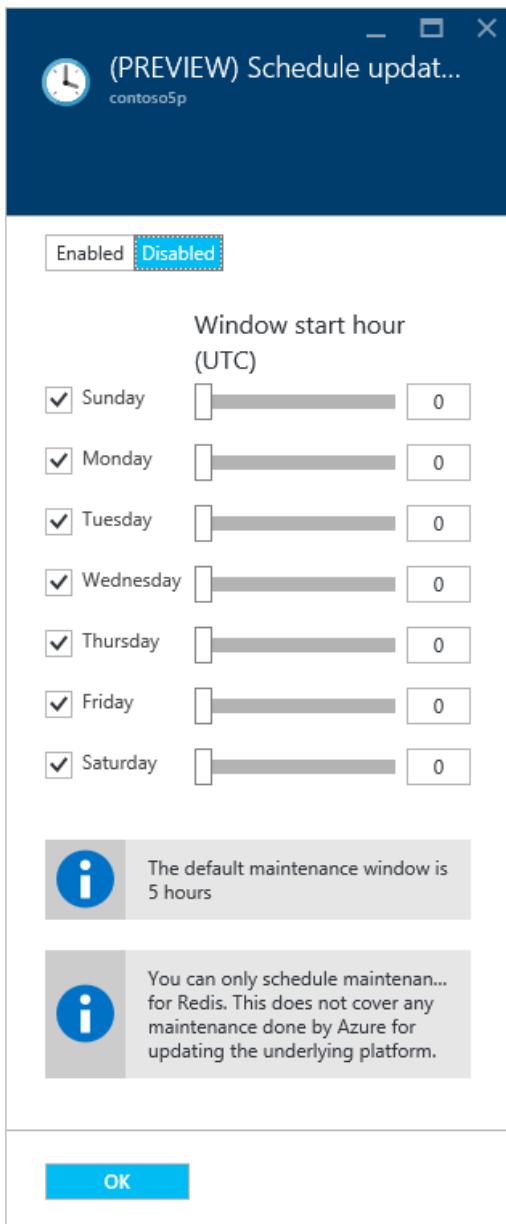
Redis data persistence is only available for Premium caches.

Schedule updates

The **Schedule updates** blade allows you to designate a maintenance window for Redis server updates for your cache.

IMPORTANT

The maintenance window applies only to Redis server updates, and not to any Azure updates or updates to the operating system of the VMs that host the cache.



To specify a maintenance window, check the desired days and specify the maintenance window start hour for each day, and click **OK**. The maintenance window time is in UTC.

IMPORTANT

The **Schedule updates** functionality is only available for Premium tier caches. For more information and instructions, see [Azure Cache for Redis administration - Schedule updates](#).

Geo-replication

The **Geo-replication** blade provides a mechanism for linking two Premium tier Azure Cache for Redis instances. One cache is designated as the primary linked cache, and the other as the secondary linked cache. The secondary linked cache becomes read-only, and data written to the primary cache is replicated to the secondary linked cache. This functionality can be used to replicate a cache across Azure regions.

IMPORTANT

Geo-replication is only available for Premium tier caches. For more information and instructions, see [How to configure Geo-replication for Azure Cache for Redis](#).

Virtual Network

The **Virtual Network** section allows you to configure the virtual network settings for your cache. For information on creating a premium cache with VNET support and updating its settings, see [How to configure Virtual Network Support for a Premium Azure Cache for Redis](#).

IMPORTANT

Virtual network settings are only available for premium caches that were configured with VNET support during cache creation.

Firewall

Firewall rules configuration is available for all Azure Cache for Redis tiers.

Click **Firewall** to view and configure firewall rules for cache.

RULE NAME	START IP ADDRESS	END IP ADDRESS	...
Contoso domains	125.50.75.1	125.50.71.99	...

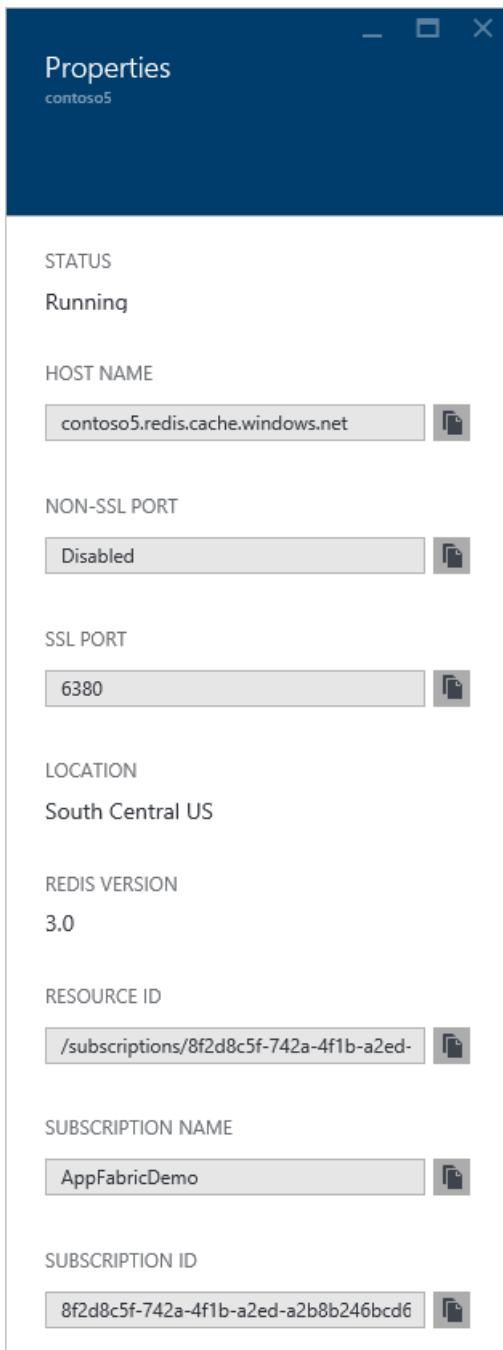
You can specify firewall rules with a start and end IP address range. When firewall rules are configured, only client connections from the specified IP address ranges can connect to the cache. When a firewall rule is saved, there is a short delay before the rule is effective. This delay is typically less than one minute.

IMPORTANT

Connections from Azure Cache for Redis monitoring systems are always permitted, even if firewall rules are configured.

Properties

Click **Properties** to view information about your cache, including the cache endpoint and ports.



Locks

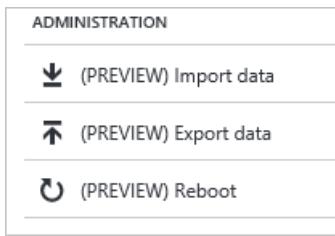
The **Locks** section allows you to lock a subscription, resource group, or resource to prevent other users in your organization from accidentally deleting or modifying critical resources. For more information, see [Lock resources with Azure Resource Manager](#).

Automation script

Click **Automation script** to build and export a template of your deployed resources for future deployments. For more information about working with templates, see [Deploy resources with Azure Resource Manager templates](#).

Administration settings

The settings in the **Administration** section allow you to perform the following administrative tasks for your cache.



- [Import data](#)
- [Export data](#)
- [Reboot](#)

Import/Export

Import/Export is an Azure Cache for Redis data management operation, which allows you to import and export data in the cache by importing and exporting an Azure Cache for Redis Database (RDB) snapshot from a premium cache to a page blob in an Azure Storage Account. Import/Export enables you to migrate between different Azure Cache for Redis instances or populate the cache with data before use.

Import can be used to bring Redis compatible RDB files from any Redis server running in any cloud or environment, including Redis running on Linux, Windows, or any cloud provider such as Amazon Web Services and others. Importing data is an easy way to create a cache with pre-populated data. During the import process, Azure Cache for Redis loads the RDB files from Azure storage into memory, and then inserts the keys into the cache.

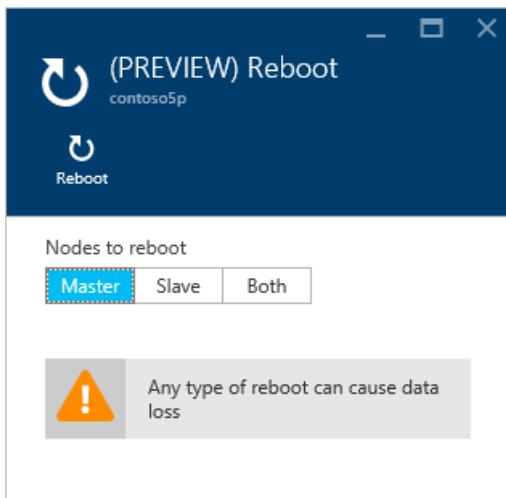
Export allows you to export the data stored in Azure Cache for Redis to Redis compatible RDB files. You can use this feature to move data from one Azure Cache for Redis instance to another or to another Redis server. During the export process, a temporary file is created on the VM that hosts the Azure Cache for Redis server instance, and the file is uploaded to the designated storage account. When the export operation completes with either a status of success or failure, the temporary file is deleted.

IMPORTANT

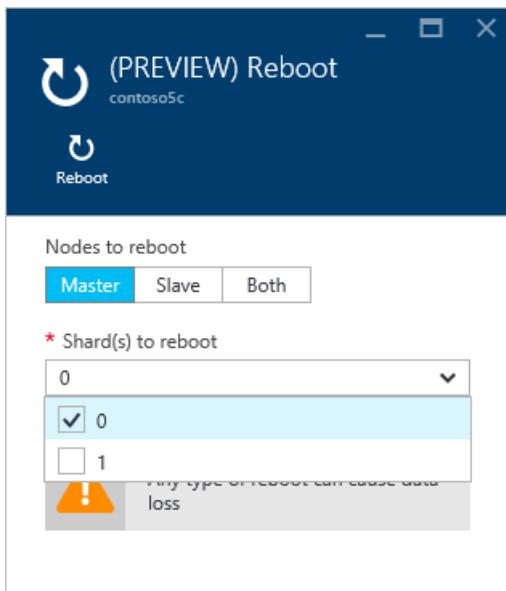
Import/Export is only available for Premium tier caches. For more information and instructions, see [Import and Export data in Azure Cache for Redis](#).

Reboot

The **Reboot** blade allows you to reboot the nodes of your cache. This reboot capability enables you to test your application for resiliency if there is a failure of a cache node.



If you have a premium cache with clustering enabled, you can select which shards of the cache to reboot.



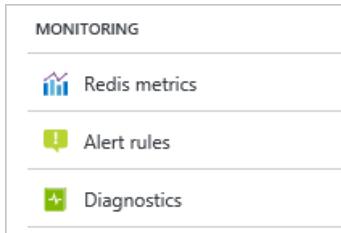
To reboot one or more nodes of your cache, select the desired nodes and click **Reboot**. If you have a premium cache with clustering enabled, select the shard(s) to reboot and then click **Reboot**. After a few minutes, the selected node(s) reboot, and are back online a few minutes later.

IMPORTANT

Reboot is now available for all pricing tiers. For more information and instructions, see [Azure Cache for Redis administration](#) - [Reboot](#).

Monitoring

The **Monitoring** section allows you to configure diagnostics and monitoring for your Azure Cache for Redis. For more information on Azure Cache for Redis monitoring and diagnostics, see [How to monitor Azure Cache for Redis](#).



- [Redis metrics](#)
- [Alert rules](#)
- [Diagnostics](#)

Redis metrics

Click **Redis metrics** to [view metrics](#) for your cache.

Alert rules

Click **Alert rules** to configure alerts based on Azure Cache for Redis metrics. For more information, see [Alerts](#).

Diagnostics

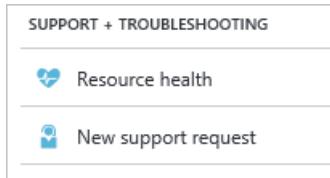
By default, cache metrics in Azure Monitor are [stored for 30 days](#) and then deleted. To persist your cache metrics for longer than 30 days, click **Diagnostics** to [configure the storage account](#) used to store cache diagnostics.

NOTE

In addition to archiving your cache metrics to storage, you can also [stream them to an Event hub or send them to Azure Monitor logs](#).

Support & troubleshooting settings

The settings in the **Support + troubleshooting** section provide you with options for resolving issues with your cache.



- [Resource health](#)
- [New support request](#)

Resource health

Resource health watches your resource and tells you if it's running as expected. For more information about the Azure Resource health service, see [Azure Resource health overview](#).

NOTE

Resource health is currently unable to report on the health of Azure Cache for Redis instances hosted in a virtual network.

For more information, see [Do all cache features work when hosting a cache in a VNET?](#)

New support request

Click **New support request** to open a support request for your cache.

Default Redis server configuration

New Azure Cache for Redis instances are configured with the following default Redis configuration values:

NOTE

The settings in this section cannot be changed using the `StackExchange.Redis.IServer.ConfigSet` method. If this method is called with one of the commands in this section, an exception similar to the following example is thrown:

```
StackExchange.Redis.RedisServerException: ERR unknown command 'CONFIG'
```

Any values that are configurable, such as **max-memory-policy**, are configurable through the Azure portal or command-line management tools such as Azure CLI or PowerShell.

SETTING	DEFAULT VALUE	DESCRIPTION
---------	---------------	-------------

Setting	Default Value	Description
<code>databases</code>	16	The default number of databases is 16 but you can configure a different number based on the pricing tier. ¹ The default database is DB 0, you can select a different one on a per-connection basis using <code>connection.GetDatabase(dbid)</code> where <code>dbid</code> is a number between <code>0</code> and <code>databases - 1</code> .
<code>maxclients</code>	Depends on the pricing tier ²	This value is the maximum number of connected clients allowed at the same time. Once the limit is reached Redis closes all the new connections, returning a 'max number of clients reached' error.
<code>maxmemory-policy</code>	<code>volatile-lru</code>	Maxmemory policy is the setting for how Redis selects what to remove when <code>maxmemory</code> (the size of the cache offering you selected when you created the cache) is reached. With Azure Cache for Redis the default setting is <code>volatile-lru</code> , which removes the keys with an expiration set using an LRU algorithm. This setting can be configured in the Azure portal. For more information, see Memory policies .
<code>maxmemory-samples</code>	3	To save memory, LRU and minimal TTL algorithms are approximated algorithms instead of precise algorithms. By default Redis checks three keys and picks the one that was used less recently.
<code>lua-time-limit</code>	5,000	Max execution time of a Lua script in milliseconds. If the maximum execution time is reached, Redis logs that a script is still in execution after the maximum allowed time, and starts to reply to queries with an error.
<code>lua-event-limit</code>	500	Max size of script event queue.
<code>client-output-buffer-limit</code> <code>normalclient-output-buffer-limit</code> <code>pubsub</code>	0 0 032mb 8mb 60	The client output buffer limits can be used to force disconnection of clients that are not reading data from the server fast enough for some reason (a common reason is that a Pub/Sub client can't consume messages as fast as the publisher can produce them). For more information, see https://redis.io/topics/clients .

¹The limit for `databases` is different for each Azure Cache for Redis pricing tier and can be set at cache creation. If no `databases` setting is specified during cache creation, the default is 16.

- Basic and Standard caches
 - C0 (250 MB) cache - up to 16 databases
 - C1 (1 GB) cache - up to 16 databases
 - C2 (2.5 GB) cache - up to 16 databases
 - C3 (6 GB) cache - up to 16 databases
 - C4 (13 GB) cache - up to 32 databases
 - C5 (26 GB) cache - up to 48 databases
 - C6 (53 GB) cache - up to 64 databases
- Premium caches
 - P1 (6 GB - 60 GB) - up to 16 databases
 - P2 (13 GB - 130 GB) - up to 32 databases
 - P3 (26 GB - 260 GB) - up to 48 databases
 - P4 (53 GB - 530 GB) - up to 64 databases
 - All premium caches with Redis cluster enabled - Redis cluster only supports use of database 0 so the `databases` limit for any premium cache with Redis cluster enabled is effectively 1 and the [Select command](#) is not allowed. For more information, see [Do I need to make any changes to my client application to use clustering?](#)

For more information about databases, see [What are Redis databases?](#)

NOTE

The `databases` setting can be configured only during cache creation and only using PowerShell, CLI, or other management clients. For an example of configuring `databases` during cache creation using PowerShell, see [New-AzRedisCache](#).

² `maxclients` is different for each Azure Cache for Redis pricing tier.

- Basic and Standard caches
 - C0 (250 MB) cache - up to 256 connections
 - C1 (1 GB) cache - up to 1,000 connections
 - C2 (2.5 GB) cache - up to 2,000 connections
 - C3 (6 GB) cache - up to 5,000 connections
 - C4 (13 GB) cache - up to 10,000 connections
 - C5 (26 GB) cache - up to 15,000 connections
 - C6 (53 GB) cache - up to 20,000 connections
- Premium caches
 - P1 (6 GB - 60 GB) - up to 7,500 connections
 - P2 (13 GB - 130 GB) - up to 15,000 connections
 - P3 (26 GB - 260 GB) - up to 30,000 connections
 - P4 (53 GB - 530 GB) - up to 40,000 connections

NOTE

While each size of cache allows *up to* a certain number of connections, each connection to Redis has overhead associated with it. An example of such overhead would be CPU and memory usage as a result of TLS/SSL encryption. The maximum connection limit for a given cache size assumes a lightly loaded cache. If load from connection overhead *plus* load from client operations exceeds capacity for the system, the cache can experience capacity issues even if you have not exceeded the connection limit for the current cache size.

Redis commands not supported in Azure Cache for Redis

IMPORTANT

Because configuration and management of Azure Cache for Redis instances is managed by Microsoft, the following commands are disabled. If you try to invoke them, you receive an error message similar to

```
"(error) ERR unknown command".
```

- BGREWRITEAOF
- BGSAVE
- CONFIG
- DEBUG
- MIGRATE
- SAVE
- SHUTDOWN
- SLAVEOF
- CLUSTER - Cluster write commands are disabled, but read-only Cluster commands are permitted.

For more information about Redis commands, see <https://redis.io/commands>.

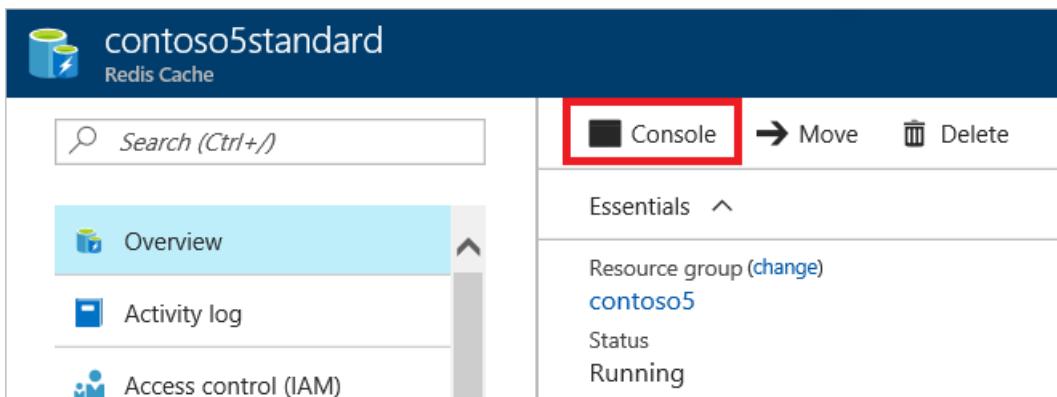
Redis console

You can securely issue commands to your Azure Cache for Redis instances using the **Redis Console**, which is available in the Azure portal for all cache tiers.

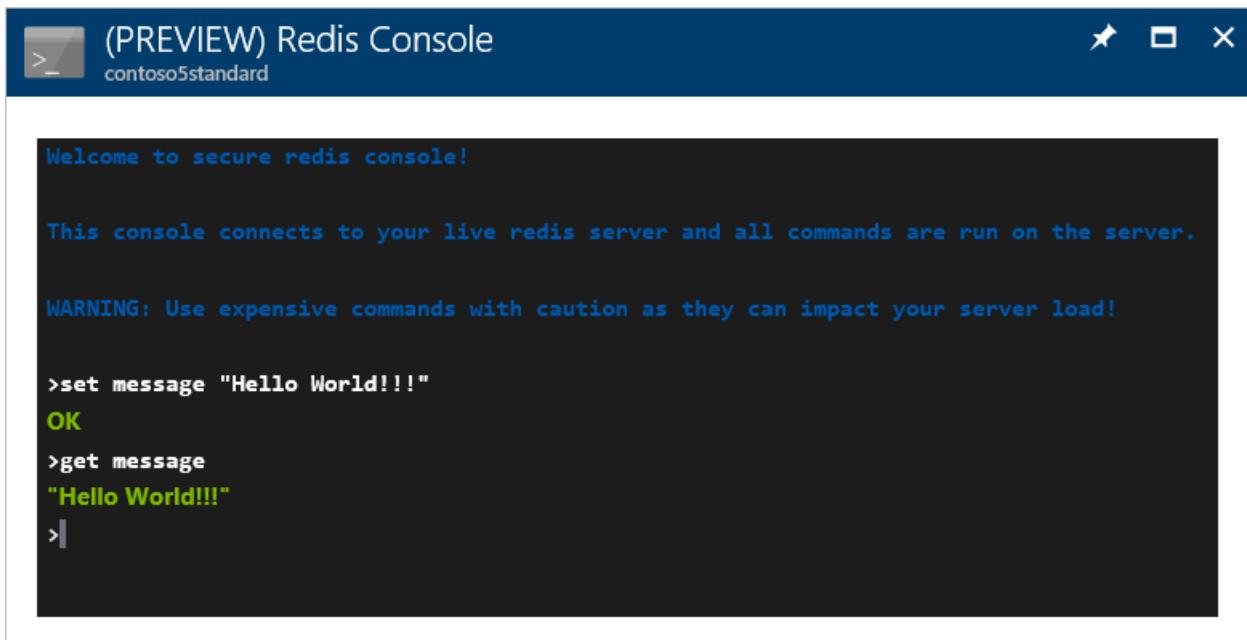
IMPORTANT

- The Redis Console does not work with **VNET**. When your cache is part of a VNET, only clients in the VNET can access the cache. Because Redis Console runs in your local browser, which is outside the VNET, it can't connect to your cache.
- Not all Redis commands are supported in Azure Cache for Redis. For a list of Redis commands that are disabled for Azure Cache for Redis, see the previous [Redis commands not supported in Azure Cache for Redis](#) section. For more information about Redis commands, see <https://redis.io/commands>.

To access the Redis Console, click **Console** from the **Azure Cache for Redis** blade.



To issue commands against your cache instance, type the desired command into the console.



(PREVIEW) Redis Console
contoso5standard

```
Welcome to secure redis console!

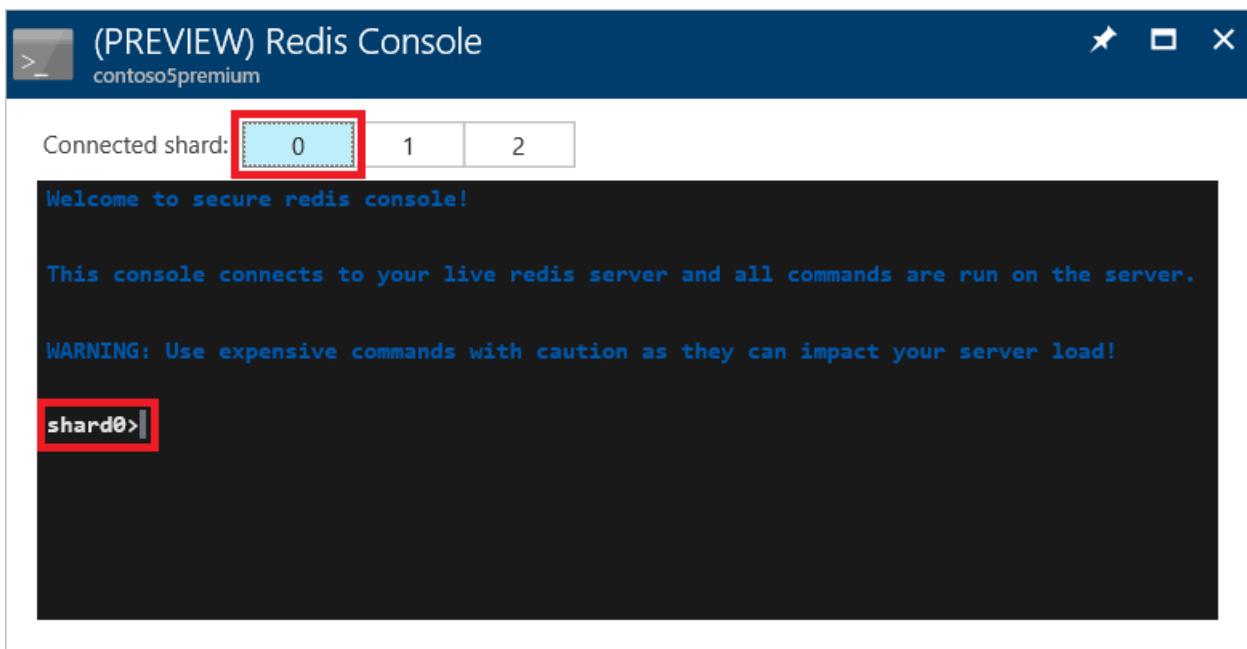
This console connects to your live redis server and all commands are run on the server.

WARNING: Use expensive commands with caution as they can impact your server load!

>set message "Hello World!!!"
OK
>get message
"Hello World!!!"
>
```

Using the Redis Console with a premium clustered cache

When using the Redis Console with a premium clustered cache, you can issue commands to a single shard of the cache. To issue a command to a specific shard, first connect to the desired shard by clicking it on the shard picker.



(PREVIEW) Redis Console
contoso5premium

Connected shard:

```
Welcome to secure redis console!

This console connects to your live redis server and all commands are run on the server.

WARNING: Use expensive commands with caution as they can impact your server load!

shard0>
```

If you attempt to access a key that is stored in a different shard than the connected shard, you receive an error message similar to the following message:

```
shard1>get myKey
(error) MOVED 866 13.90.202.154:13000 (shard 0)
```

In the previous example, shard 1 is the selected shard, but `myKey` is located in shard 0, as indicated by the `(shard 0)` portion of the error message. In this example, to access `myKey`, select shard 0 using the shard picker, and then issue the desired command.

Move your cache to a new subscription

You can move your cache to a new subscription by clicking **Move**.

The screenshot shows the Azure portal interface for a Redis Cache named 'contoso5standard'. At the top, there's a search bar labeled 'Search (Ctrl+ /)' and a navigation bar with 'Console', 'Move' (which is highlighted with a red box), and 'Delete' buttons. Below this, a sidebar lists 'Overview', 'Activity log', and 'Access control (IAM)'. The main content area is titled 'Essentials' and shows 'Resource group (change) contoso5', 'Status Running', and other details. A vertical scroll bar is visible on the right side of the main content area.

For information on moving resources from one resource group to another, and from one subscription to another, see [Move resources to new resource group or subscription](#).

Next steps

- For more information on working with Redis commands, see [How can I run Redis commands?](#)

Import and Export data in Azure Cache for Redis

1/23/2020 • 7 minutes to read • [Edit Online](#)

Import/Export is an Azure Cache for Redis data management operation, which allows you to import data into Azure Cache for Redis or export data from Azure Cache for Redis by importing and exporting an Azure Cache for Redis Database (RDB) snapshot from a premium cache to a blob in an Azure Storage Account.

- **Export** - you can export your Azure Cache for Redis RDB snapshots to a Page Blob.
- **Import** - you can import your Azure Cache for Redis RDB snapshots from either a Page Blob or a Block Blob.

Import/Export enables you to migrate between different Azure Cache for Redis instances or populate the cache with data before use.

This article provides a guide for importing and exporting data with Azure Cache for Redis and provides the answers to commonly asked questions.

IMPORTANT

Import/Export is only available for [premium tier](#) caches.

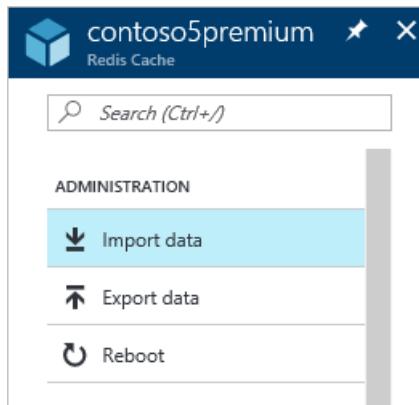
Import

Import can be used to bring Redis compatible RDB files from any Redis server running in any cloud or environment, including Redis running on Linux, Windows, or any cloud provider such as Amazon Web Services and others. Importing data is an easy way to create a cache with pre-populated data. During the import process, Azure Cache for Redis loads the RDB files from Azure storage into memory and then inserts the keys into the cache.

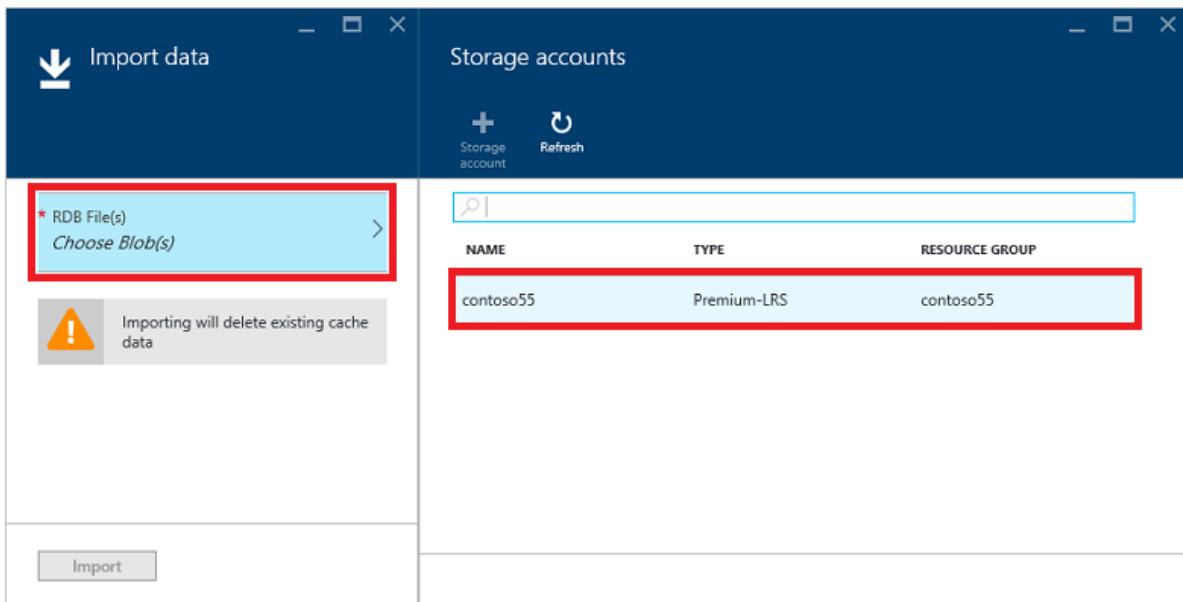
NOTE

Before beginning the import operation, ensure that your Redis Database (RDB) file or files are uploaded into page or block blobs in Azure storage, in the same region and subscription as your Azure Cache for Redis instance. For more information, see [Get started with Azure Blob storage](#). If you exported your RDB file using the [Azure Cache for Redis Export](#) feature, your RDB file is already stored in a page blob and is ready for importing.

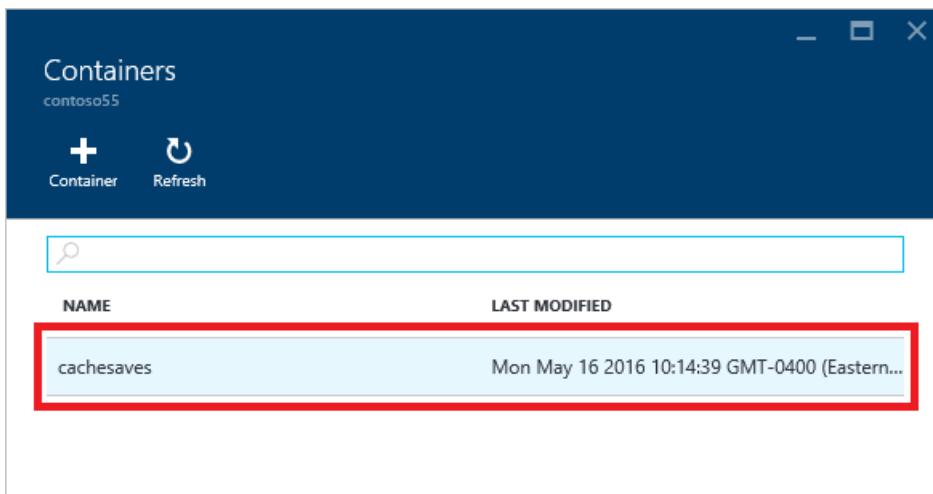
1. To import one or more exported cache blobs, [browse to your cache](#) in the Azure portal and click **Import data** from the **Resource menu**.



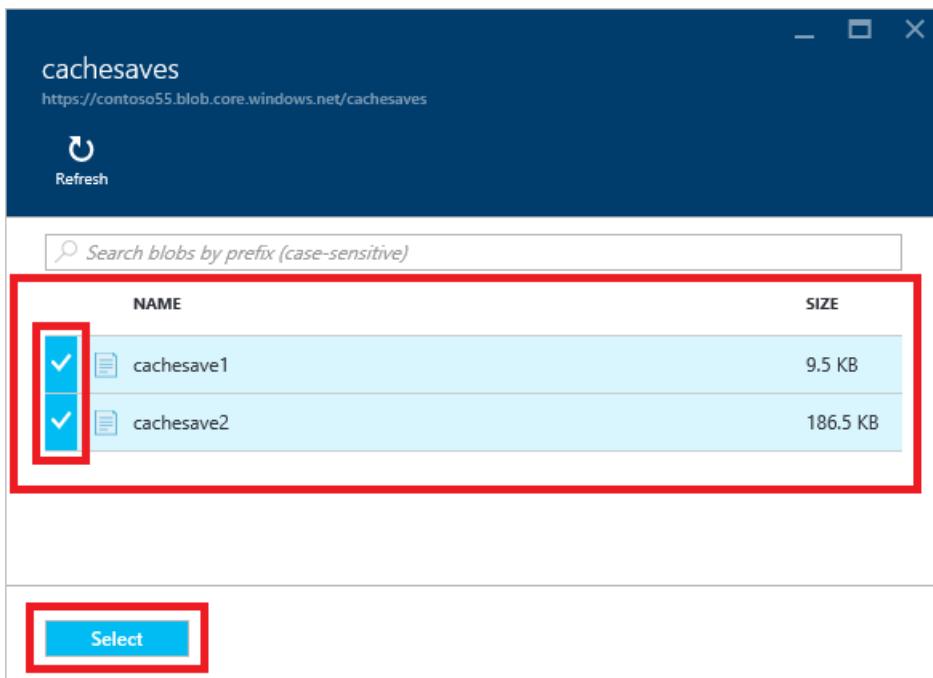
2. Click **Choose Blob(s)** and select the storage account that contains the data to import.



3. Click the container that contains the data to import.



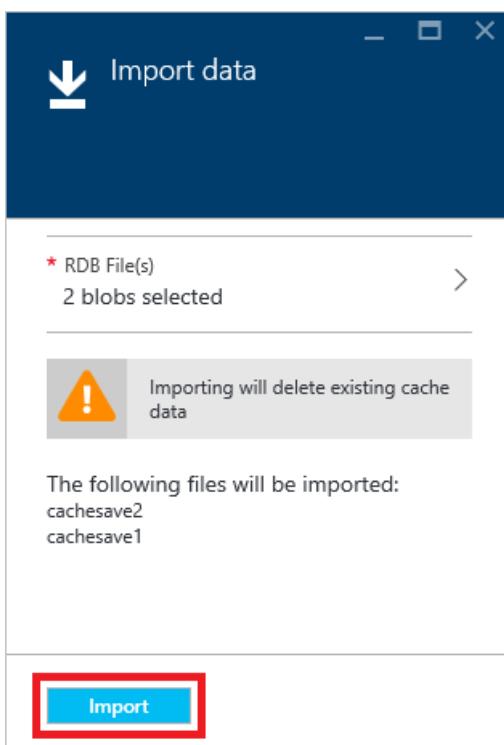
4. Select one or more blobs to import by clicking the area to the left of the blob name, and then click **Select**.



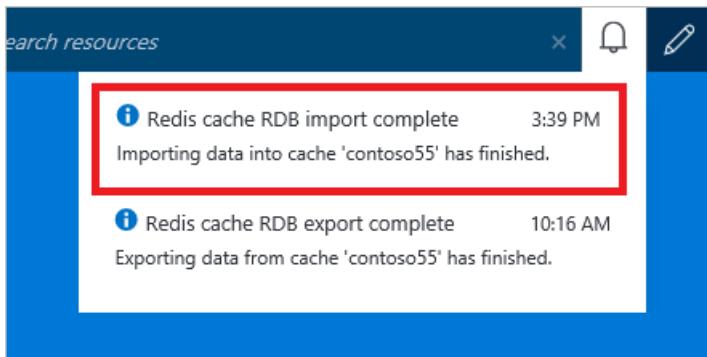
5. Click **Import** to begin the import process.

IMPORTANT

The cache is not accessible by cache clients during the import process, and any existing data in the cache is deleted.



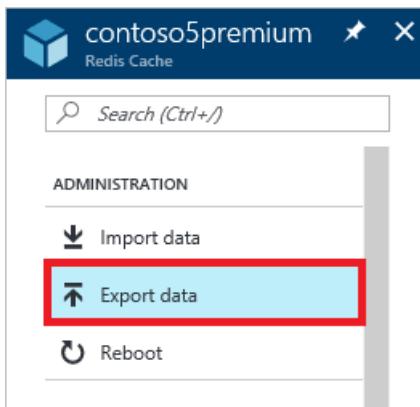
You can monitor the progress of the import operation by following the notifications from the Azure portal, or by viewing the events in the [audit log](#).



Export

Export allows you to export the data stored in Azure Cache for Redis to Redis compatible RDB file(s). You can use this feature to move data from one Azure Cache for Redis instance to another or to another Redis server. During the export process, a temporary file is created on the VM that hosts the Azure Cache for Redis server instance, and the file is uploaded to the designated storage account. When the export operation completes with either a status of success or failure, the temporary file is deleted.

1. To export the current contents of the cache to storage, [browse to your cache](#) in the Azure portal and click **Export data** from the **Resource menu**.



2. Click **Choose Storage Container** and select the desired storage account. The storage account must be in the same subscription and region as your cache.

IMPORTANT

Export works with page blobs, which are supported by both classic and Resource Manager storage accounts, but are not supported by Blob storage accounts at this time. For more information, see [Azure storage account overview](#).

NAME	TYPE	RESOURCE GROUP
contoso55	Premium-LRS	contoso55

3. Choose the desired blob container and click **Select**. To use new a container, click **Add Container** to add it first and then select it from the list.

The screenshot shows the 'Containers' blade in the Azure portal. At the top, there's a search bar labeled 'Search containers by prefix'. Below it is a table with two columns: 'NAME' and 'LAST MODIFIED'. A single row for 'cachesaves' is listed, with its entire row highlighted by a red box. The 'LAST MODIFIED' column shows the date and time: 'Mon May 16 2016 10:14:39 GMT-0400 (Eastern...)'. At the bottom of the table, there's a blue 'Select' button, which is also highlighted with a red box.

4. Type a **Blob name prefix** and click **Export** to start the export process. The blob name prefix is used to prefix the names of files generated by this export operation.

The screenshot shows the 'Export data' dialog. It has a title bar with an upward arrow icon and the text 'Export data'. Below that is a sub-header 'Export to RDB file'. The main area contains two fields with red boxes around them:

- '* Blob name prefix' containing 'cachesave1' with a green checkmark icon.
- '* Export output' containing 'cachesaves' with a right-pointing arrow icon.

Below these fields is an information message: 'Data will be exported as page blobs' with an info icon. At the bottom is a blue 'Export' button, which is also highlighted with a red box.

You can monitor the progress of the export operation by following the notifications from the Azure portal, or by viewing the events in the [audit log](#).

The screenshot shows the Azure portal audit log. It lists three events:

- A blue info icon next to the text 'Redis cache RDB export complete 10:16 AM' with the subtext 'Exporting data from cache 'contoso55' has finished.'
- A blue info icon next to the text 'Successfully created storage container 10:14 AM' with the subtext 'Successfully created storage container 'cachesaves'.'
- A green checkmark icon next to the text 'Successfully updated diagnostics settings 10:04 AM' with the subtext 'Successfully updated diagnostics settings for contoso55.'

Each event is highlighted with a red box.

Caches remain available for use during the export process.

Import/Export FAQ

This section contains frequently asked questions about the Import/Export feature.

- [What pricing tiers can use Import/Export?](#)
- [Can I import data from any Redis server?](#)
- [What RDB versions can I import?](#)
- [Is my cache available during an Import/Export operation?](#)
- [Can I use Import/Export with Redis cluster?](#)
- [How does Import/Export work with a custom databases setting?](#)
- [How is Import/Export different from Redis persistence?](#)
- [Can I automate Import/Export using PowerShell, CLI, or other management clients?](#)
- [I received a timeout error during my Import/Export operation. What does it mean?](#)
- [I got an error when exporting my data to Azure Blob Storage. What happened?](#)

What pricing tiers can use Import/Export?

Import/Export is available only in the premium pricing tier.

Can I import data from any Redis server?

Yes, in addition to importing data exported from Azure Cache for Redis instances, you can import RDB files from any Redis server running in any cloud or environment, such as Linux, Windows, or cloud providers such as Amazon Web Services. To do this, upload the RDB file from the desired Redis server into a page or block blob in an Azure Storage Account, and then import it into your premium Azure Cache for Redis instance. For example, you may want to export the data from your production cache and import it into a cache used as part of a staging environment for testing or migration.

IMPORTANT

To successfully import data exported from Redis servers other than Azure Cache for Redis when using a page blob, the page blob size must be aligned on a 512 byte boundary. For sample code to perform any required byte padding, see [Sample page blob upload](#).

What RDB versions can I import?

Azure Cache for Redis supports RDB import up through RDB version 7.

Is my cache available during an Import/Export operation?

- **Export** - Caches remain available and you can continue to use your cache during an export operation.
- **Import** - Caches become unavailable when an import operation starts, and become available for use when the import operation completes.

Can I use Import/Export with Redis cluster?

Yes, and you can import/export between a clustered cache and a non-clustered cache. Since Redis cluster [only supports database 0](#), any data in databases other than 0 isn't imported. When clustered cache data is imported, the keys are redistributed among the shards of the cluster.

How does Import/Export work with a custom databases setting?

Some pricing tiers have different [databases limits](#), so there are some considerations when importing if you configured a custom value for the `databases` setting during cache creation.

- When importing to a pricing tier with a lower `databases` limit than the tier from which you exported:
 - If you are using the default number of `databases`, which is 16 for all pricing tiers, no data is lost.

- If you are using a custom number of databases that falls within the limits for the tier to which you are importing, no data is lost.
- If your exported data contained data in a database that exceeds the limits of the new tier, the data from those higher databases is not imported.

How is Import/Export different from Redis persistence?

Azure Cache for Redis persistence allows you to persist data stored in Redis to Azure Storage. When persistence is configured, Azure Cache for Redis persists a snapshot of the Azure Cache for Redis in a Redis binary format to disk based on a configurable backup frequency. If a catastrophic event occurs that disables both the primary and replica cache, the cache data is restored automatically using the most recent snapshot. For more information, see [How to configure data persistence for a Premium Azure Cache for Redis](#).

Import/ Export allows you to bring data into or export from Azure Cache for Redis. It does not configure backup and restore using Redis persistence.

Can I automate Import/Export using PowerShell, CLI, or other management clients?

Yes, for PowerShell instructions see [To import an Azure Cache for Redis](#) and [To export an Azure Cache for Redis](#).

I received a timeout error during my Import/Export operation. What does it mean?

If you remain on the **Import data** or **Export data** blade for longer than 15 minutes before initiating the operation, you receive an error with an error message similar to the following example:

The request to import data into cache 'contoso55' failed with status 'error' and error 'One of the SAS URIs provided could not be used for the following reason: The SAS token end time (se) must be at least 1 hour from now and the start time (st), if given, must be at least 15 minutes in the past.'

To resolve this, initiate the import or export operation before 15 minutes has elapsed.

I got an error when exporting my data to Azure Blob Storage. What happened?

Export works only with RDB files stored as page blobs. Other blob types are not currently supported, including Blob storage accounts with hot and cool tiers. For more information, see [Azure storage account overview](#).

Next steps

Learn how to use more premium cache features.

- [Introduction to the Azure Cache for Redis Premium tier](#)

How to administer Azure Cache for Redis

11/15/2019 • 5 minutes to read • [Edit Online](#)

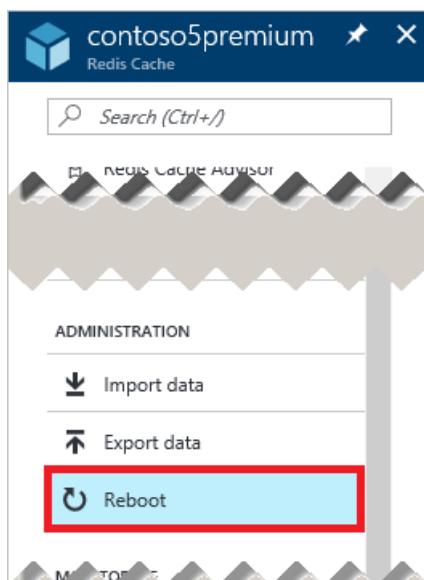
This topic describes how to perform administration tasks such as [rebooting](#) and [scheduling updates](#) for your Azure Cache for Redis instances.

NOTE

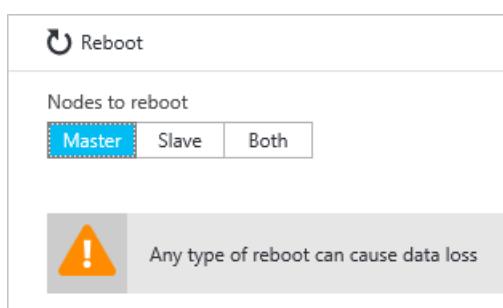
This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

Reboot

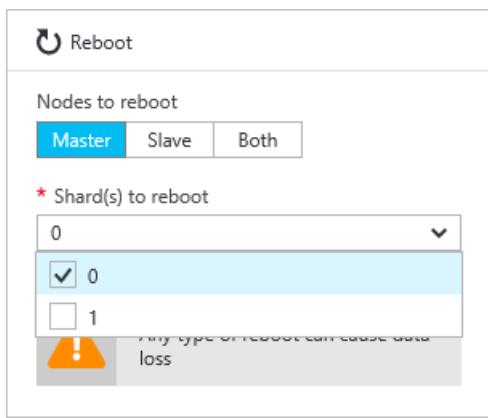
The **Reboot** blade allows you to reboot one or more nodes of your cache. This reboot capability enables you to test your application for resiliency if there is a failure of a cache node.



Select the nodes to reboot and click **Reboot**.



If you have a premium cache with clustering enabled, you can select which shards of the cache to reboot.



To reboot one or more nodes of your cache, select the desired nodes and click **Reboot**. If you have a premium cache with clustering enabled, select the desired shards to reboot and then click **Reboot**. After a few minutes, the selected nodes reboot, and are back online a few minutes later.

The impact on client applications varies depending on which nodes that you reboot.

- **Master** - When the master node is rebooted, Azure Cache for Redis fails over to the replica node and promotes it to master. During this failover, there may be a short interval in which connections may fail to the cache.
- **Slave** - When the slave node is rebooted, there is typically no impact to cache clients.
- **Both master and slave** - When both cache nodes are rebooted, all data is lost in the cache and connections to the cache fail until the primary node comes back online. If you have configured [data persistence](#), the most recent backup is restored when the cache comes back online, but any cache writes that occurred after the most recent backup are lost.
- **Nodes of a premium cache with clustering enabled** - When you reboot one or more nodes of a premium cache with clustering enabled, the behavior for the selected nodes is the same as when you reboot the corresponding node or nodes of a non-clustered cache.

Reboot FAQ

- [Which node should I reboot to test my application?](#)
- [Can I reboot the cache to clear client connections?](#)
- [Will I lose data from my cache if I do a reboot?](#)
- [Can I reboot my cache using PowerShell, CLI, or other management tools?](#)

Which node should I reboot to test my application?

To test the resiliency of your application against failure of the primary node of your cache, reboot the **Master** node. To test the resiliency of your application against failure of the secondary node, reboot the **Slave** node. To test the resiliency of your application against total failure of the cache, reboot **Both** nodes.

Can I reboot the cache to clear client connections?

Yes, if you reboot the cache all client connections are cleared. Rebooting can be useful in the case where all client connections are used up due to a logic error or a bug in the client application. Each pricing tier has different [client connection limits](#) for the various sizes, and once these limits are reached, no more client connections are accepted. Rebooting the cache provides a way to clear all client connections.

IMPORTANT

If you reboot your cache to clear client connections, StackExchange.Redis automatically reconnects once the Redis node is back online. If the underlying issue is not resolved, the client connections may continue to be used up.

Will I lose data from my cache if I do a reboot?

If you reboot both the **Master** and **Slave** nodes, all data in the cache (or in that shard if you are using a premium

cache with clustering enabled) may be lost, but this is not guaranteed either. If you have configured [data persistence](#), the most recent backup will be restored when the cache comes back online, but any cache writes that have occurred after the backup was made are lost.

If you reboot just one of the nodes, data is not typically lost, but it still may be. For example if the master node is rebooted and a cache write is in progress, the data from the cache write is lost. Another scenario for data loss would be if you reboot one node and the other node happens to go down due to a failure at the same time. For more information about possible causes for data loss, see [What happened to my data in Redis?](#)

Can I reboot my cache using PowerShell, CLI, or other management tools?

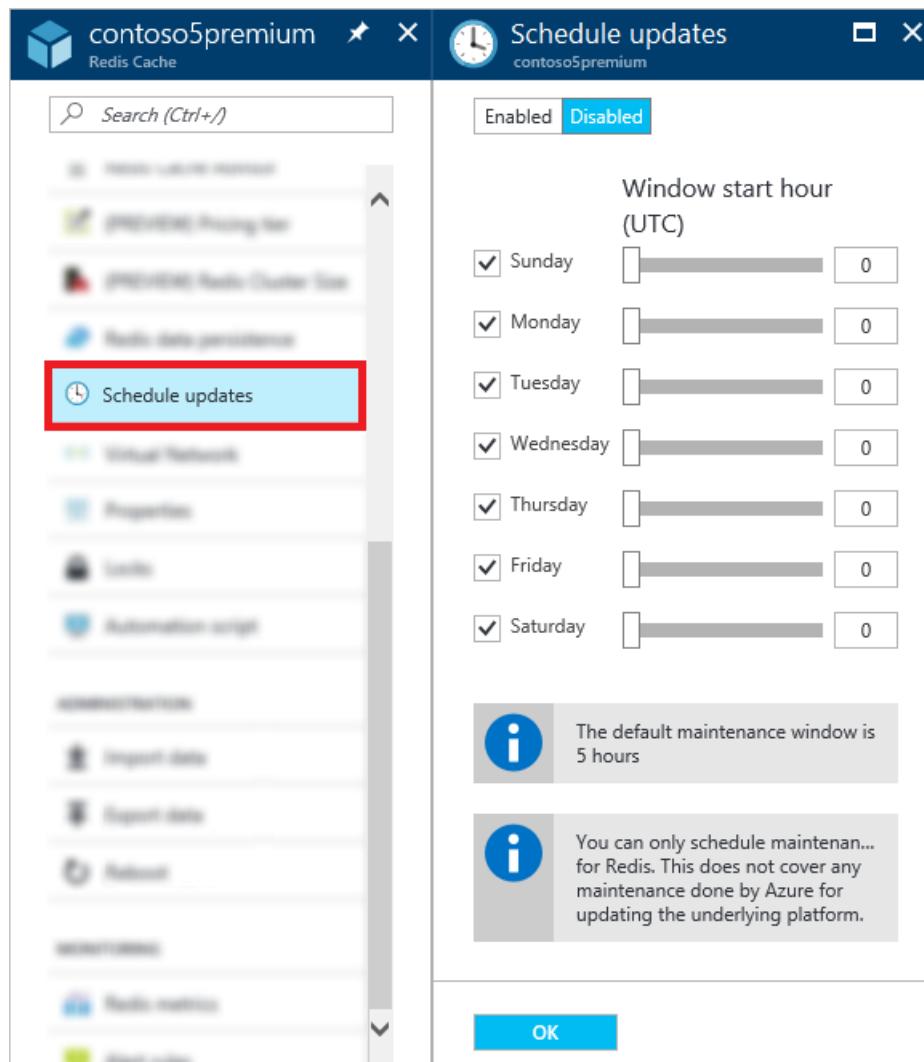
Yes, for PowerShell instructions see [To reboot an Azure Cache for Redis](#).

Schedule updates

The **Schedule updates** blade allows you to designate a maintenance window for your cache instance. When the maintenance window is specified, any Redis server updates are made during this window.

NOTE

The maintenance window applies only to Redis server updates, and not to any Azure updates or updates to the operating system of the VMs that host the cache.



To specify a maintenance window, check the desired days and specify the maintenance window start hour for each day, and click **OK**. Note that the maintenance window time is in UTC.

The default, and minimum, maintenance window for updates is five hours. This value is not configurable from the

Azure portal, but you can configure it in PowerShell using the `MaintenanceWindow` parameter of the `New-AzRedisCacheScheduleEntry` cmdlet. For more information, see [Can I manage scheduled updates using PowerShell, CLI, or other management tools?](#)

Schedule updates FAQ

- [When do updates occur if I don't use the schedule updates feature?](#)
- [What type of updates are made during the scheduled maintenance window?](#)
- [Can I manage scheduled updates using PowerShell, CLI, or other management tools?](#)

When do updates occur if I don't use the schedule updates feature?

If you don't specify a maintenance window, updates can be made at any time.

What type of updates are made during the scheduled maintenance window?

Only Redis server updates are made during the scheduled maintenance window. The maintenance window does not apply to Azure updates or updates to the VM operating system.

Can I managed scheduled updates using PowerShell, CLI, or other management tools?

Yes, you can manage your scheduled updates using the following PowerShell cmdlets:

- [Get-AzRedisCachePatchSchedule](#)
- [New-AzRedisCachePatchSchedule](#)
- [New-AzRedisCacheScheduleEntry](#)
- [Remove-AzRedisCachePatchSchedule](#)

Next steps

- Explore more [Azure Cache for Redis premium tier](#) features.

How to administer Azure Cache for Redis

11/15/2019 • 5 minutes to read • [Edit Online](#)

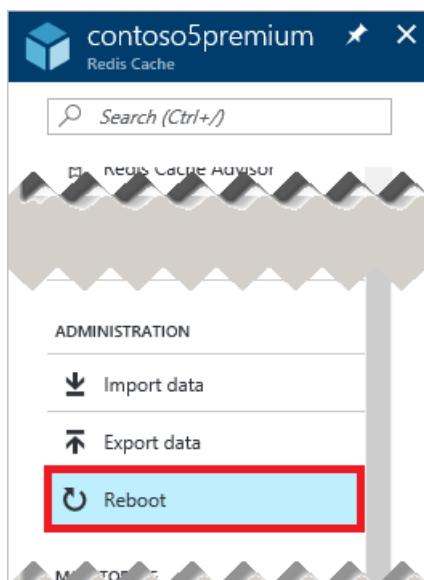
This topic describes how to perform administration tasks such as [rebooting](#) and [scheduling updates](#) for your Azure Cache for Redis instances.

NOTE

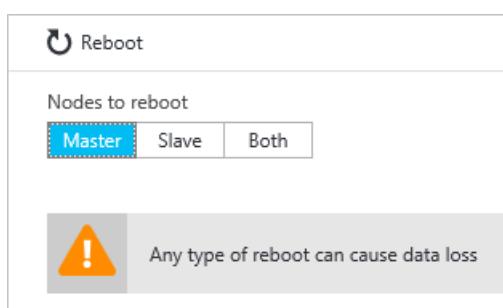
This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

Reboot

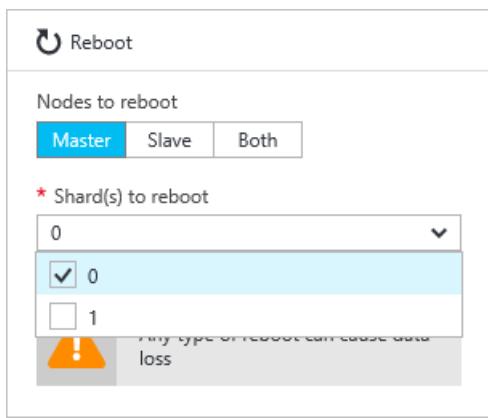
The **Reboot** blade allows you to reboot one or more nodes of your cache. This reboot capability enables you to test your application for resiliency if there is a failure of a cache node.



Select the nodes to reboot and click **Reboot**.



If you have a premium cache with clustering enabled, you can select which shards of the cache to reboot.



To reboot one or more nodes of your cache, select the desired nodes and click **Reboot**. If you have a premium cache with clustering enabled, select the desired shards to reboot and then click **Reboot**. After a few minutes, the selected nodes reboot, and are back online a few minutes later.

The impact on client applications varies depending on which nodes that you reboot.

- **Master** - When the master node is rebooted, Azure Cache for Redis fails over to the replica node and promotes it to master. During this failover, there may be a short interval in which connections may fail to the cache.
- **Slave** - When the slave node is rebooted, there is typically no impact to cache clients.
- **Both master and slave** - When both cache nodes are rebooted, all data is lost in the cache and connections to the cache fail until the primary node comes back online. If you have configured [data persistence](#), the most recent backup is restored when the cache comes back online, but any cache writes that occurred after the most recent backup are lost.
- **Nodes of a premium cache with clustering enabled** - When you reboot one or more nodes of a premium cache with clustering enabled, the behavior for the selected nodes is the same as when you reboot the corresponding node or nodes of a non-clustered cache.

Reboot FAQ

- [Which node should I reboot to test my application?](#)
- [Can I reboot the cache to clear client connections?](#)
- [Will I lose data from my cache if I do a reboot?](#)
- [Can I reboot my cache using PowerShell, CLI, or other management tools?](#)

Which node should I reboot to test my application?

To test the resiliency of your application against failure of the primary node of your cache, reboot the **Master** node. To test the resiliency of your application against failure of the secondary node, reboot the **Slave** node. To test the resiliency of your application against total failure of the cache, reboot **Both** nodes.

Can I reboot the cache to clear client connections?

Yes, if you reboot the cache all client connections are cleared. Rebooting can be useful in the case where all client connections are used up due to a logic error or a bug in the client application. Each pricing tier has different [client connection limits](#) for the various sizes, and once these limits are reached, no more client connections are accepted. Rebooting the cache provides a way to clear all client connections.

IMPORTANT

If you reboot your cache to clear client connections, StackExchange.Redis automatically reconnects once the Redis node is back online. If the underlying issue is not resolved, the client connections may continue to be used up.

Will I lose data from my cache if I do a reboot?

If you reboot both the **Master** and **Slave** nodes, all data in the cache (or in that shard if you are using a premium

cache with clustering enabled) may be lost, but this is not guaranteed either. If you have configured [data persistence](#), the most recent backup will be restored when the cache comes back online, but any cache writes that have occurred after the backup was made are lost.

If you reboot just one of the nodes, data is not typically lost, but it still may be. For example if the master node is rebooted and a cache write is in progress, the data from the cache write is lost. Another scenario for data loss would be if you reboot one node and the other node happens to go down due to a failure at the same time. For more information about possible causes for data loss, see [What happened to my data in Redis?](#)

Can I reboot my cache using PowerShell, CLI, or other management tools?

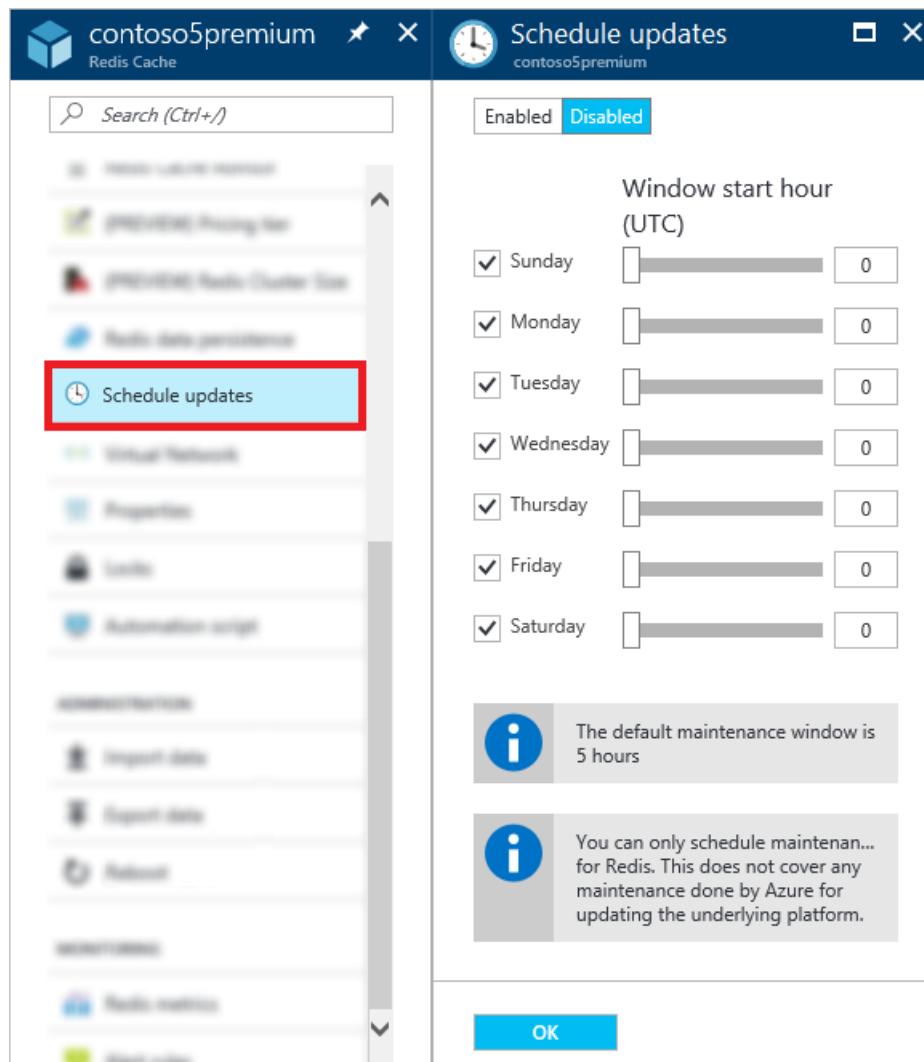
Yes, for PowerShell instructions see [To reboot an Azure Cache for Redis](#).

Schedule updates

The **Schedule updates** blade allows you to designate a maintenance window for your cache instance. When the maintenance window is specified, any Redis server updates are made during this window.

NOTE

The maintenance window applies only to Redis server updates, and not to any Azure updates or updates to the operating system of the VMs that host the cache.



To specify a maintenance window, check the desired days and specify the maintenance window start hour for each day, and click **OK**. Note that the maintenance window time is in UTC.

The default, and minimum, maintenance window for updates is five hours. This value is not configurable from the

Azure portal, but you can configure it in PowerShell using the `MaintenanceWindow` parameter of the `New-AzRedisCacheScheduleEntry` cmdlet. For more information, see [Can I manage scheduled updates using PowerShell, CLI, or other management tools?](#)

Schedule updates FAQ

- [When do updates occur if I don't use the schedule updates feature?](#)
- [What type of updates are made during the scheduled maintenance window?](#)
- [Can I manage scheduled updates using PowerShell, CLI, or other management tools?](#)

When do updates occur if I don't use the schedule updates feature?

If you don't specify a maintenance window, updates can be made at any time.

What type of updates are made during the scheduled maintenance window?

Only Redis server updates are made during the scheduled maintenance window. The maintenance window does not apply to Azure updates or updates to the VM operating system.

Can I managed scheduled updates using PowerShell, CLI, or other management tools?

Yes, you can manage your scheduled updates using the following PowerShell cmdlets:

- [Get-AzRedisCachePatchSchedule](#)
- [New-AzRedisCachePatchSchedule](#)
- [New-AzRedisCacheScheduleEntry](#)
- [Remove-AzRedisCachePatchSchedule](#)

Next steps

- Explore more [Azure Cache for Redis premium tier](#) features.

How to use the Redis command-line tool with Azure Cache for Redis

12/13/2019 • 2 minutes to read • [Edit Online](#)

redis-cli.exe is a popular command-line tool for interacting with an Azure Cache for Redis as a client. This tool is also available for use with Azure Cache for Redis.

The tool is available for Windows platforms by downloading the [Redis command-line tools for Windows](#).

If you want to run the command-line tool on another platform, download Azure Cache for Redis from <https://redis.io/download>.

Gather cache access information

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

You can gather the information needed to access the cache using three methods:

1. Azure CLI using [az redis list-keys](#)
2. Azure PowerShell using [Get-AzRedisCacheKey](#)
3. Using the Azure portal.

In this section, you will retrieve the keys from the Azure portal.

Retrieve host name, ports, and access keys from the Azure portal

To connect to an Azure Cache for Redis instance, cache clients need the host name, ports, and a key for the cache. Some clients might refer to these items by slightly different names. You can get the host name, ports, and keys from the [Azure portal](#).

- To get the access keys, from your cache left navigation, select **Access keys**.

The screenshot shows the 'Access keys' section of the Azure Cache for Redis properties. It displays two connection strings: Primary (d2Y2ba56GvC2kEM3NmOrKuctu+emE1A1A1A1A1A1A=) and Secondary (5MLIRS9d56Lgt+61AjWU9r6KJsFwS6691A1A1A1A1A=). A 'Copy to clipboard' button is visible next to each string.

- To get the host name and ports, from your cache left navigation, select **Properties**. The host name is of the form <DNS name>.redis.cache.windows.net.

The screenshot shows the 'Properties' section of the Azure Cache for Redis properties. It displays the host name (contoso-tmp.redis.cache.windows.net), non-SSL port (Disabled), and SSL port (6380). A 'Copy to clipboard' button is visible next to the host name.

Enable access for redis-cli.exe

With Azure Cache for Redis, only the SSL port (6380) is enabled by default. The `redis-cli.exe` command-line tool doesn't support SSL. You have two configuration choices to use it:

1. [Enable the non-SSL port \(6379\)](#) - **This configuration is not recommended** because in this configuration, the access keys are sent via TCP in clear text. This change can compromise access to your cache. The only scenario where you might consider this configuration is when you are just accessing a test cache.
2. Download and install [stunnel](#).

Run **stunnel GUI Start** to start the server.

Right-click the taskbar icon for the stunnel server and click **Show Log Window**.

On the stunnel Log Window menu, click **Configuration > Edit Configuration** to open the current

configuration file.

Add the following entry for `redis-cli.exe` under the **Service definitions** section. Insert your actual cache name in place of `yourcachename`.

```
[redis-cli]
client = yes
accept = 127.0.0.1:6380
connect = yourcachename.redis.cache.windows.net:6380
```

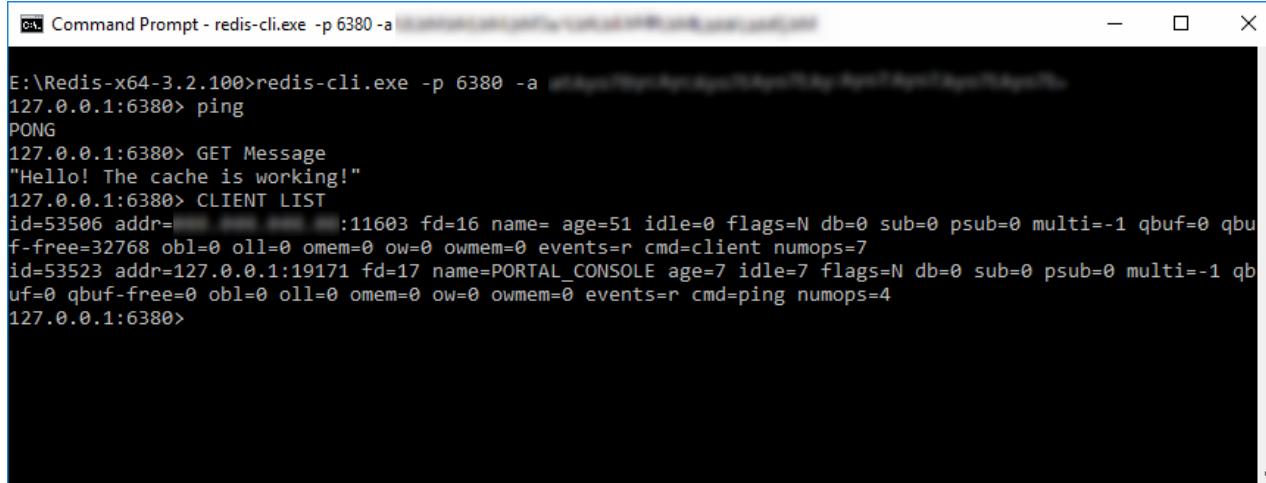
Save and close the configuration file.

On the stunnel Log Window menu, click **Configuration > Reload Configuration**.

Connect using the Redis command-line tool.

When using stunnel, run `redis-cli.exe`, and pass only your *port*, and *access key* (primary or secondary) to connect to the cache.

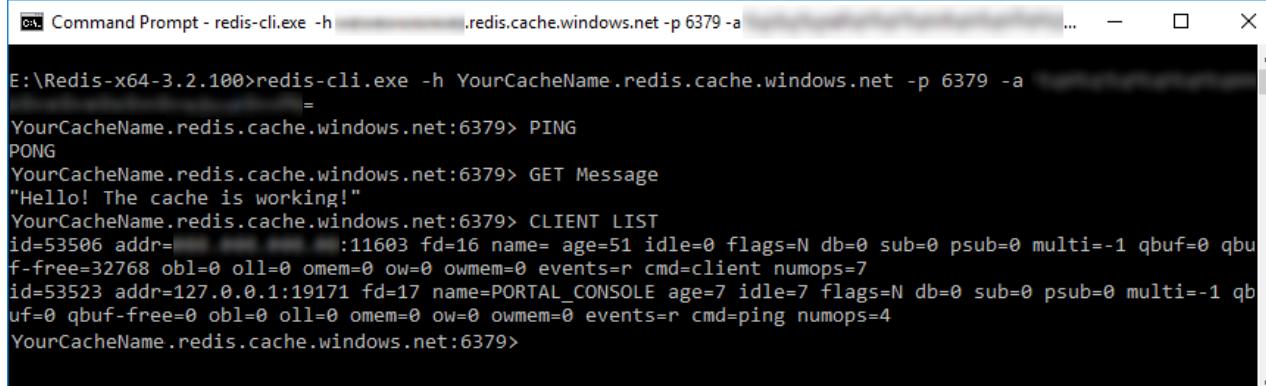
```
redis-cli.exe -p 6380 -a YourAccessKey
```



```
E:\Redis-x64-3.2.100>redis-cli.exe -p 6380 -a
127.0.0.1:6380> ping
PONG
127.0.0.1:6380> GET Message
"Hello! The cache is working!"
127.0.0.1:6380> CLIENT LIST
id=53506 addr=... port=6380:11603 fd=16 name= age=51 idle=0 flags=N db=0 sub=0 psub=0 multi=-1 qbuf=0 qbu
f-free=32768 obl=0 oll=0 omem=0 ow=0 owmem=0 events=r cmd=client numops=7
id=53523 addr=127.0.0.1:19171 fd=17 name=PORTAL_CONSOLE age=7 idle=7 flags=N db=0 sub=0 psub=0 multi=-1 qb
uf=0 qbuf-free=0 obl=0 oll=0 omem=0 ow=0 owmem=0 events=r cmd=ping numops=4
127.0.0.1:6380>
```

If you're using a test cache with the **unsecure** non-SSL port, run `redis-cli.exe` and pass your *host name*, *port*, and *access key* (primary or secondary) to connect to the test cache.

```
redis-cli.exe -h yourcachename.redis.cache.windows.net -p 6379 -a YourAccessKey
```



```
E:\Redis-x64-3.2.100>redis-cli.exe -h YourCacheName.redis.cache.windows.net -p 6379 -a
YourCacheName.redis.cache.windows.net:6379> PING
PONG
YourCacheName.redis.cache.windows.net:6379> GET Message
"Hello! The cache is working!"
YourCacheName.redis.cache.windows.net:6379> CLIENT LIST
id=53506 addr=... port=6379:11603 fd=16 name= age=51 idle=0 flags=N db=0 sub=0 psub=0 multi=-1 qbuf=0 qbu
f-free=32768 obl=0 oll=0 omem=0 ow=0 owmem=0 events=r cmd=client numops=7
id=53523 addr=127.0.0.1:19171 fd=17 name=PORTAL_CONSOLE age=7 idle=7 flags=N db=0 sub=0 psub=0 multi=-1 qb
uf=0 qbuf-free=0 obl=0 oll=0 omem=0 ow=0 owmem=0 events=r cmd=ping numops=4
YourCacheName.redis.cache.windows.net:6379>
```

Next steps

Learn more about using the [Redis Console](#) to issue commands.

Remove TLS 1.0 and 1.1 from use with Azure Cache for Redis

1/17/2020 • 3 minutes to read • [Edit Online](#)

There's an industry-wide push toward the exclusive use of Transport Layer Security (TLS) version 1.2 or later. TLS versions 1.0 and 1.1 are known to be susceptible to attacks such as BEAST and POODLE, and to have other Common Vulnerabilities and Exposures (CVE) weaknesses. They also don't support the modern encryption methods and cipher suites recommended by Payment Card Industry (PCI) compliance standards. This [TLS security blog](#) explains some of these vulnerabilities in more detail.

As a part of this effort, we'll be making the following changes to Azure Cache for Redis:

- **Phase 1:** We'll configure the default minimum TLS version to be 1.2 for newly created cache instances. Existing cache instances won't be updated at this point. You'll be allowed to [change the minimum TLS version](#) back to 1.0 or 1.1 for backward compatibility, if needed. This change can be done through the Azure portal or other management APIs.
- **Phase 2:** We'll stop supporting TLS versions 1.0 and 1.1. After this change, your application will be required to use TLS 1.2 or later to communicate with your cache.

Additionally, as a part of this change, we'll be removing support for older, insecure cypher suites. Our supported cypher suites will be restricted to the following when the cache is configured with a minimum TLS version of 1.2.

- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384_P384
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256_P256

This article provides general guidance about how to detect dependencies on these earlier TLS versions and remove them from your application.

The dates when these changes take effect are:

CLOUD	PHASE 1 START DATE	PHASE 2 START DATE
Azure (global)	January 13, 2020	March 31, 2020
Azure Government	March 13, 2020	May 11, 2020
Azure Germany	March 13, 2020	May 11, 2020
Azure China	March 13, 2020	May 11, 2020

Check whether your application is already compliant

The easiest way to find out whether your application will work with TLS 1.2 is to set the **Minimum TLS version** value to TLS 1.2 on a test or staging cache that it uses. The **Minimum TLS version** setting is in the [Advanced settings](#) of your cache instance in the Azure portal. If the application continues to function as expected after this change, it's probably compliant. You might need to configure some Redis client libraries used by your application specifically to enable TLS 1.2, so they can connect to Azure Cache for Redis over that security protocol.

Configure your application to use TLS 1.2

Most applications use Redis client libraries to handle communication with their caches. Here are instructions for configuring some of the popular client libraries, in various programming languages and frameworks, to use TLS 1.2.

.NET Framework

Redis .NET clients use the earliest TLS version by default on .NET Framework 4.5.2 or earlier, and use the latest TLS version on .NET Framework 4.6 or later. If you're using an older version of .NET Framework, you can enable TLS 1.2 manually:

- **StackExchange.Redis:** Set `ssl=true` and `sslprotocols=tls12` in the connection string.
- **ServiceStack.Redis:** Follow the [ServiceStack.Redis instructions](#).

.NET Core

Redis .NET Core clients use the latest TLS version by default.

Java

Redis Java clients use TLS 1.0 on Java version 6 or earlier. Jedis, Lettuce, and Redisson can't connect to Azure Cache for Redis if TLS 1.0 is disabled on the cache. Upgrade your Java framework to use new TLS versions.

For Java 7, Redis clients don't use TLS 1.2 by default but can be configured for it. Jedis allows you to specify the underlying TLS settings with the following code snippet:

```
SSLContext sslSocketFactory = (SSLContext) SSLContext.getDefault();
SSLParameters sslParameters = new SSLParameters();
sslParameters.setEndpointIdentificationAlgorithm("HTTPS");
sslParameters.setProtocols(new String[]{"TLSv1.2"});

URI uri = URI.create("rediss://host:port");
JedisShardInfo shardInfo = new JedisShardInfo(uri, sslSocketFactory, sslParameters, null);

shardInfo.setPassword("cachePassword");

Jedis jedis = new Jedis(shardInfo);
```

The Lettuce and Redisson clients don't yet support specifying the TLS version, so they'll break if the cache accepts only TLS 1.2 connections. Fixes for these clients are being reviewed, so check with those packages for an updated version with this support.

In Java 8, TLS 1.2 is used by default and shouldn't require updates to your client configuration in most cases. To be safe, test your application.

Node.js

Node Redis and IORedis use TLS 1.2 by default.

PHP

Predis on PHP 7 won't work because PHP 7 supports only TLS 1.0. On PHP 7.2.1 or earlier, Predis uses TLS 1.0 or 1.1 by default. You can specify TLS 1.2 when you create the client instance:

```
$redis=newPredis\Client([
    'scheme'=>'tls',
    'host'=>'host',
    'port'=>6380,
    'password'=>'password',
    'ssl'=>[
        'crypto_type'=>STREAM_CRYPTO_METHOD_TLSv1_2_CLIENT,
    ],
]);
```

On PHP 7.3 or above, Predis uses the latest TLS version.

PhpRedis doesn't support TLS on any PHP version.

Python

Redis-py uses TLS 1.2 by default.

GO

Redigo uses TLS 1.2 by default.

Additional information

- [How to configure Azure Cache for Redis](#)

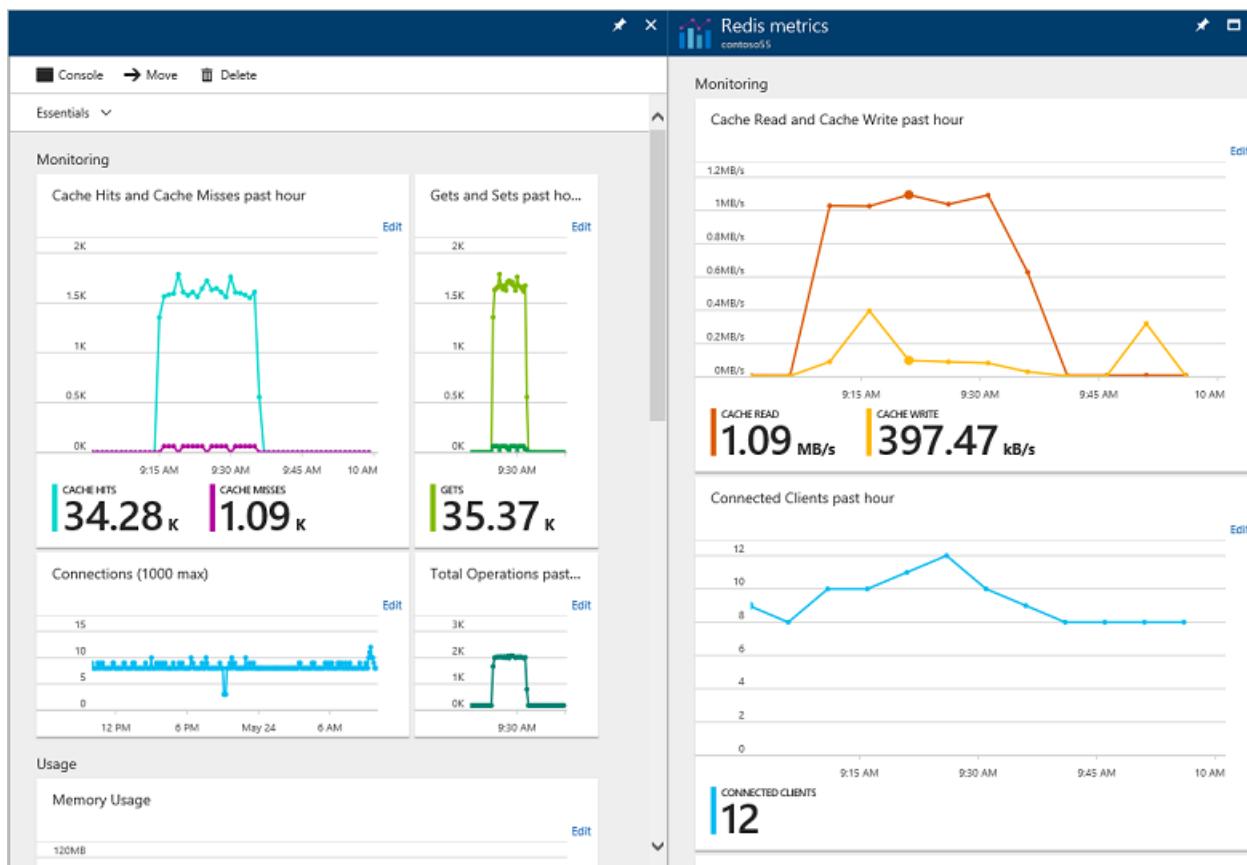
How to monitor Azure Cache for Redis

1/23/2020 • 9 minutes to read • [Edit Online](#)

Azure Cache for Redis uses [Azure Monitor](#) to provide several options for monitoring your cache instances. You can view metrics, pin metrics charts to the Startboard, customize the date and time range of monitoring charts, add and remove metrics from the charts, and set alerts when certain conditions are met. These tools enable you to monitor the health of your Azure Cache for Redis instances and help you manage your caching applications.

Metrics for Azure Cache for Redis instances are collected using the Redis [INFO](#) command approximately twice per minute and automatically stored for 30 days (see [Export cache metrics](#) to configure a different retention policy) so they can be displayed in the metrics charts and evaluated by alert rules. For more information about the different [INFO](#) values used for each cache metric, see [Available metrics and reporting intervals](#).

To view cache metrics, [browse](#) to your cache instance in the [Azure portal](#). Azure Cache for Redis provides some built-in charts on the **Overview** blade and the **Redis metrics** blade. Each chart can be customized by adding or removing metrics and changing the reporting interval.



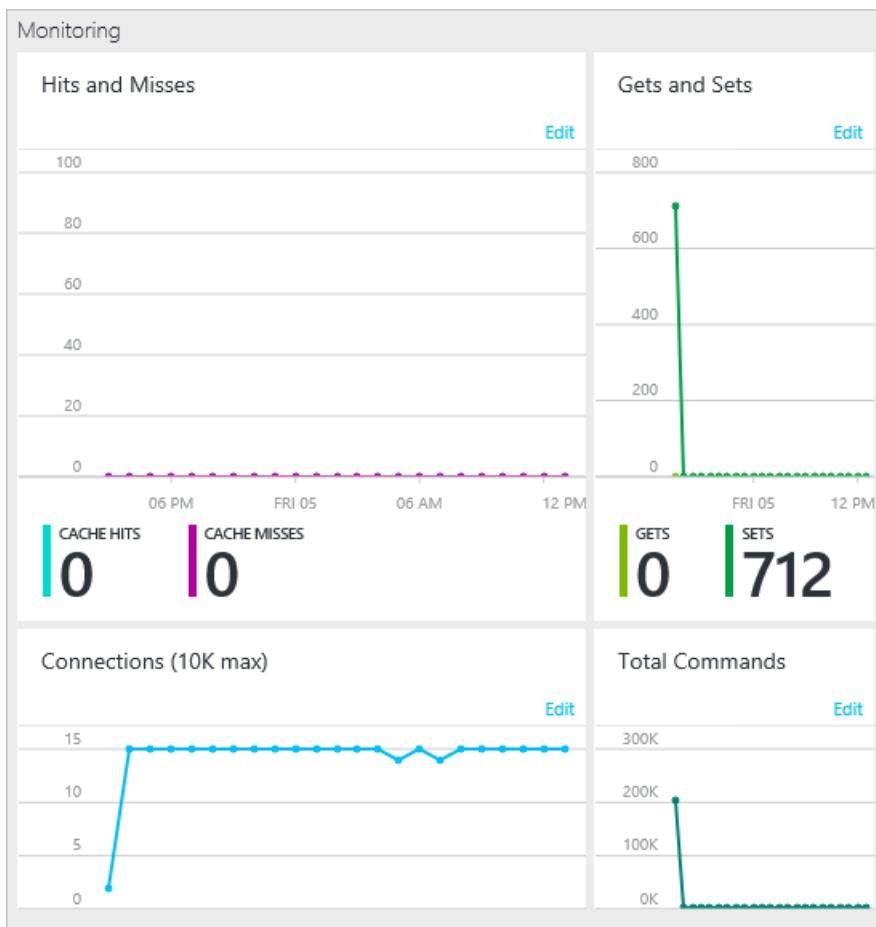
View pre-configured metrics charts

The **Overview** blade has the following pre-configured monitoring charts.

- [Monitoring charts](#)
- [Usage charts](#)

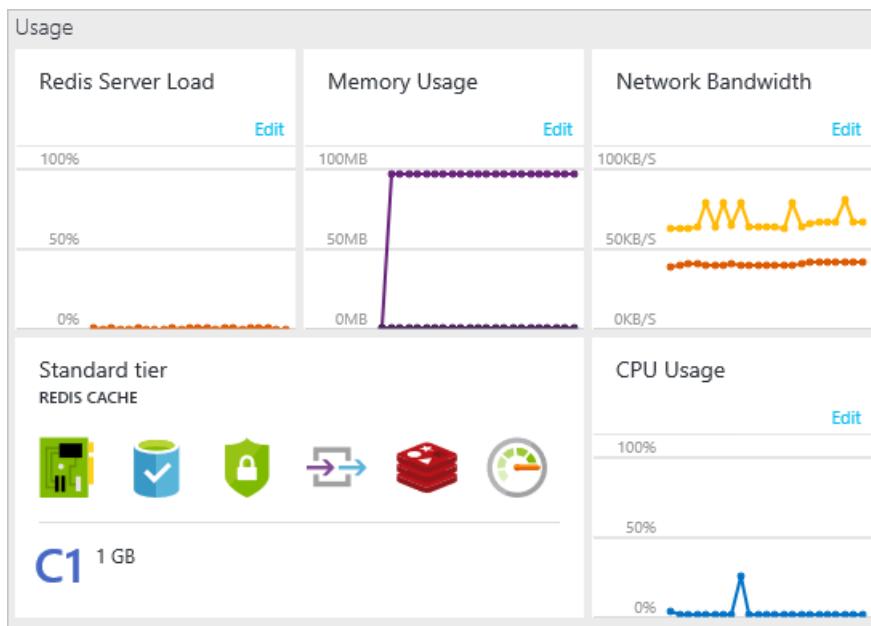
Monitoring charts

The **Monitoring** section in the **Overview** blade has **Hits and Misses**, **Gets and Sets**, **Connections**, and **Total Commands** charts.



Usage charts

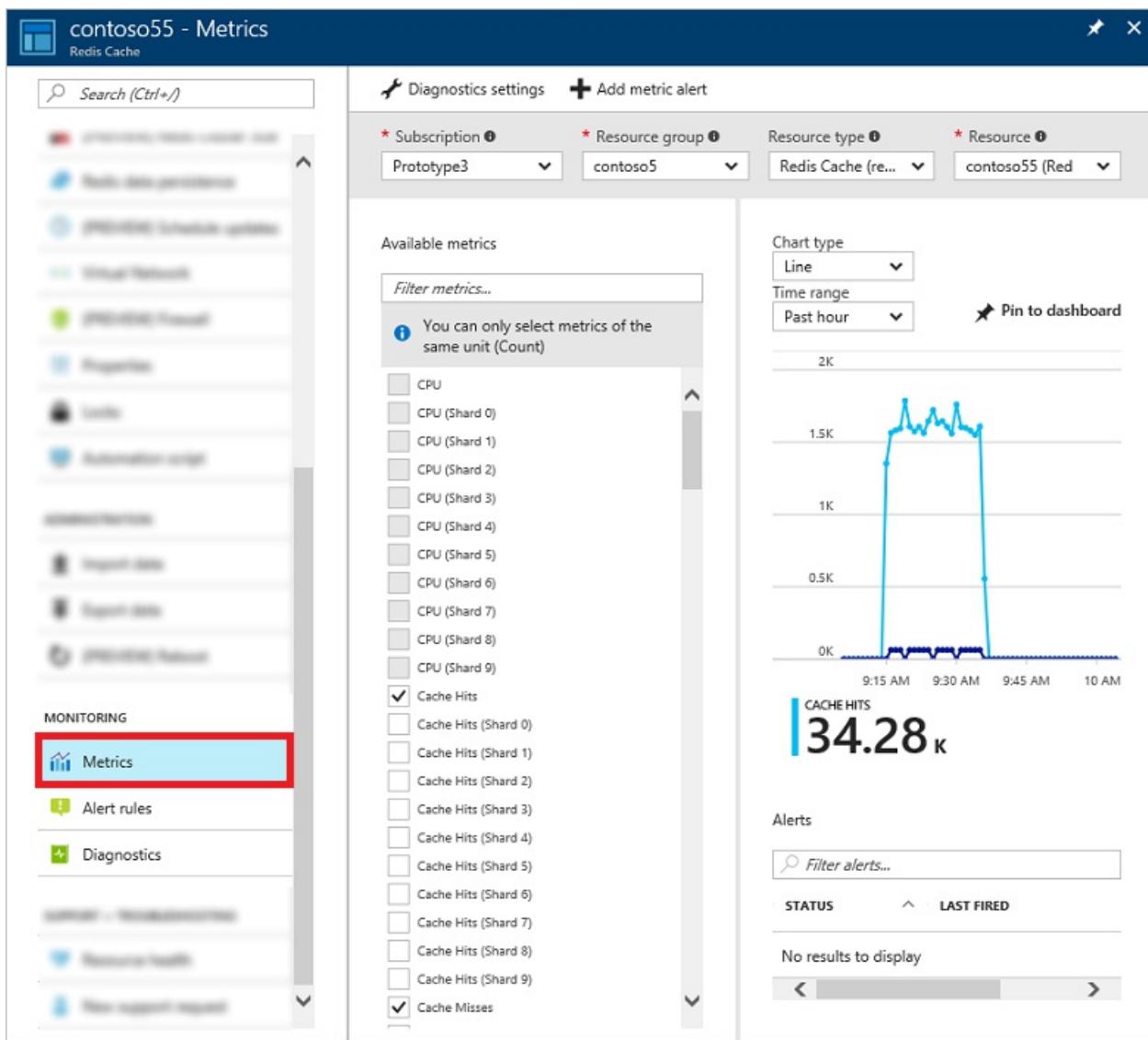
The **Usage** section in the **Overview** blade has **Redis Server Load**, **Memory Usage**, **Network Bandwidth**, and **CPU Usage** charts, and also displays the **Pricing tier** for the cache instance.



The **Pricing tier** displays the cache pricing tier, and can be used to [scale](#) the cache to a different pricing tier.

View metrics with Azure monitor

To view Redis metrics and create custom charts using Azure Monitor, click **Metrics** from the **Resource menu**, and customize your chart using the desired metrics, reporting interval, chart type, and more.



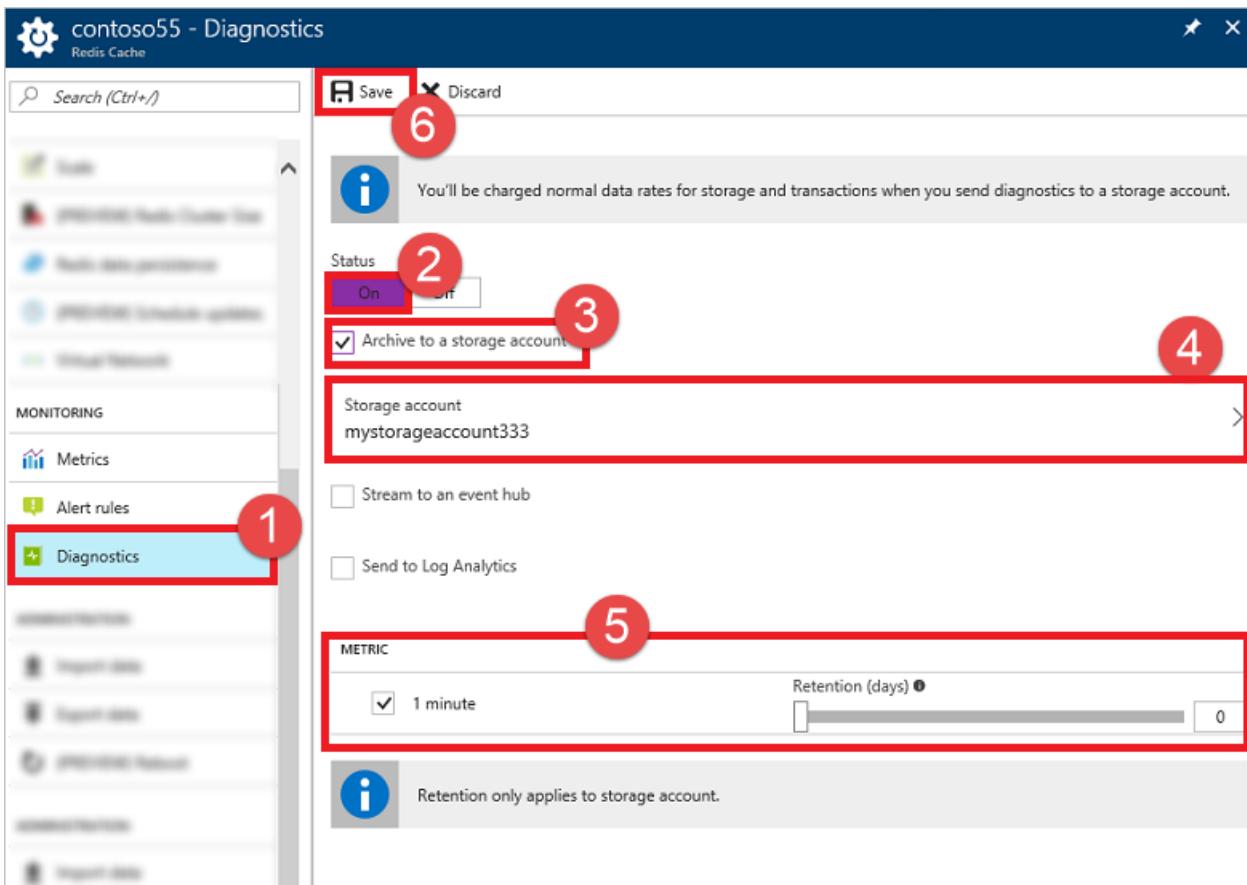
For more information on working with metrics using Azure Monitor, see [Overview of metrics in Microsoft Azure](#).

Export cache metrics

By default, cache metrics in Azure Monitor are [stored for 30 days](#) and then deleted. To persist your cache metrics for longer than 30 days, you can [designate a storage account](#) and specify a **Retention (days)** policy for your cache metrics.

To configure a storage account for your cache metrics:

1. In the **Azure Cache for Redis** page, under the **Monitoring** heading, select **Diagnostics**.
2. Select **+ Add diagnostic setting**.
3. Name the settings.
4. Check **Archive to a storage account**. You'll be charged normal data rates for storage and transactions when you send diagnostics to a storage account.
5. Select **Configure** to choose the storage account in which to store the cache metrics.
6. Under the table heading **metric**, check box beside the line items you want to store, such as **AllMetrics**. Specify a **Retention (days)** policy. The maximum days retention you can specify is **365 days**. However, if you want to retain the metrics data forever, set **Retention (days)** to **0**.
7. Click **Save**.



NOTE

In addition to archiving your cache metrics to storage, you can also [stream them to an Event hub](#) or [send them to Azure Monitor logs](#).

To access your metrics, you can view them in the Azure portal as previously described in this article, and you can also access them using the [Azure Monitor Metrics REST API](#).

NOTE

If you change storage accounts, the data in the previously configured storage account remains available for download, but it is not displayed in the Azure portal.

Available metrics and reporting intervals

Cache metrics are reported using several reporting intervals, including **Past hour**, **Today**, **Past week**, and **Custom**. The **Metric** blade for each metrics chart displays the average, minimum, and maximum values for each metric in the chart, and some metrics display a total for the reporting interval.

Each metric includes two versions. One metric measures performance for the entire cache, and for caches that use [clustering](#), a second version of the metric that includes `(Shard 0-9)` in the name measures performance for a single shard in a cache. For example if a cache has four shards, `Cache Hits` is the total number of hits for the entire cache, and `Cache Hits (Shard 3)` is just the hits for that shard of the cache.

NOTE

Even when the cache is idle with no connected active client applications, you may see some cache activity, such as connected clients, memory usage, and operations being performed. This activity is normal during the operation of an Azure Cache for Redis instance.

METRIC	DESCRIPTION
Cache Hits	The number of successful key lookups during the specified reporting interval. This number maps to <code>keyspace_hits</code> from the Redis INFO command.
Cache Latency (Preview)	The latency of the cache calculated based off the internode latency of the cache. This metric is measured in microseconds, and has three dimensions: <code>Avg</code> , <code>Min</code> , and <code>Max</code> , which represent the average, minimum, and maximum latency of the cache respectively during the specified reporting interval.
Cache Misses	The number of failed key lookups during the specified reporting interval. This number maps to <code>keyspace_misses</code> from the Redis INFO command. Cache misses do not necessarily mean there is an issue with the cache. For example, when using the cache-aside programming pattern, an application looks first in the cache for an item. If the item is not there (cache miss), the item is retrieved from the database and added to the cache for next time. Cache misses are normal behavior for the cache-aside programming pattern. If the number of cache misses is higher than expected, examine the application logic that populates and reads from the cache. If items are being evicted from the cache due to memory pressure, then there may be some cache misses, but a better metric to monitor for memory pressure would be <code>Used Memory</code> or <code>Evicted Keys</code> .
Cache Read	The amount of data read from the cache in Megabytes per second (MB/s) during the specified reporting interval. This value is derived from the network interface cards that support the virtual machine that hosts the cache and is not Redis specific. This value corresponds to the network bandwidth used by this cache. If you want to set up alerts for server-side network bandwidth limits, then create it using this <code>Cache Read</code> counter. See this table for the observed bandwidth limits for various cache pricing tiers and sizes.
Cache Write	The amount of data written to the cache in Megabytes per second (MB/s) during the specified reporting interval. This value is derived from the network interface cards that support the virtual machine that hosts the cache and is not Redis specific. This value corresponds to the network bandwidth of data sent to the cache from the client.
Connected Clients	The number of client connections to the cache during the specified reporting interval. This number maps to <code>connected_clients</code> from the Redis INFO command. Once the <code>connection limit</code> is reached, subsequent connection attempts to the cache will fail. Even if there are no active client applications, there may still be a few instances of connected clients due to internal processes and connections.

METRIC	DESCRIPTION
CPU	The CPU utilization of the Azure Cache for Redis server as a percentage during the specified reporting interval. This value maps to the operating system <code>\Processor(_Total)\% Processor Time</code> performance counter.
Errors	Specific failures and performance issues that the cache could be experiencing during a specified reporting interval. This metric has eight dimensions representing different error types, but could have more added in the future. The error types represented now are as follows: <ul style="list-style-type: none"> • Failover – when a cache fails over (subordinate promotes to master) • Dataloss – when there is data loss on the cache • UnresponsiveClients – when the clients are not reading data from the server fast enough • AOF – when there is an issue related to AOF persistence • RDB – when there is an issue related to RDB persistence • Import – when there is an issue related to Import RDB • Export – when there is an issue related to Export RDB
Evicted Keys	The number of items evicted from the cache during the specified reporting interval due to the <code>maxmemory</code> limit. This number maps to <code>evicted_keys</code> from the Redis INFO command.
Expired Keys	The number of items expired from the cache during the specified reporting interval. This value maps to <code>expired_keys</code> from the Redis INFO command.
Gets	The number of get operations from the cache during the specified reporting interval. This value is the sum of the following values from the Redis INFO all command: <code>cmdstat_get</code> , <code>cmdstat_hget</code> , <code>cmdstat_hgetall</code> , <code>cmdstat_hmget</code> , <code>cmdstat_mget</code> , <code>cmdstat_getbit</code> , and <code>cmdstat_getrange</code> , and is equivalent to the sum of cache hits and misses during the reporting interval.
Operations per Second	The total number of commands processed per second by the cache server during the specified reporting interval. This value maps to "instantaneous_ops_per_sec" from the Redis INFO command.
Redis Server Load	The percentage of cycles in which the Redis server is busy processing and not waiting idle for messages. If this counter reaches 100, it means the Redis server has hit a performance ceiling and the CPU can't process work any faster. If you are seeing high Redis Server Load, then you will see timeout exceptions in the client. In this case, you should consider scaling up or partitioning your data into multiple caches.

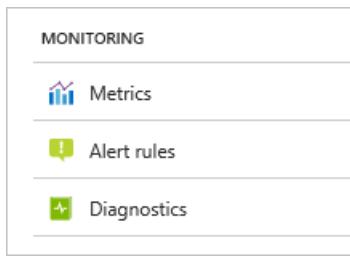
METRIC	DESCRIPTION
Sets	The number of set operations to the cache during the specified reporting interval. This value is the sum of the following values from the Redis INFO all command: <code>cmdstat_set</code> , <code>cmdstat_hset</code> , <code>cmdstat_hmset</code> , <code>cmdstat_hsetnx</code> , <code>cmdstat_lset</code> , <code>cmdstat_mset</code> , <code>cmdstat_msetnx</code> , <code>cmdstat_setbit</code> , <code>cmdstat_setex</code> , <code>cmdstat_setrange</code> , and <code>cmdstat_setnx</code> .
Total Keys	The maximum number of keys in the cache during the past reporting time period. This number maps to <code>keyspace</code> from the Redis INFO command. Due to a limitation of the underlying metrics system, for caches with clustering enabled, Total Keys returns the maximum number of keys of the shard that had the maximum number of keys during the reporting interval.
Total Operations	The total number of commands processed by the cache server during the specified reporting interval. This value maps to <code>total_commands_processed</code> from the Redis INFO command. When Azure Cache for Redis is used purely for pub/sub there will be no metrics for <code>Cache Hits</code> , <code>Cache Misses</code> , <code>Gets</code> , or <code>Sets</code> , but there will be <code>Total Operations</code> metrics that reflect the cache usage for pub/sub operations.
Used Memory	The amount of cache memory used for key/value pairs in the cache in MB during the specified reporting interval. This value maps to <code>used_memory</code> from the Redis INFO command. This value does not include metadata or fragmentation.
Used Memory Percentage	The % of total memory that is being used during the specified reporting interval. This value references the <code>used_memory</code> value from the Redis INFO command to calculate the percentage.
Used Memory RSS	The amount of cache memory used in MB during the specified reporting interval, including fragmentation and metadata. This value maps to <code>used_memory_rss</code> from the Redis INFO command.

Alerts

You can configure to receive alerts based on metrics and activity logs. Azure Monitor allows you to configure an alert to do the following when it triggers:

- Send an email notification
- Call a webhook
- Invoke an Azure Logic App

To configure Alert rules for your cache, click **Alert rules** from the **Resource menu**.



For more information about configuring and using Alerts, see [Overview of Alerts](#).

Activity Logs

Activity logs provide insight into the operations that were performed on your Azure Cache for Redis instances. It was previously known as "audit logs" or "operational logs". Using activity logs, you can determine the "what, who, and when" for any write operations (PUT, POST, DELETE) taken on your Azure Cache for Redis instances.

NOTE

Activity logs do not include read (GET) operations.

To view activity logs for your cache, click **Activity logs** from the **Resource menu**.

For more information about Activity logs, see [Overview of the Azure Activity Log](#).

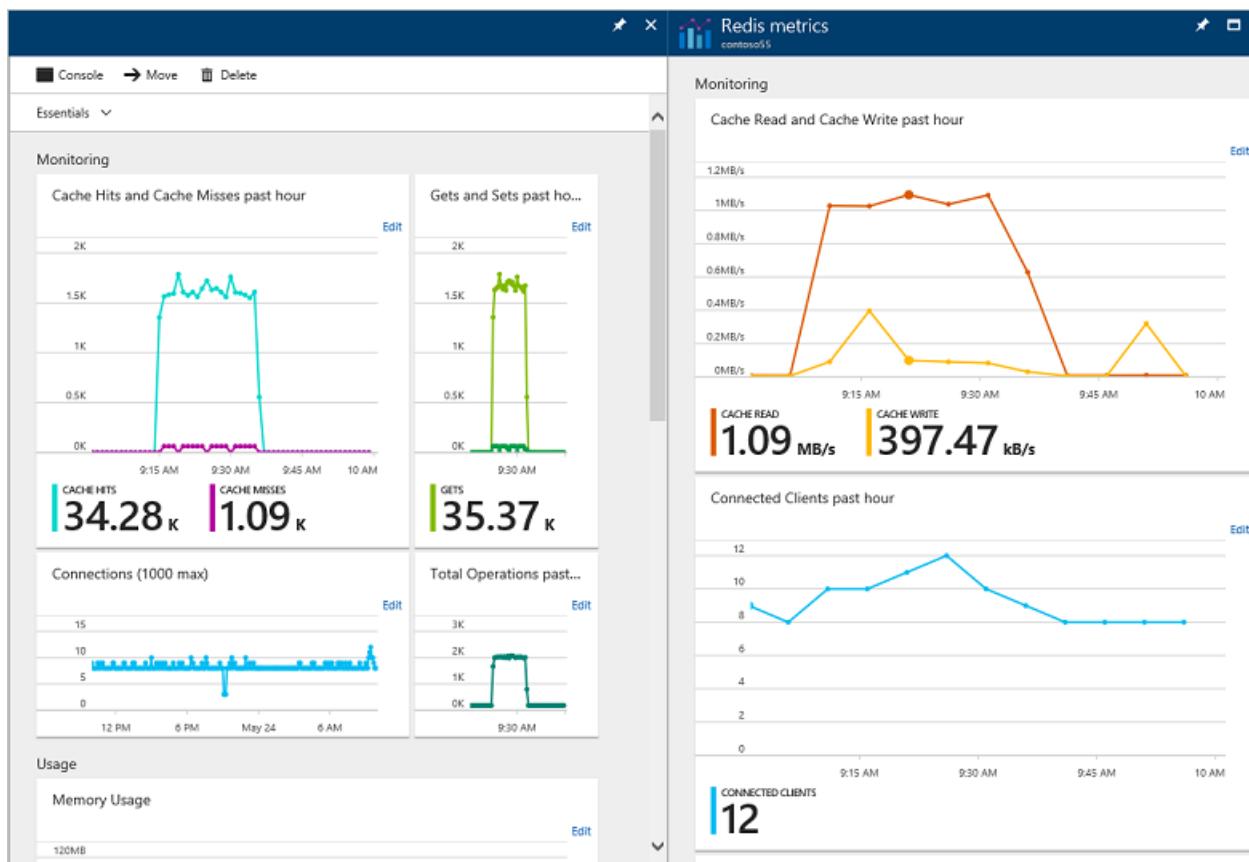
How to monitor Azure Cache for Redis

1/23/2020 • 9 minutes to read • [Edit Online](#)

Azure Cache for Redis uses [Azure Monitor](#) to provide several options for monitoring your cache instances. You can view metrics, pin metrics charts to the Startboard, customize the date and time range of monitoring charts, add and remove metrics from the charts, and set alerts when certain conditions are met. These tools enable you to monitor the health of your Azure Cache for Redis instances and help you manage your caching applications.

Metrics for Azure Cache for Redis instances are collected using the Redis [INFO](#) command approximately twice per minute and automatically stored for 30 days (see [Export cache metrics](#) to configure a different retention policy) so they can be displayed in the metrics charts and evaluated by alert rules. For more information about the different INFO values used for each cache metric, see [Available metrics and reporting intervals](#).

To view cache metrics, [browse](#) to your cache instance in the [Azure portal](#). Azure Cache for Redis provides some built-in charts on the **Overview** blade and the **Redis metrics** blade. Each chart can be customized by adding or removing metrics and changing the reporting interval.



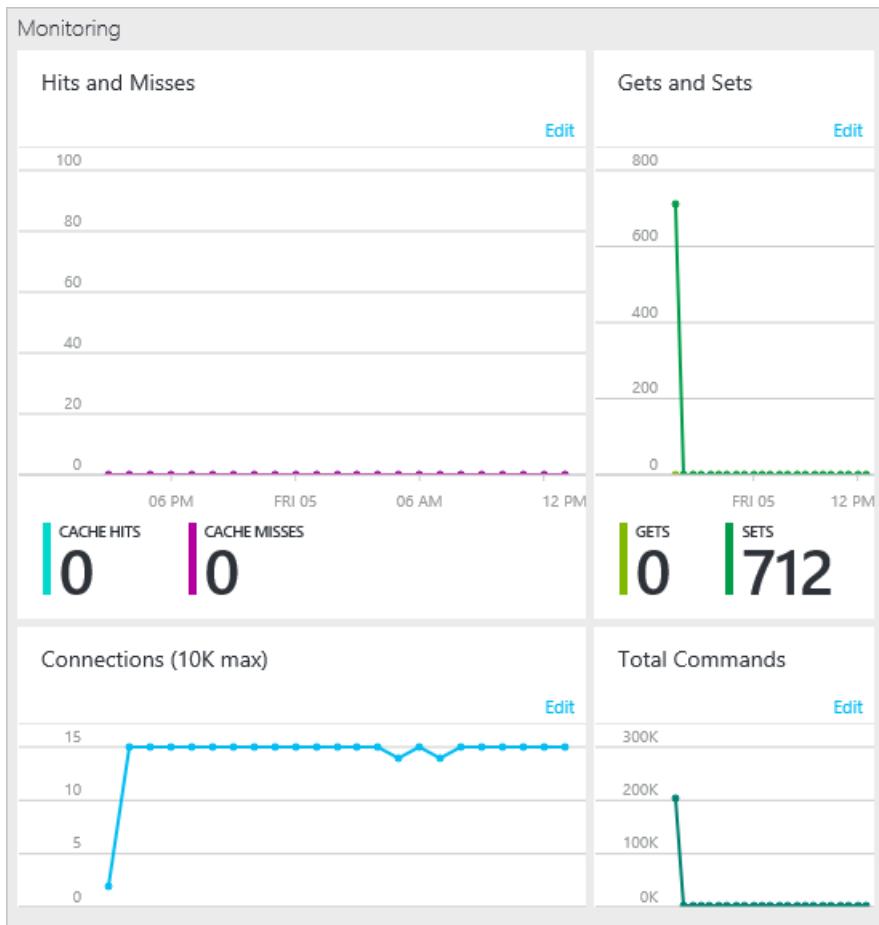
View pre-configured metrics charts

The **Overview** blade has the following pre-configured monitoring charts.

- [Monitoring charts](#)
- [Usage charts](#)

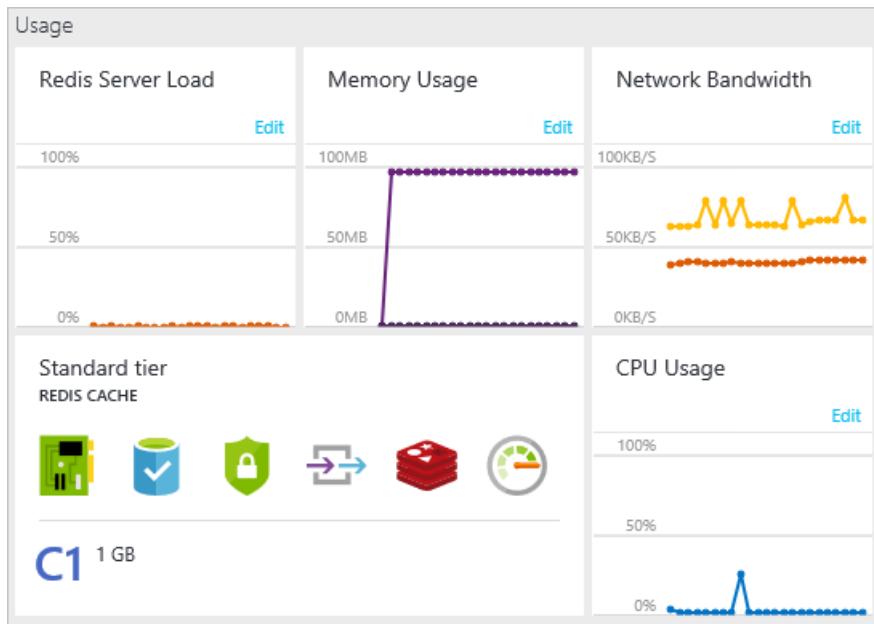
Monitoring charts

The **Monitoring** section in the **Overview** blade has **Hits and Misses**, **Gets and Sets**, **Connections**, and **Total Commands** charts.



Usage charts

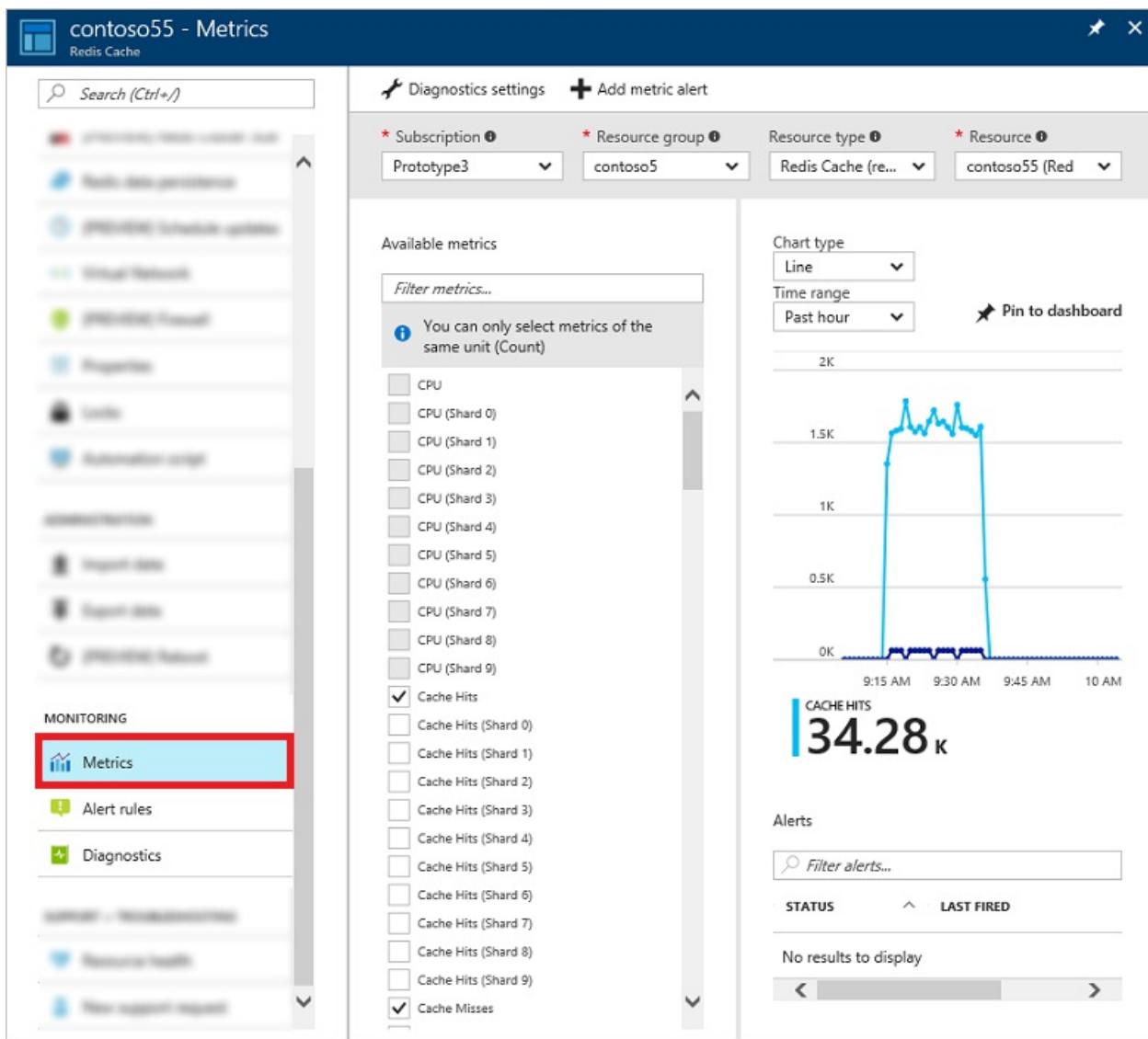
The **Usage** section in the **Overview** blade has **Redis Server Load**, **Memory Usage**, **Network Bandwidth**, and **CPU Usage** charts, and also displays the **Pricing tier** for the cache instance.



The **Pricing tier** displays the cache pricing tier, and can be used to [scale](#) the cache to a different pricing tier.

View metrics with Azure monitor

To view Redis metrics and create custom charts using Azure Monitor, click **Metrics** from the **Resource menu**, and customize your chart using the desired metrics, reporting interval, chart type, and more.



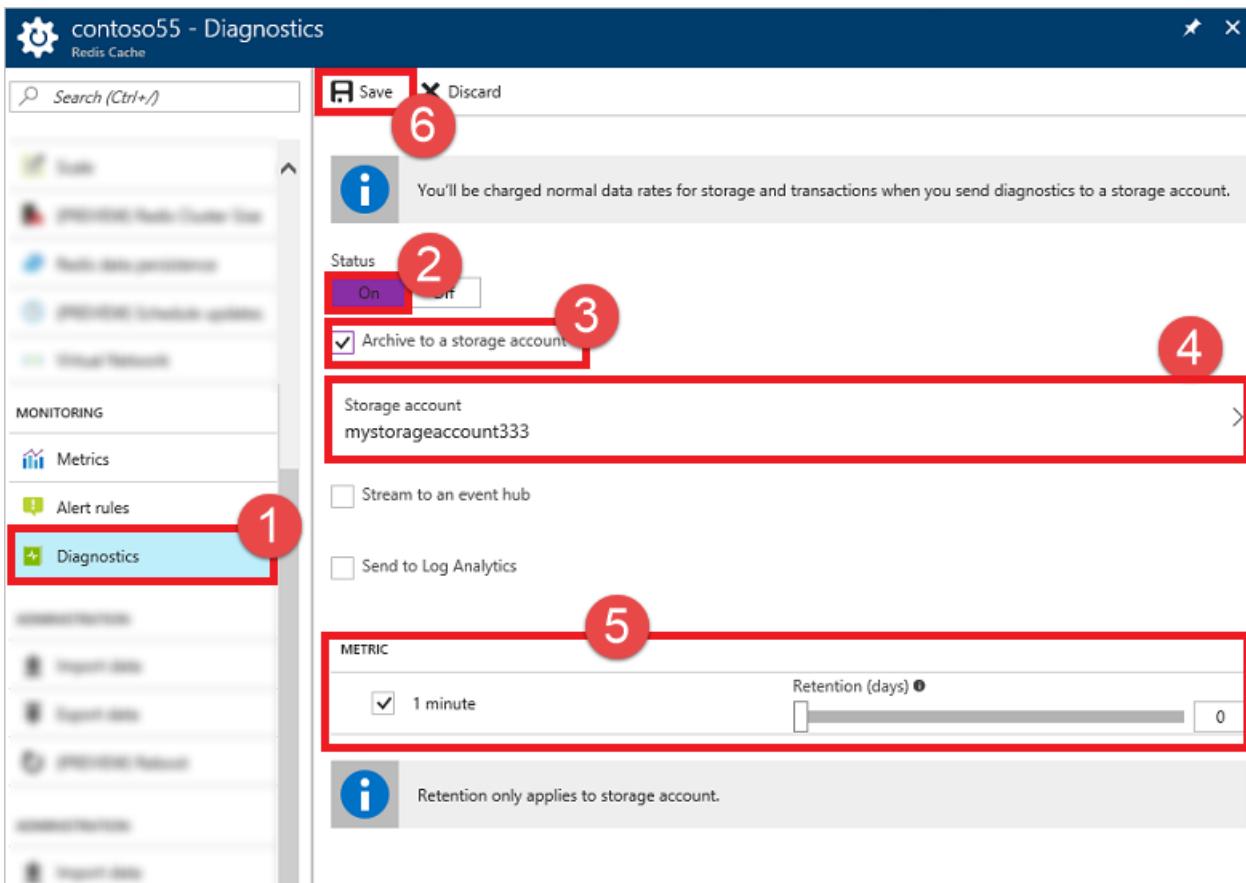
For more information on working with metrics using Azure Monitor, see [Overview of metrics in Microsoft Azure](#).

Export cache metrics

By default, cache metrics in Azure Monitor are [stored for 30 days](#) and then deleted. To persist your cache metrics for longer than 30 days, you can [designate a storage account](#) and specify a **Retention (days)** policy for your cache metrics.

To configure a storage account for your cache metrics:

1. In the **Azure Cache for Redis** page, under the **Monitoring** heading, select **Diagnostics**.
2. Select **+ Add diagnostic setting**.
3. Name the settings.
4. Check **Archive to a storage account**. You'll be charged normal data rates for storage and transactions when you send diagnostics to a storage account.
5. Select **Configure** to choose the storage account in which to store the cache metrics.
6. Under the table heading **metric**, check box beside the line items you want to store, such as **AllMetrics**. Specify a **Retention (days)** policy. The maximum days retention you can specify is **365 days**. However, if you want to retain the metrics data forever, set **Retention (days)** to **0**.
7. Click **Save**.



NOTE

In addition to archiving your cache metrics to storage, you can also [stream them to an Event hub](#) or [send them to Azure Monitor logs](#).

To access your metrics, you can view them in the Azure portal as previously described in this article, and you can also access them using the [Azure Monitor Metrics REST API](#).

NOTE

If you change storage accounts, the data in the previously configured storage account remains available for download, but it is not displayed in the Azure portal.

Available metrics and reporting intervals

Cache metrics are reported using several reporting intervals, including **Past hour**, **Today**, **Past week**, and **Custom**. The **Metric** blade for each metrics chart displays the average, minimum, and maximum values for each metric in the chart, and some metrics display a total for the reporting interval.

Each metric includes two versions. One metric measures performance for the entire cache, and for caches that use [clustering](#), a second version of the metric that includes `(Shard 0-9)` in the name measures performance for a single shard in a cache. For example if a cache has four shards, `Cache Hits` is the total number of hits for the entire cache, and `Cache Hits (Shard 3)` is just the hits for that shard of the cache.

NOTE

Even when the cache is idle with no connected active client applications, you may see some cache activity, such as connected clients, memory usage, and operations being performed. This activity is normal during the operation of an Azure Cache for Redis instance.

METRIC	DESCRIPTION
Cache Hits	The number of successful key lookups during the specified reporting interval. This number maps to <code>keyspace_hits</code> from the Redis INFO command.
Cache Latency (Preview)	The latency of the cache calculated based off the internode latency of the cache. This metric is measured in microseconds, and has three dimensions: <code>Avg</code> , <code>Min</code> , and <code>Max</code> , which represent the average, minimum, and maximum latency of the cache respectively during the specified reporting interval.
Cache Misses	The number of failed key lookups during the specified reporting interval. This number maps to <code>keyspace_misses</code> from the Redis INFO command. Cache misses do not necessarily mean there is an issue with the cache. For example, when using the cache-aside programming pattern, an application looks first in the cache for an item. If the item is not there (cache miss), the item is retrieved from the database and added to the cache for next time. Cache misses are normal behavior for the cache-aside programming pattern. If the number of cache misses is higher than expected, examine the application logic that populates and reads from the cache. If items are being evicted from the cache due to memory pressure, then there may be some cache misses, but a better metric to monitor for memory pressure would be <code>Used Memory</code> or <code>Evicted Keys</code> .
Cache Read	The amount of data read from the cache in Megabytes per second (MB/s) during the specified reporting interval. This value is derived from the network interface cards that support the virtual machine that hosts the cache and is not Redis specific. This value corresponds to the network bandwidth used by this cache. If you want to set up alerts for server-side network bandwidth limits, then create it using this <code>Cache Read</code> counter. See this table for the observed bandwidth limits for various cache pricing tiers and sizes.
Cache Write	The amount of data written to the cache in Megabytes per second (MB/s) during the specified reporting interval. This value is derived from the network interface cards that support the virtual machine that hosts the cache and is not Redis specific. This value corresponds to the network bandwidth of data sent to the cache from the client.

METRIC	DESCRIPTION
Connected Clients	The number of client connections to the cache during the specified reporting interval. This number maps to <code>connected_clients</code> from the Redis INFO command. Once the connection limit is reached, subsequent connection attempts to the cache will fail. Even if there are no active client applications, there may still be a few instances of connected clients due to internal processes and connections.
CPU	The CPU utilization of the Azure Cache for Redis server as a percentage during the specified reporting interval. This value maps to the operating system <code>\Processor(_Total)\% Processor Time</code> performance counter.
Errors	Specific failures and performance issues that the cache could be experiencing during a specified reporting interval. This metric has eight dimensions representing different error types, but could have more added in the future. The error types represented now are as follows: <ul style="list-style-type: none"> • Failover – when a cache fails over (subordinate promotes to master) • Dataloss – when there is data loss on the cache • UnresponsiveClients – when the clients are not reading data from the server fast enough • AOF – when there is an issue related to AOF persistence • RDB – when there is an issue related to RDB persistence • Import – when there is an issue related to Import RDB • Export – when there is an issue related to Export RDB
Evicted Keys	The number of items evicted from the cache during the specified reporting interval due to the <code>maxmemory</code> limit. This number maps to <code>evicted_keys</code> from the Redis INFO command.
Expired Keys	The number of items expired from the cache during the specified reporting interval. This value maps to <code>expired_keys</code> from the Redis INFO command.
Gets	The number of get operations from the cache during the specified reporting interval. This value is the sum of the following values from the Redis INFO all command: <code>cmdstat_get</code> , <code>cmdstat_hget</code> , <code>cmdstat_hgetall</code> , <code>cmdstat_hmget</code> , <code>cmdstat_mget</code> , <code>cmdstat_getbit</code> , and <code>cmdstat_getrange</code> , and is equivalent to the sum of cache hits and misses during the reporting interval.
Operations per Second	The total number of commands processed per second by the cache server during the specified reporting interval. This value maps to "instantaneous_ops_per_sec" from the Redis INFO command.

METRIC	DESCRIPTION
Redis Server Load	The percentage of cycles in which the Redis server is busy processing and not waiting idle for messages. If this counter reaches 100, it means the Redis server has hit a performance ceiling and the CPU can't process work any faster. If you are seeing high Redis Server Load, then you will see timeout exceptions in the client. In this case, you should consider scaling up or partitioning your data into multiple caches.
Sets	The number of set operations to the cache during the specified reporting interval. This value is the sum of the following values from the Redis INFO all command: <code>cmdstat_set</code> , <code>cmdstat_hset</code> , <code>cmdstat_hmset</code> , <code>cmdstat_hsetnx</code> , <code>cmdstat_lset</code> , <code>cmdstat_mset</code> , <code>cmdstat_msetnx</code> , <code>cmdstat_setbit</code> , <code>cmdstat_setex</code> , <code>cmdstat_setrange</code> , and <code>cmdstat_setnx</code> .
Total Keys	The maximum number of keys in the cache during the past reporting time period. This number maps to <code>keyspace</code> from the Redis INFO command. Due to a limitation of the underlying metrics system, for caches with clustering enabled, Total Keys returns the maximum number of keys of the shard that had the maximum number of keys during the reporting interval.
Total Operations	The total number of commands processed by the cache server during the specified reporting interval. This value maps to <code>total_commands_processed</code> from the Redis INFO command. When Azure Cache for Redis is used purely for pub/sub there will be no metrics for <code>Cache Hits</code> , <code>Cache Misses</code> , <code>Gets</code> , or <code>Sets</code> , but there will be <code>Total Operations</code> metrics that reflect the cache usage for pub/sub operations.
Used Memory	The amount of cache memory used for key/value pairs in the cache in MB during the specified reporting interval. This value maps to <code>used_memory</code> from the Redis INFO command. This value does not include metadata or fragmentation.
Used Memory Percentage	The % of total memory that is being used during the specified reporting interval. This value references the <code>used_memory</code> value from the Redis INFO command to calculate the percentage.
Used Memory RSS	The amount of cache memory used in MB during the specified reporting interval, including fragmentation and metadata. This value maps to <code>used_memory_rss</code> from the Redis INFO command.

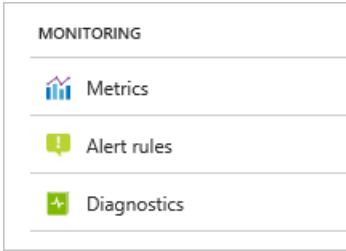
Alerts

You can configure to receive alerts based on metrics and activity logs. Azure Monitor allows you to configure an alert to do the following when it triggers:

- Send an email notification
- Call a webhook

- Invoke an Azure Logic App

To configure Alert rules for your cache, click **Alert rules** from the **Resource menu**.



For more information about configuring and using Alerts, see [Overview of Alerts](#).

Activity Logs

Activity logs provide insight into the operations that were performed on your Azure Cache for Redis instances. It was previously known as "audit logs" or "operational logs". Using activity logs, you can determine the "what, who, and when" for any write operations (PUT, POST, DELETE) taken on your Azure Cache for Redis instances.

NOTE

Activity logs do not include read (GET) operations.

To view activity logs for your cache, click **Activity logs** from the **Resource menu**.

For more information about Activity logs, see [Overview of the Azure Activity Log](#).

Troubleshoot Azure Cache for Redis server-side issues

12/13/2019 • 3 minutes to read • [Edit Online](#)

This section discusses troubleshooting issues that occur because of a condition on an Azure Cache for Redis or the virtual machine(s) hosting it.

- [Memory pressure on Redis server](#)
- [High CPU usage or server load](#)
- [Long-running commands](#)
- [Server-side bandwidth limitation](#)

NOTE

Several of the troubleshooting steps in this guide include instructions to run Redis commands and monitor various performance metrics. For more information and instructions, see the articles in the [Additional information](#) section.

Memory pressure on Redis server

Memory pressure on the server side leads to all kinds of performance problems that can delay processing of requests. When memory pressure hits, the system may page data to disk. This *page faulting* causes the system to slow down significantly. There are several possible causes of this memory pressure:

- The cache is filled with data near its maximum capacity.
- Redis is seeing high memory fragmentation. This fragmentation is most often caused by storing large objects since Redis is optimized for small objects.

Redis exposes two stats through the [INFO](#) command that can help you identify this issue: "used_memory" and "used_memory_rss". You can [view these metrics](#) using the portal.

There are several possible changes you can make to help keep memory usage healthy:

- [Configure a memory policy](#) and set expiration times on your keys. This policy may not be sufficient if you have fragmentation.
- [Configure a maxmemory-reserved value](#) that is large enough to compensate for memory fragmentation.
- Break up your large cached objects into smaller related objects.
- [Create alerts](#) on metrics like used memory to be notified early about potential impacts.
- [Scale](#) to a larger cache size with more memory capacity.

High CPU usage or server load

A high server load or CPU usage means the server can't process requests in a timely fashion. The server may be slow to respond and unable to keep up with request rates.

[Monitor metrics](#) such as CPU or server load. Watch for spikes in CPU usage that correspond with timeouts.

There are several changes you can make to mitigate high server load:

- Investigate what is causing CPU spikes such as [long-running commands](#) noted below or page faulting because of high memory pressure.
- [Create alerts](#) on metrics like CPU or server load to be notified early about potential impacts.
- [Scale](#) to a larger cache size with more CPU capacity.

Long-running commands

Some Redis commands are more expensive to execute than others. The [Redis commands documentation](#) shows the time complexity of each command. Because Redis command processing is single-threaded, a command that takes time to run will block all others that come after it. You should review the commands that you're issuing to your Redis server to understand their performance impacts. For instance, the `KEYS` command is often used without knowing that it's an O(N) operation. You can avoid `KEYS` by using `SCAN` to reduce CPU spikes.

Using the `SLOWLOG` command, you can measure expensive commands being executed against the server.

Server-side bandwidth limitation

Different cache sizes have different network bandwidth capacities. If the server exceeds the available bandwidth, then data won't be sent to the client as quickly. Clients requests could time out because the server can't push data to the client fast enough.

The "Cache Read" and "Cache Write" metrics can be used to see how much server-side bandwidth is being used. You can [view these metrics](#) in the portal.

To mitigate situations where network bandwidth usage is close to maximum capacity:

- Change client call behavior to reduce network demand.
- [Create alerts](#) on metrics like cache read or cache write to be notified early about potential impacts.
- [Scale](#) to a larger cache size with more network bandwidth capacity.

Additional information

- [Troubleshoot Azure Cache for Redis client-side issues](#)
- [What Azure Cache for Redis offering and size should I use?](#)
- [How can I benchmark and test the performance of my cache?](#)
- [How to monitor Azure Cache for Redis](#)
- [How can I run Redis commands?](#)

Troubleshoot Azure Cache for Redis client-side issues

2/21/2020 • 4 minutes to read • [Edit Online](#)

This section discusses troubleshooting issues that occur because of a condition on the Redis client that your application uses.

- [Memory pressure on Redis client](#)
- [Traffic burst](#)
- [High client CPU usage](#)
- [Client-side bandwidth limitation](#)
- [Large request or response size](#)

Memory pressure on Redis client

Memory pressure on the client machine leads to all kinds of performance problems that can delay processing of responses from the cache. When memory pressure hits, the system may page data to disk. This *page faulting* causes the system to slow down significantly.

To detect memory pressure on the client:

- Monitor memory usage on machine to make sure that it doesn't exceed available memory.
- Monitor the client's `Page Faults/Sec` performance counter. During normal operation, most systems have some page faults. Spikes in page faults corresponding with request timeouts can indicate memory pressure.

High memory pressure on the client can be mitigated several ways:

- Dig into your memory usage patterns to reduce memory consumption on the client.
- Upgrade your client VM to a larger size with more memory.

Traffic burst

Bursts of traffic combined with poor `ThreadPool` settings can result in delays in processing data already sent by the Redis Server but not yet consumed on the client side.

Monitor how your `ThreadPool` statistics change over time using [an example `ThreadPoolLogger`](#). You can use `TimeoutException` messages from StackExchange.Redis like below to further investigate:

```
System.TimeoutException: Timeout performing EVAL, inst: 8, mgr: Inactive, queue: 0, qu: 0, qs: 0, qc: 0, wr: 0, wq: 0, in: 64221, ar: 0,  
IOCP: (Busy=6,Free=999,Min=2,Max=1000), WORKER: (Busy=7,Free=8184,Min=2,Max=8191)
```

In the preceding exception, there are several issues that are interesting:

- Notice that in the `IOCP` section and the `WORKER` section you have a `Busy` value that is greater than the `Min` value. This difference means your `ThreadPool` settings need adjusting.
- You can also see `in: 64221`. This value indicates that 64,211 bytes have been received at the client's kernel socket layer but haven't been read by the application. This difference typically means that your application (for example, StackExchange.Redis) isn't reading data from the network as quickly as the server is sending it to you.

You can [configure your `ThreadPool` Settings](#) to make sure that your thread pool scales up quickly under burst scenarios.

High client CPU usage

High client CPU usage indicates the system can't keep up with the work it's been asked to do. Even though the cache sent the response quickly, the client may fail to process the response in a timely fashion.

Monitor the client's system-wide CPU usage using metrics available in the Azure portal or through performance counters on the machine. Be careful not to monitor *process* CPU because a single process can have low CPU usage but the system-wide CPU can be high. Watch for spikes in CPU usage that correspond with timeouts. High CPU may also cause high `in: xxx` values in `TimeoutException` error messages as described in the [Traffic burst](#) section.

NOTE

StackExchange.Redis 1.1.603 and later includes the `local-cpu` metric in `TimeoutException` error messages. Ensure you are using the latest version of the [StackExchange.Redis NuGet package](#). There are bugs constantly being fixed in the code to make it more robust to timeouts so having the latest version is important.

To mitigate a client's high CPU usage:

- Investigate what is causing CPU spikes.
- Upgrade your client to a larger VM size with more CPU capacity.

Client-side bandwidth limitation

Depending on the architecture of client machines, they may have limitations on how much network bandwidth they have available. If the client exceeds the available bandwidth by overloading network capacity, then data isn't processed on the client side as quickly as the server is sending it. This situation can lead to timeouts.

Monitor how your Bandwidth usage change over time using [an example BandwidthLogger](#). This code may not run successfully in some environments with restricted permissions (like Azure web sites).

To mitigate, reduce network bandwidth consumption or increase the client VM size to one with more network capacity.

Large request or response Size

A large request/response can cause timeouts. As an example, suppose your timeout value configured on your client is 1 second. Your application requests two keys (for example, 'A' and 'B') at the same time (using the same physical network connection). Most clients support request "pipelining", where both requests 'A' and 'B' are sent one after the other without waiting for their responses. The server sends the responses back in the same order. If response 'A' is large, it can eat up most of the timeout for later requests.

In the following example, request 'A' and 'B' are sent quickly to the server. The server starts sending responses 'A' and 'B' quickly. Because of data transfer times, response 'B' must wait behind response 'A' times out even though the server responded quickly.

```
|----- 1 Second Timeout (A)-----|
|-Request A-|
|----- 1 Second Timeout (B) -----|
|-Request B-|
    |- Read Response A -----|
        |- Read Response B-| (**TIMEOUT**)
```

This request/response is a difficult one to measure. You could instrument your client code to track large requests and responses.

Resolutions for large response sizes are varied but include:

1. Optimize your application for a large number of small values, rather than a few large values.
 - The preferred solution is to break up your data into related smaller values.
 - See the post [What is the ideal value size range for redis? Is 100 KB too large?](#) for details on why smaller values are recommended.
2. Increase the size of your VM to get higher bandwidth capabilities
 - More bandwidth on your client or server VM may reduce data transfer times for larger responses.
 - Compare your current network usage on both machines to the limits of your current VM size. More bandwidth on only the server or only on the client may not be enough.
3. Increase the number of connection objects your application uses.
 - Use a round-robin approach to make requests over different connection objects.

Additional information

- [Troubleshoot Azure Cache for Redis server-side issues](#)
- [How can I benchmark and test the performance of my cache?](#)

Troubleshoot Azure Cache for Redis timeouts

12/13/2019 • 8 minutes to read • [Edit Online](#)

This section discusses troubleshooting timeout issues that occur when connecting to Azure Cache for Redis.

- [Redis server patching](#)
- [StackExchange.Redis timeout exceptions](#)

NOTE

Several of the troubleshooting steps in this guide include instructions to run Redis commands and monitor various performance metrics. For more information and instructions, see the articles in the [Additional information](#) section.

Redis server patching

Azure Cache for Redis regularly updates its server software as part of the managed service functionality that it provides. This [patching](#) activity takes place largely behind the scene. During the failovers when Redis server nodes are being patched, Redis clients connected to these nodes may experience temporary timeouts as connections are switched between these nodes. See [How does a failover affect my client application](#) for more information on what side-effects patching can have on your application and how you can improve its handling of patching events.

StackExchange.Redis timeout exceptions

StackExchange.Redis uses a configuration setting named `synctimeout` for synchronous operations with a default value of 1000 ms. If a synchronous call doesn't complete in this time, the StackExchange.Redis client throws a timeout error similar to the following example:

```
System.TimeoutException: Timeout performing MGET 2728cc84-58ae-406b-8ec8-3f962419f641, inst: 1,mgr: Inactive,  
queue: 73, qu=6, qs=67, qc=0, wr=1/1, in=0/0 IOCP: (Busy=6, Free=999, Min=2,Max=1000), WORKER  
(Busy=7,Free=8184,Min=2,Max=8191)
```

This error message contains metrics that can help point you to the cause and possible resolution of the issue. The following table contains details about the error message metrics.

ERROR MESSAGE METRIC	DETAILS
inst	In the last time slice: 0 commands have been issued
mgr	The socket manager is doing <code>socket.select</code> , which means it's asking the OS to indicate a socket that has something to do. The reader isn't actively reading from the network because it doesn't think there's anything to do
queue	There are 73 total in-progress operations
qu	6 of the in-progress operations are in the unsent queue and haven't yet been written to the outbound network

ERROR MESSAGE METRIC	DETAILS
qs	67 of the in-progress operations have been sent to the server but a response isn't yet available. The response could be Not yet sent by the server or sent by the server but not yet processed by the client.
qc	0 of the in-progress operations have seen replies but haven't yet been marked as complete because they're waiting on the completion loop
wr	There's an active writer (meaning the 6 unsent requests aren't being ignored) bytes/activewriters
in	There are no active readers and zero bytes are available to be read on the NIC bytes/activereaders

You can use the following steps to investigate possible root causes.

1. As a best practice, make sure you're using the following pattern to connect when using the StackExchange.Redis client.

```
private static Lazy<ConnectionMultiplexer> lazyConnection = new Lazy<ConnectionMultiplexer>(() =>
{
    return
        ConnectionMultiplexer.Connect("cachename.redis.cache.windows.net,abortConnect=false,ssl=true,password=..");
});

public static ConnectionMultiplexer Connection
{
    get
    {
        return lazyConnection.Value;
    }
}
```

For more information, see [Connect to the cache using StackExchange.Redis](#).

2. Ensure that your server and the client application are in the same region in Azure. For example, you might be getting timeouts when your cache is in East US but the client is in West US and the request doesn't complete within the `synctimeout` interval or you might be getting timeouts when you're debugging from your local development machine.

It's highly recommended to have the cache and the client in the same Azure region. If you have a scenario that includes cross region calls, you should set the `synctimeout` interval to a value higher than the default 1000-ms interval by including a `synctimeout` property in the connection string. The following example shows a snippet of a connection string for StackExchange.Redis provided by Azure Cache for Redis with a `synctimeout` of 2000 ms.

```
synctimeout=2000,cachename.redis.cache.windows.net,abortConnect=false,ssl=true,password=...
```

3. Ensure you are using the latest version of the [StackExchange.Redis NuGet package](#). There are bugs constantly being fixed in the code to make it more robust to timeouts so having the latest version is important.
4. If your requests are bound by bandwidth limitations on the server or client, it takes longer for them to

complete and can cause timeouts. To see if your timeout is because of network bandwidth on the server, see [Server-side bandwidth limitation](#). To see if your timeout is because of client network bandwidth, see [Client-side bandwidth limitation](#).

5. Are you getting CPU bound on the server or on the client?

- Check if you're getting bound by CPU on your client. High CPU could cause the request to not be processed within the `synctimeout` interval and cause a request to time out. Moving to a larger client size or distributing the load can help to control this problem.
- Check if you're getting CPU bound on the server by monitoring the CPU [cache performance metric](#). Requests coming in while Redis is CPU bound can cause those requests to time out. To address this condition, you can distribute the load across multiple shards in a premium cache, or upgrade to a larger size or pricing tier. For more information, see [Server-side bandwidth limitation](#).

6. Are there commands taking long time to process on the server? Long-running commands that are taking long time to process on the redis-server can cause timeouts. For more information about long-running commands, see [Long-running commands](#). You can connect to your Azure Cache for Redis instance using the redis-cli client or the [Redis Console](#). Then, run the `SLOWLOG` command to see if there are requests slower than expected. Redis Server and StackExchange.Redis are optimized for many small requests rather than fewer large requests. Splitting your data into smaller chunks may improve things here.

For information on connecting to your cache's SSL endpoint using redis-cli and stunnel, see the blog post [Announcing ASP.NET Session State Provider for Redis Preview Release](#).

7. High Redis server load can cause timeouts. You can monitor the server load by monitoring the `Redis Server Load` [cache performance metric](#). A server load of 100 (maximum value) signifies that the redis server has been busy, with no idle time, processing requests. To see if certain requests are taking up all of the server capability, run the SlowLog command, as described in the previous paragraph. For more information, see [High CPU usage / Server Load](#).

8. Was there any other event on the client side that could have caused a network blip? Common events include: scaling the number of client instances up or down, deploying a new version of the client, or autoscale enabled. In our testing, we have found that autoscale or scaling up/down can cause outbound network connectivity to be lost for several seconds. StackExchange.Redis code is resilient to such events and reconnects. While reconnecting, any requests in the queue can time out.

9. Was there a large request preceding several small requests to the cache that timed out? The parameter `qs` in the error message tells you how many requests were sent from the client to the server, but haven't processed a response. This value can keep growing because StackExchange.Redis uses a single TCP connection and can only read one response at a time. Even though the first operation timed out, it doesn't stop more data from being sent to or from the server. Other requests will be blocked until the large request is finished and can cause time outs. One solution is to minimize the chance of timeouts by ensuring that your cache is large enough for your workload and splitting large values into smaller chunks. Another possible solution is to use a pool of `ConnectionMultiplexer` objects in your client, and choose the least loaded `ConnectionMultiplexer` when sending a new request. Loading across multiple connection objects should prevent a single timeout from causing other requests to also time out.

10. If you're using `RedisSessionStateProvider`, ensure you have set the retry timeout correctly.

`retryTimeoutInMilliseconds` should be higher than `operationTimeoutInMilliseconds`, otherwise no retries occur. In the following example `retryTimeoutInMilliseconds` is set to 3000. For more information, see [ASP.NET Session State Provider for Azure Cache for Redis](#) and [How to use the configuration parameters of Session State Provider and Output Cache Provider](#).

```
<add  
    name="AFRedisCacheSessionstateProvider"  
    type="Microsoft.Web.Redis.RedisSessionstateProvider"  
    host="enbwcache.redis.cache.windows.net"  
    port="6380"  
    accessKey="..."  
    ssl="true"  
    databaseId="0"  
    applicationName="AFRedisCacheSessionState"  
    connectionTimeoutInMilliseconds = "5000"  
    operationTimeoutInMilliseconds = "1000"  
    retryTimeoutInMilliseconds="3000" />
```

11. Check memory usage on the Azure Cache for Redis server by [monitoring](#) `Used Memory RSS` and `Used Memory`. If an eviction policy is in place, Redis starts evicting keys when `Used_Memory` reaches the cache size. Ideally, `Used Memory RSS` should be only slightly higher than `Used memory`. A large difference means there's memory fragmentation (internal or external). When `Used Memory RSS` is less than `Used Memory`, it means part of the cache memory has been swapped by the operating system. If this swapping occurs, you can expect some significant latencies. Because Redis doesn't have control over how its allocations are mapped to memory pages, high `Used Memory RSS` is often the result of a spike in memory usage. When Redis server frees memory, the allocator takes the memory but it may or may not give the memory back to the system. There may be a discrepancy between the `Used Memory` value and memory consumption as reported by the operating system. Memory may have been used and released by Redis but not given back to the system. To help mitigate memory issues, you can do the following steps:

- Upgrade the cache to a larger size so that you aren't running against memory limitations on the system.
- Set expiration times on the keys so that older values are evicted proactively.
- Monitor the `used_memory_rss` cache metric. When this value approaches the size of their cache, you're likely to start seeing performance issues. Distribute the data across multiple shards if you're using a premium cache, or upgrade to a larger cache size.

For more information, see [Memory pressure on Redis server](#).

Additional information

- [Troubleshoot Azure Cache for Redis client-side issues](#)
- [Troubleshoot Azure Cache for Redis server-side issues](#)
- [How can I benchmark and test the performance of my cache?](#)
- [How to monitor Azure Cache for Redis](#)

Troubleshoot data loss in Azure Cache for Redis

12/27/2019 • 4 minutes to read • [Edit Online](#)

This article discusses how to diagnose actual or perceived data losses that might occur in Azure Cache for Redis.

NOTE

Several of the troubleshooting steps in this guide include instructions to run Redis commands and monitor various performance metrics. For more information and instructions, see the articles in the [Additional information](#) section.

Partial loss of keys

Azure Cache for Redis doesn't randomly delete keys after they've been stored in memory. However, it does remove keys in response to expiration or eviction policies and to explicit key-deletion commands. Keys that have been written to the master node in a Premium or Standard Azure Cache for Redis instance also might not be available on a replica right away. Data is replicated from the master to the replica in an asynchronous and non-blocking manner.

If you find that keys have disappeared from your cache, check the following possible causes:

CAUSE	DESCRIPTION
Key expiration	Keys are removed because of time-outs set on them.
Key eviction	Keys are removed under memory pressure.
Key deletion	Keys are removed by explicit delete commands.
Async replication	Keys are not available on a replica because of data-replication delays.

Key expiration

Azure Cache for Redis removes a key automatically if the key is assigned a time-out and that period has passed. For more information about Redis key expiration, see the [EXPIRE](#) command documentation. Time-out values also can be set by using the [SET](#), [SETEX](#), [GETSET](#), and other [*STORE](#) commands.

To get stats on how many keys have expired, use the [INFO](#) command. The `Stats` section shows the total number of expired keys. The `Keyspace` section provides more information about the number of keys with time-outs and the average time-out value.

```
# Stats  
  
expired_keys:46583  
  
# Keyspace  
  
db0:keys=3450,expires=2,avg_ttl=91861015336
```

You can also look at diagnostic metrics for your cache, to see if there's a correlation between when the key went missing and a spike in expired keys. See the Appendix of [Debugging Redis Keyspace Misses](#) for information about

using keyspace notifications or **MONITOR** to debug these types of issues.

Key eviction

Azure Cache for Redis requires memory space to store data. It purges keys to free up available memory when necessary. When the **used_memory** or **used_memory_rss** values in the [INFO](#) command approach the configured **maxmemory** setting, Azure Cache for Redis starts evicting keys from memory based on [cache policy](#).

You can monitor the number of evicted keys by using the [INFO](#) command:

```
# Stats  
  
evicted_keys:13224
```

You can also look at diagnostic metrics for your cache, to see if there's a correlation between when the key went missing and a spike in evicted keys. See the Appendix of [Debugging Redis Keyspace Misses](#) for information about using keyspace notifications or **MONITOR** to debug these types of issues.

Key deletion

Redis clients can issue the **DEL** or **HDEL** command to explicitly remove keys from Azure Cache for Redis. You can track the number of delete operations by using the [INFO](#) command. If **DEL** or **HDEL** commands have been called, they'll be listed in the [Commandstats](#) section.

```
# Commandstats  
  
cmdstat_del:calls=2,usec=90,usec_per_call=45.00  
  
cmdstat_hdel:calls=1,usec=47,usec_per_call=47.00
```

Async replication

Any Azure Cache for Redis instance in the Standard or Premium tier is configured with a master node and at least one replica. Data is copied from the master to a replica asynchronously by using a background process. The [redis.io](#) website describes how Redis data replication works in general. For scenarios where clients write to Redis frequently, partial data loss can occur because this replication is guaranteed to be instantaneous. For example, if the master goes down *after* a client writes a key to it, but *before* the background process has a chance to send that key to the replica, the key is lost when the replica takes over as the new master.

Major or complete loss of keys

If most or all keys have disappeared from your cache, check the following possible causes:

CAUSE	DESCRIPTION
Key flushing	Keys have been purged manually.
Incorrect database selection	Azure Cache for Redis is set to use a non-default database.
Redis instance failure	The Redis server is unavailable.

Key flushing

Clients can call the **FLUSHDB** command to remove all keys in a *single* database or **FLUSHALL** to remove all keys from *all* databases in a Redis cache. To find out whether keys have been flushed, use the [INFO](#) command. The [Commandstats](#) section shows whether either **FLUSH** command has been called:

```
# Commandstats

cmdstat_flushall:calls=2,usec=112,usec_per_call=56.00

cmdstat_flushdb:calls=1,usec=110,usec_per_call=52.00
```

Incorrect database selection

Azure Cache for Redis uses the **db0** database by default. If you switch to another database (for example, **db1**) and try to read keys from it, Azure Cache for Redis won't find them there. Every database is a logically separate unit and holds a different dataset. Use the [SELECT](#) command to use other available databases and look for keys in each of them.

Redis instance failure

Redis is an in-memory data store. Data is kept on the physical or virtual machines that host the Redis cache. An Azure Cache for Redis instance in the Basic tier runs on only a single virtual machine (VM). If that VM is down, all data that you've stored in the cache is lost.

Caches in the Standard and Premium tiers offer much higher resiliency against data loss by using two VMs in a replicated configuration. When the master node in such a cache fails, the replica node takes over to serve data automatically. These VMs are located on separate domains for faults and updates, to minimize the chance of both becoming unavailable simultaneously. If a major datacenter outage happens, however, the VMs might still go down together. Your data will be lost in these rare cases.

Consider using [Redis data persistence](#) and [geo-replication](#) to improve protection of your data against these infrastructure failures.

Additional information

- [Troubleshoot Azure Cache for Redis server-side issues](#)
- [What Azure Cache for Redis offering and size should I use?](#)
- [How to monitor Azure Cache for Redis](#)
- [How can I run Redis commands?](#)

How to Scale Azure Cache for Redis

1/24/2020 • 8 minutes to read • [Edit Online](#)

Azure Cache for Redis has different cache offerings, which provide flexibility in the choice of cache size and features. After a cache is created, you can scale the size and the pricing tier of the cache if the requirements of your application change. This article shows you how to scale your cache using the Azure portal, and tools such as Azure PowerShell, and Azure CLI.

When to scale

You can use the [monitoring](#) features of Azure Cache for Redis to monitor the health and performance of your cache and help determine when to scale the cache.

You can monitor the following metrics to help determine if you need to scale.

- Redis Server Load
- Memory Usage
- Network Bandwidth
- CPU Usage

If you determine that your cache is no longer meeting your application's requirements, you can scale to a larger or smaller cache pricing tier that is right for your application. For more information on determining which cache pricing tier to use, see [What Azure Cache for Redis offering and size should I use](#).

Scale a cache

To scale your cache, [browse to the cache](#) in the [Azure portal](#) and click **Scale** from the **Resource menu**.



Select the desired pricing tier from the **Select pricing tier** blade and click **Select**.

Select pricing tier

You can scale your cache instance up or down. [Learn more](#)

★ Recommended | View all

P1 Premium	P2 Premium	P3 Premium
6 GB Cache	13 GB Cache	26 GB Cache
Replication	Replication	Replication
Moderate network bandwidth	Moderate network bandwidth	High network bandwidth
All Standard features	All Standard features	All Standard features
Data Persistence	Data Persistence	Data Persistence
Virtual Network	Virtual Network	Virtual Network
Redis Cluster	Redis Cluster	Redis Cluster
99.9% SLA	99.9% SLA	99.9% SLA
412.92	825.84	1,650.94
USD/MONTH (ESTIMATED) PER SH...	USD/MONTH (ESTIMATED) PER SH...	USD/MONTH (ESTIMATED) PER SH...
P4 Premium	C1 Standard	C2 Standard ★
53 GB Cache	1 GB Cache	2.5 GB Cache
Replication	Replication	Replication
Highest network bandwidth	Low network bandwidth	Moderate network bandwidth
All Standard features	Dedicated service	Dedicated service
Data Persistence	SSL	SSL
Virtual Network	Up to 1,000 connections	Up to 2,000 connections
Redis Cluster	Configure Redis (keys)	Configure Redis (keys)
Select		

You can scale to a different pricing tier with the following restrictions:

- You can't scale from a higher pricing tier to a lower pricing tier.
 - You can't scale from a **Premium** cache down to a **Standard** or a **Basic** cache.
 - You can't scale from a **Standard** cache down to a **Basic** cache.
- You can scale from a **Basic** cache to a **Standard** cache but you can't change the size at the same time. If you need a different size, you can do a subsequent scaling operation to the desired size.
- You can't scale from a **Basic** cache directly to a **Premium** cache. First, scale from **Basic** to **Standard** in one scaling operation, and then from **Standard** to **Premium** in a subsequent scaling operation.
- You can't scale from a larger size down to the **C0 (250 MB)** size.

While the cache is scaling to the new pricing tier, a **Scaling** status is displayed in the **Azure Cache for Redis** blade.

The screenshot shows the Azure portal interface. On the left, there's a sidebar with options like 'Console', 'Move', and 'Delete'. Below that is a section for 'Essentials' with a dropdown arrow. Under 'Resource group (change)', it lists 'contosoGroup'. There are buttons for 'Status' and 'Scaling...', which is highlighted with a red box. Other details shown include 'Location: Central US', 'Subscription name (change): Prototype3', and 'Subscription ID' (redacted). On the right, a 'Notifications' pane is open, displaying a message about scaling a Redis Cache.

When scaling is complete, the status changes from **Scaling** to **Running**.

How to automate a scaling operation

In addition to scaling your cache instances in the Azure portal, you can scale using PowerShell cmdlets, Azure CLI, and by using the Microsoft Azure Management Libraries (MAML).

- [Scale using PowerShell](#)
- [Scale using Azure CLI](#)
- [Scale using MAML](#)

Scale using PowerShell

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

You can scale your Azure Cache for Redis instances with PowerShell by using the `Set-AzRedisCache` cmdlet when the `Size`, `Sku`, or `ShardCount` properties are modified. The following example shows how to scale a cache named `myCache` to a 2.5 GB cache.

```
Set-AzRedisCache -ResourceGroupName myGroup -Name myCache -Size 2.5GB
```

For more information on scaling with PowerShell, see [To scale an Azure Cache for Redis using Powershell](#).

Scale using Azure CLI

To scale your Azure Cache for Redis instances using Azure CLI, call the `azure redisache set` command and pass in the desired configuration changes that include a new size, sku, or cluster size, depending on the desired scaling operation.

For more information on scaling with Azure CLI, see [Change settings of an existing Azure Cache for Redis](#).

Scale using MAML

To scale your Azure Cache for Redis instances using the [Microsoft Azure Management Libraries \(MAML\)](#), call the `IRedisOperations.CreateOrUpdate` method and pass in the new size for the `RedisProperties.SKU.Capacity`.

```

static void Main(string[] args)
{
    // For instructions on getting the access token, see
    // https://azure.microsoft.com/documentation/articles/cache-configure/#access-keys
    string token = GetAuthorizationHeader();

    TokenCloudCredentials creds = new TokenCloudCredentials(subscriptionId,token);

    RedisManagementClient client = new RedisManagementClient(creds);
    var redisProperties = new RedisProperties();

    // To scale, set a new size for the redisSKUCapacity parameter.
    redisProperties.Sku = new Sku(redisSKUName,redisSKUFamily,redisSKUCapacity);
    redisProperties.RedisVersion = redisVersion;
    var redisParams = new RedisCreateOrUpdateParameters(redisProperties, redisCacheRegion);
    client.Redis.CreateOrUpdate(resourceGroupName,cacheName, redisParams);
}

```

For more information, see the [Manage Azure Cache for Redis using MAML](#) sample.

Scaling FAQ

The following list contains answers to commonly asked questions about Azure Cache for Redis scaling.

- [Can I scale to, from, or within a Premium cache?](#)
- [After scaling, do I have to change my cache name or access keys?](#)
- [How does scaling work?](#)
- [Will I lose data from my cache during scaling?](#)
- [Is my custom databases setting affected during scaling?](#)
- [Will my cache be available during scaling?](#)
- With Geo-replication configured, why am I not able to scale my cache or change the shards in a cluster?
- [Operations that are not supported](#)
- [How long does scaling take?](#)
- [How can I tell when scaling is complete?](#)

Can I scale to, from, or within a Premium cache?

- You can't scale from a **Premium** cache down to a **Basic** or **Standard** pricing tier.
- You can scale from one **Premium** cache pricing tier to another.
- You can't scale from a **Basic** cache directly to a **Premium** cache. First, scale from **Basic** to **Standard** in one scaling operation, and then from **Standard** to **Premium** in a subsequent scaling operation.
- If you enabled clustering when you created your **Premium** cache, you can [change the cluster size](#). If your cache was created without clustering enabled, you can configure clustering at a later time.

For more information, see [How to configure clustering for a Premium Azure Cache for Redis](#).

After scaling, do I have to change my cache name or access keys?

No, your cache name and keys are unchanged during a scaling operation.

How does scaling work?

- When a **Basic** cache is scaled to a different size, it is shut down and a new cache is provisioned using the new size. During this time, the cache is unavailable and all data in the cache is lost.
- When a **Basic** cache is scaled to a **Standard** cache, a replica cache is provisioned and the data is copied from the primary cache to the replica cache. The cache remains available during the scaling process.
- When a **Standard** cache is scaled to a different size or to a **Premium** cache, one of the replicas is shut down

and reprovisioned to the new size and the data transferred over, and then the other replica performs a failover before it is reprovisioned, similar to the process that occurs during a failure of one of the cache nodes.

Will I lose data from my cache during scaling?

- When a **Basic** cache is scaled to a new size, all data is lost and the cache is unavailable during the scaling operation.
- When a **Basic** cache is scaled to a **Standard** cache, the data in the cache is typically preserved.
- When a **Standard** cache is scaled to a larger size or tier, or a **Premium** cache is scaled to a larger size, all data is typically preserved. When scaling a **Standard** or **Premium** cache down to a smaller size, data may be lost depending on how much data is in the cache related to the new size when it is scaled. If data is lost when scaling down, keys are evicted using the [allkeys-lru](#) eviction policy.

Is my custom databases setting affected during scaling?

If you configured a custom value for the `databases` setting during cache creation, keep in mind that some pricing tiers have different [databases limits](#). Here are some considerations when scaling in this scenario:

- When scaling to a pricing tier with a lower `databases` limit than the current tier:
 - If you are using the default number of `databases`, which is 16 for all pricing tiers, no data is lost.
 - If you are using a custom number of `databases` that falls within the limits for the tier to which you are scaling, this `databases` setting is retained and no data is lost.
 - If you are using a custom number of `databases` that exceeds the limits of the new tier, the `databases` setting is lowered to the limits of the new tier and all data in the removed databases is lost.
- When scaling to a pricing tier with the same or higher `databases` limit than the current tier, your `databases` setting is retained and no data is lost.

While Standard and Premium caches have a 99.9% SLA for availability, there is no SLA for data loss.

Will my cache be available during scaling?

- **Standard** and **Premium** caches remain available during the scaling operation. However, connection blips can occur while scaling Standard and Premium caches, and also while scaling from Basic to Standard caches. These connection blips are expected to be small and redis clients should be able to re-establish their connection instantly.
- **Basic** caches are offline during scaling operations to a different size. Basic caches remain available when scaling from **Basic** to **Standard** but, may experience a small connection blip. If a connection blip occurs, redis clients should be able to re-establish their connection instantly.

Scaling limitations with Geo-replication

Once you have added a Geo-replication link between two caches, you will no longer be able to initiate a scaling operation or change the number of shards in a cluster. You must unlink the cache to issue these commands. For more information, see [Configure Geo-replication](#).

Operations that are not supported

- You can't scale from a higher pricing tier to a lower pricing tier.
 - You can't scale from a **Premium** cache down to a **Standard** or a **Basic** cache.
 - You can't scale from a **Standard** cache down to a **Basic** cache.
- You can scale from a **Basic** cache to a **Standard** cache but you can't change the size at the same time. If you need a different size, you can do a subsequent scaling operation to the desired size.
- You can't scale from a **Basic** cache directly to a **Premium** cache. First scale from **Basic** to **Standard** in one scaling operation, and then scale from **Standard** to **Premium** in a subsequent operation.
- You can't scale from a larger size down to the **C0 (250 MB)** size.

If a scaling operation fails, the service tries to revert the operation, and the cache will revert to the original size.

How long does scaling take?

Scaling time depends on how much data is in the cache, with larger amounts of data taking a longer time to complete. Scaling takes approximately 20 minutes. For clustered caches, scaling takes approximately 20 minutes per shard.

How can I tell when scaling is complete?

In the Azure portal, you can see the scaling operation in progress. When scaling is complete, the status of the cache changes to **Running**.

How to configure Redis clustering for a Premium Azure Cache for Redis

1/23/2020 • 9 minutes to read • [Edit Online](#)

Azure Cache for Redis has different cache offerings, which provide flexibility in the choice of cache size and features, including Premium tier features such as clustering, persistence, and virtual network support. This article describes how to configure clustering in a premium Azure Cache for Redis instance.

For information on other premium cache features, see [Introduction to the Azure Cache for Redis Premium tier](#).

What is Redis Cluster?

Azure Cache for Redis offers Redis cluster as [implemented in Redis](#). With Redis Cluster, you get the following benefits:

- The ability to automatically split your dataset among multiple nodes.
- The ability to continue operations when a subset of the nodes is experiencing failures or are unable to communicate with the rest of the cluster.
- More throughput: Throughput increases linearly as you increase the number of shards.
- More memory size: Increases linearly as you increase the number of shards.

Clustering does not increase the number of connections available for a clustered cache. For more information about size, throughput, and bandwidth with premium caches, see [What Azure Cache for Redis offering and size should I use?](#)

In Azure, Redis cluster is offered as a primary/replica model where each shard has a primary/replica pair with replication where the replication is managed by Azure Cache for Redis service.

Clustering

Clustering is enabled on the **New Azure Cache for Redis** blade during cache creation.

To create a premium cache, sign in to the [Azure portal](#) and click **Create a resource > Databases > Azure Cache for Redis**.

The screenshot shows the Azure Marketplace interface under the 'Databases' category. The left sidebar lists various Azure services, and the main area displays a grid of database offerings. The 'Databases' category is highlighted with a red box. Within this category, the 'Redis Cache' option is also highlighted with a red box.

Category	Offering	Description
Databases	SQL Database	Quickstart tutorial
	SQL Data Warehouse	Learn more
	SQL Elastic database pool	Learn more
	Azure Database for MySQL (preview)	Learn more
	Azure Database for PostgreSQL (preview)	Learn more
	SQL Server 2017 Enterprise Windows Server 2016	Learn more
	Azure Cosmos DB	Quickstart tutorial
	Database as a service for MongoDB	Learn more
	Redis Cache	Learn more
Data + Analytics		
AI + Cognitive Services		
Internet of Things		
Enterprise Integration		
Security + Identity		
Developer tools		
Monitoring + Management		
Add-ons		
Blockchain		

NOTE

In addition to creating caches in the Azure portal, you can also create them using Resource Manager templates, PowerShell, or Azure CLI. For more information about creating an Azure Cache for Redis, see [Create a cache](#).

To configure premium features, first select one of the premium pricing tiers in the **Pricing tier** drop-down list. For more information about each pricing tier, click **View full pricing details** and select a pricing tier from the **Choose your pricing tier** blade.

The screenshot shows two overlapping windows. The left window is titled "New Redis Cache" and contains fields for DNS name (contoso5premium), Subscription (Prototype3), Resource group (Create new), Location (Central US), and Pricing tier (Premium P1 (6 GB Cache, Replication)). A red box labeled "1" highlights the "Pricing tier" dropdown. The right window is titled "Choose your pricing tier" and lists various Redis Cache plans across Premium and Standard tiers. A red box labeled "2" highlights the "P1 Premium" plan, which includes 6 GB Cache, Replication, and a price of 412.92 USD/MONTH. A red box labeled "3" highlights the "Select" button at the bottom of the pricing table.

Pricing Tier	Plan	Cache Size	Replication	Network Bandwidth	Standard Features	Data Persistence	Virtual Network	Redis Cluster	SLA	Price (USD/Month)	
Premium	P1 Premium	6 GB Cache	Replication	Moderate network bandwidth	All Standard features	Data Persistence	Virtual Network	Redis Cluster	99.9% SLA	412.92	
	P2 Premium	13 GB Cache	Replication	Moderate network bandwidth	All Standard features	Data Persistence	Virtual Network	Redis Cluster	99.9% SLA	825.84	
	P3 Premium	26 GB Cache	Replication	High network bandwidth	All Standard features	Data Persistence	Virtual Network	Redis Cluster	99.9% SLA	1,650.94	
	Standard	C0 Standard	250 MB Cache	Replication	Low network bandwidth	Shared infrastructure	SSL	Up to 256 connections	Configure Redis (key-value)	99.9% SLA	40.92
		C1 Standard	1 GB Cache	Replication	Low network bandwidth	Dedicated service	SSL	Up to 1,000 connections	Configure Redis (key-value)	99.9% SLA	102.67
		C2 Standard	2.5 GB Cache								
		C3 Standard	6 GB Cache								
C4 Standard		13 GB Cache									

Clustering is configured on the **Redis Cluster** blade.

The screenshot shows two side-by-side blades. On the left is the 'New Redis Cache' blade, which includes fields for 'Redis name', 'Subscription', 'Resource group', 'Location', and 'Pricing tier' (Premium: S-08). A red box highlights the 'Redis Cluster' section, which is currently set to 'Not configured'. On the right is the 'Redis Cluster' configuration blade, showing 'Clustering' settings where 'Enabled' is selected (highlighted by a blue box) and 'Disabled' is unselected. Below it is a slider for 'Shard count' with a value of 1, and a note 'Total size: 6 GB'. At the bottom of both blades are 'Create' and 'OK' buttons.

You can have up to 10 shards in the cluster. Click **Enabled** and slide the slider or type a number between 1 and 10 for **Shard count** and click **OK**.

Each shard is a primary/replica cache pair managed by Azure, and the total size of the cache is calculated by multiplying the number of shards by the cache size selected in the pricing tier.

This screenshot shows the 'Redis Cluster' configuration blade. Step 1 highlights the 'Enabled' button under 'Clustering'. Step 2 highlights the 'Shard count' slider and its value of 3, with a note 'Total size: 18 GB'. Step 3 highlights the 'OK' button at the bottom.

Once the cache is created you connect to it and use it just like a non-clustered cache, and Redis distributes the data throughout the Cache shards. If diagnostics is [enabled](#), metrics are captured separately for each shard and can be [viewed](#) in the Azure Cache for Redis blade.

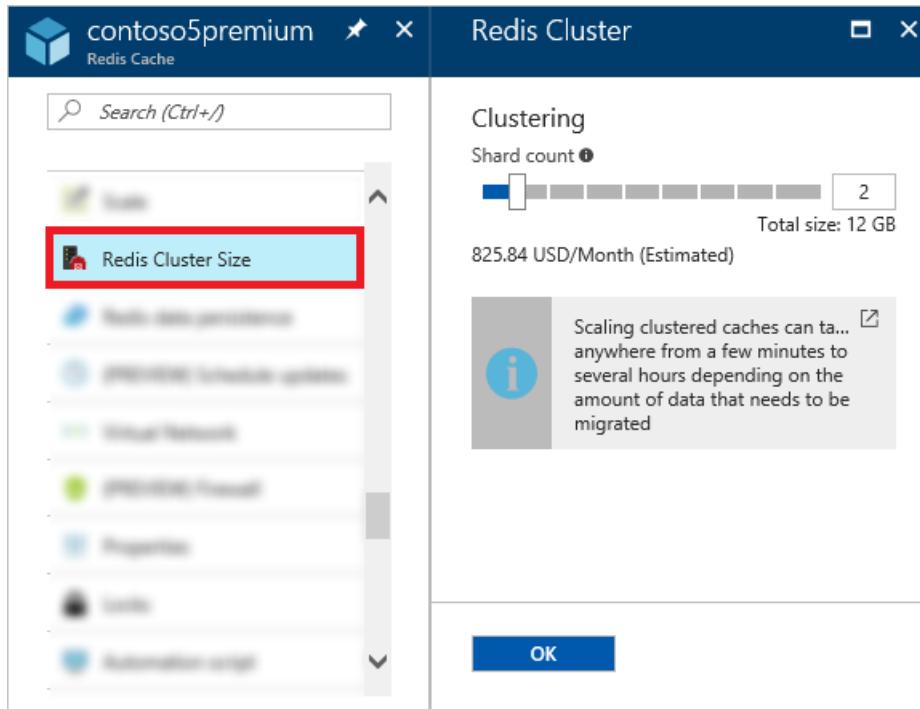
NOTE

There are some minor differences required in your client application when clustering is configured. For more information, see [Do I need to make any changes to my client application to use clustering?](#)

For sample code on working with clustering with the StackExchange.Redis client, see the [clustering.cs](#) portion of the [Hello World](#) sample.

Change the cluster size on a running premium cache

To change the cluster size on a running premium cache with clustering enabled, click **Cluster Size** from the **Resource menu**.



To change the cluster size, use the slider or type a number between 1 and 10 in the **Shard count** text box and click **OK** to save.

Increasing the cluster size increases max throughput and cache size. Increasing the cluster size doesn't increase the max. connections available to clients.

NOTE

Scaling a cluster runs the [MIGRATE](#) command, which is an expensive command, so for minimal impact, consider running this operation during non-peak hours. During the migration process, you will see a spike in server load. Scaling a cluster is a long running process and the amount of time taken depends on the number of keys and size of the values associated with those keys.

Clustering FAQ

The following list contains answers to commonly asked questions about Azure Cache for Redis clustering.

- [Do I need to make any changes to my client application to use clustering?](#)
- [How are keys distributed in a cluster?](#)
- [What is the largest cache size I can create?](#)
- [Do all Redis clients support clustering?](#)

- How do I connect to my cache when clustering is enabled?
- Can I directly connect to the individual shards of my cache?
- Can I configure clustering for a previously created cache?
- Can I configure clustering for a basic or standard cache?
- Can I use clustering with the Redis ASP.NET Session State and Output Caching providers?
- I am getting MOVE exceptions when using StackExchange.Redis and clustering, what should I do?

Do I need to make any changes to my client application to use clustering?

- When clustering is enabled, only database 0 is available. If your client application uses multiple databases and it tries to read or write to a database other than 0, the following exception is thrown.

```
Unhandled Exception: StackExchange.Redis.RedisConnectionException: ProtocolFailure on GET --->
StackExchange.Redis.RedisCommandException: Multiple databases are not supported on this server; cannot
switch to database: 6
```

For more information, see [Redis Cluster Specification - Implemented subset](#).

- If you are using [StackExchange.Redis](#), you must use 1.0.481 or later. You connect to the cache using the same [endpoints, ports, and keys](#) that you use when connecting to a cache that does not have clustering enabled. The only difference is that all reads and writes must be done to database 0.
 - Other clients may have different requirements. See [Do all Redis clients support clustering?](#)
- If your application uses multiple key operations batched into a single command, all keys must be located in the same shard. To locate keys in the same shard, see [How are keys distributed in a cluster?](#)
- If you are using Redis ASP.NET Session State provider you must use 2.0.1 or higher. See [Can I use clustering with the Redis ASP.NET Session State and Output Caching providers?](#)

How are keys distributed in a cluster?

Per the Redis [Keys distribution model](#) documentation: The key space is split into 16384 slots. Each key is hashed and assigned to one of these slots, which are distributed across the nodes of the cluster. You can configure which part of the key is hashed to ensure that multiple keys are located in the same shard using hash tags.

- Keys with a hash tag - if any part of the key is enclosed in `{` and `}`, only that part of the key is hashed for the purposes of determining the hash slot of a key. For example, the following 3 keys would be located in the same shard: `{key}1`, `{key}2`, and `{key}3` since only the `key` part of the name is hashed. For a complete list of keys hash tag specifications, see [Keys hash tags](#).
- Keys without a hash tag - the entire key name is used for hashing. This results in a statistically even distribution across the shards of the cache.

For best performance and throughput, we recommend distributing the keys evenly. If you are using keys with a hash tag it is the application's responsibility to ensure the keys are distributed evenly.

For more information, see [Keys distribution model](#), [Redis Cluster data sharding](#), and [Keys hash tags](#).

For sample code on working with clustering and locating keys in the same shard with the StackExchange.Redis client, see the [clustering.cs](#) portion of the [Hello World](#) sample.

What is the largest cache size I can create?

The largest premium cache size is 120 GB. You can create up to 10 shards giving you a maximum size of 1.2TB GB. If you need a larger size you can [request more](#). For more information, see [Azure Cache for Redis Pricing](#).

Do all Redis clients support clustering?

Not all clients support Redis clustering! Please check the documentation for the library you are using, to verify you are using a library and version which support clustering. StackExchange.Redis is one library that does support clustering, in its newer versions. For more information on other clients, see the [Playing with the cluster](#) section of the [Redis cluster tutorial](#).

The Redis clustering protocol requires each client to connect to each shard directly in clustering mode, and also defines new error responses such as 'MOVED' na 'CROSSLOTS'. Attempting to use a client that doesn't support clustering with a cluster mode cache can result in a lot of [MOVED redirection exceptions](#), or just break your application, if you are doing cross-slot multi-key requests.

NOTE

If you are using StackExchange.Redis as your client, ensure you are using the latest version of [StackExchange.Redis](#) 1.0.481 or later for clustering to work correctly. If you have any issues with move exceptions, see [move exceptions](#) for more information.

How do I connect to my cache when clustering is enabled?

You can connect to your cache using the same [endpoints](#), [ports](#), and [keys](#) that you use when connecting to a cache that does not have clustering enabled. Redis manages the clustering on the backend so you don't have to manage it from your client.

Can I directly connect to the individual shards of my cache?

The clustering protocol requires that the client make the correct shard connections. So the client should do this correctly for you. With that said, each shard consists of a primary/replica cache pair, collectively known as a cache instance. You can connect to these cache instances using the redis-cli utility in the [unstable](#) branch of the Redis repository at GitHub. This version implements basic support when started with the `-c` switch. For more information, see [Playing with the cluster](#) on <https://redis.io> in the Redis cluster tutorial.

For non-ssl, use the following commands.

```
Redis-cli.exe -h <<cachename>> -p 13000 (to connect to instance 0)
Redis-cli.exe -h <<cachename>> -p 13001 (to connect to instance 1)
Redis-cli.exe -h <<cachename>> -p 13002 (to connect to instance 2)
...
Redis-cli.exe -h <<cachename>> -p 1300N (to connect to instance N)
```

For ssl, replace `1300N` with `1500N`.

Can I configure clustering for a previously created cache?

Yes. First ensure that your cache is premium, by scaling if is not. Next, you should be able to see the cluster configuration options, including an option to enable cluster. You can change the cluster size after the cache is created, or after you have enabled clustering for the first time.

IMPORTANT

You can't undo enabling clustering. And a cache with clustering enabled and only one shard behaves *differently* than a cache of the same size with *no* clustering.

Can I configure clustering for a basic or standard cache?

Clustering is only available for premium caches.

Can I use clustering with the Redis ASP.NET Session State and Output Caching providers?

- **Redis Output Cache provider** - no changes required.
- **Redis Session State provider** - to use clustering, you must use [RedisSessionStateProvider](#) 2.0.1 or higher or an exception is thrown. This is a breaking change; for more information, see [v2.0.0 Breaking Change Details](#).

I am getting MOVE exceptions when using StackExchange.Redis and clustering, what should I do?

If you are using StackExchange.Redis and receive `MOVE` exceptions when using clustering, ensure that you are using [StackExchange.Redis 1.1.603](#) or later. For instructions on configuring your .NET applications to use StackExchange.Redis, see [Configure the cache clients](#).

Next steps

Learn how to use more premium cache features.

- [Introduction to the Azure Cache for Redis Premium tier](#)

Azure Cache for Redis FAQ

12/17/2019 • 28 minutes to read • [Edit Online](#)

Learn the answers to common questions, patterns, and best practices for Azure Cache for Redis.

What if my question isn't answered here?

If your question isn't listed here, let us know and we'll help you find an answer.

- You can post a question in the comments at the end of this FAQ and engage with the Azure Cache team and other community members about this article.
- To reach a wider audience, you can post a question on the [Azure Cache MSDN Forum](#) and engage with the Azure Cache team and other members of the community.
- If you want to make a feature request, you can submit your requests and ideas to [Azure Cache for Redis User Voice](#).
- You can also send an email to us at [Azure Cache External Feedback](#).

Azure Cache for Redis basics

The FAQs in this section cover some of the basics of Azure Cache for Redis.

- [What is Azure Cache for Redis?](#)
- [How can I get started with Azure Cache for Redis?](#)

The following FAQs cover basic concepts and questions about Azure Cache for Redis and are answered in one of the other FAQ sections.

- [What Azure Cache for Redis offering and size should I use?](#)
- [What Azure Cache for Redis clients can I use?](#)
- [Is there a local emulator for Azure Cache for Redis?](#)
- [How do I monitor the health and performance of my cache?](#)

Planning FAQs

- [What Azure Cache for Redis offering and size should I use?](#)
- [Azure Cache for Redis performance](#)
- [In what region should I locate my cache?](#)
- [How am I billed for Azure Cache for Redis?](#)
- [Can I use Azure Cache for Redis with Azure Government Cloud, Azure China Cloud, or Microsoft Azure Germany?](#)

Development FAQs

- [What do the StackExchange.Redis configuration options do?](#)
- [What Azure Cache for Redis clients can I use?](#)
- [Is there a local emulator for Azure Cache for Redis?](#)
- [How can I run Redis commands?](#)
- [Why doesn't Azure Cache for Redis have an MSDN class library reference like some of the other Azure services?](#)

- [Can I use Azure Cache for Redis as a PHP session cache?](#)
- [What are Redis databases?](#)

Security FAQs

- [When should I enable the non-SSL port for connecting to Redis?](#)

Production FAQs

- [What are some production best practices?](#)
- [What are some of the considerations when using common Redis commands?](#)
- [How can I benchmark and test the performance of my cache?](#)
- [Important details about ThreadPool growth](#)
- [Enable server GC to get more throughput on the client when using StackExchange.Redis](#)
- [Performance considerations around connections](#)

Monitoring and troubleshooting FAQs

The FAQs in this section cover common monitoring and troubleshooting questions. For more information about monitoring and troubleshooting your Azure Cache for Redis instances, see [How to monitor Azure Cache for Redis](#) and the various troubleshoot guides.

- [How do I monitor the health and performance of my cache?](#)
- [Why am I seeing timeouts?](#)
- [Why was my client disconnected from the cache?](#)

Prior Cache offering FAQs

- [Which Azure Cache offering is right for me?](#)

What is Azure Cache for Redis?

Azure Cache for Redis is based on the popular open-source software [Redis](#). It gives you access to a secure, dedicated Azure Cache for Redis, managed by Microsoft, and accessible from any application within Azure. For a more detailed overview, see the [Azure Cache for Redis](#) product page on Azure.com.

How can I get started with Azure Cache for Redis?

There are several ways you can get started with Azure Cache for Redis.

- You can check out one of our tutorials available for [.NET](#), [ASP.NET](#), [Java](#), [Node.js](#), and [Python](#).
- You can watch [How to Build High-Performance Apps Using Microsoft Azure Cache for Redis](#).
- You can check out the client documentation for the clients that match the development language of your project to see how to use Redis. There are many Redis clients that can be used with Azure Cache for Redis. For a list of Redis clients, see <https://redis.io/clients>.

If you don't already have an Azure account, you can:

- [Open an Azure account for free](#). You get credits that can be used to try out paid Azure services. Even after the credits are used up, you can keep the account and use free Azure services and features.
- [Activate Visual Studio subscriber benefits](#). Your MSDN subscription gives you credits every month that you can use for paid Azure services.

What Azure Cache for Redis offering and size should I use?

Each Azure Cache for Redis offering provides different levels of **size**, **bandwidth**, **high availability**, and **SLA** options.

The following are considerations for choosing a Cache offering.

- **Memory:** The Basic and Standard tiers offer 250 MB – 53 GB. The Premium tier offers up to 1.2 TB (as a cluster) or 120 GB (non-clustered). For more information, see [Azure Cache for Redis Pricing](#).
- **Network Performance:** If you have a workload that requires high throughput, the Premium tier offers more bandwidth compared to Standard or Basic. Also within each tier, larger size caches have more bandwidth because of the underlying VM that hosts the cache. For more information, see the [following table](#).
- **Throughput:** The Premium tier offers the maximum available throughput. If the cache server or client reaches the bandwidth limits, you may receive timeouts on the client side. For more information, see the following table.
- **High Availability/SLA:** Azure Cache for Redis guarantees that a Standard/Premium cache is available at least 99.9% of the time. To learn more about our SLA, see [Azure Cache for Redis Pricing](#). The SLA only covers connectivity to the Cache endpoints. The SLA does not cover protection from data loss. We recommend using the Redis data persistence feature in the Premium tier to increase resiliency against data loss.
- **Redis Data Persistence:** The Premium tier allows you to persist the cache data in an Azure Storage account. In a Basic/Standard cache, all the data is stored only in memory. Underlying infrastructure issues might result in potential data loss. We recommend using the Redis data persistence feature in the Premium tier to increase resiliency against data loss. Azure Cache for Redis offers RDB and AOF (coming soon) options in Redis persistence. For more information, see [How to configure persistence for a Premium Azure Cache for Redis](#).
- **Redis Cluster:** To create caches larger than 120 GB, or to shard data across multiple Redis nodes, you can use Redis clustering, which is available in the Premium tier. Each node consists of a primary/replica cache pair for high availability. For more information, see [How to configure clustering for a Premium Azure Cache for Redis](#).
- **Enhanced security and network isolation:** Azure Virtual Network (VNET) deployment provides enhanced security and isolation for your Azure Cache for Redis, as well as subnets, access control policies, and other features to further restrict access. For more information, see [How to configure Virtual Network support for a Premium Azure Cache for Redis](#).
- **Configure Redis:** In both the Standard and Premium tiers, you can configure Redis for Keyspace notifications.
- **Maximum number of client connections:** The Premium tier offers the maximum number of clients that can connect to Redis, with a higher number of connections for larger sized caches. Clustering does not increase the number of connections available for a clustered cache. For more information, see [Azure Cache for Redis pricing](#).
- **Dedicated Core for Redis Server:** In the Premium tier, all cache sizes have a dedicated core for Redis. In the Basic/Standard tiers, the C1 size and above have a dedicated core for Redis server.
- **Redis is single-threaded** so having more than two cores does not provide additional benefit over having just two cores, but larger VM sizes typically have more bandwidth than smaller sizes. If the cache server or client reaches the bandwidth limits, then you receive timeouts on the client side.
- **Performance improvements:** Caches in the Premium tier are deployed on hardware that has faster processors, giving better performance compared to the Basic or Standard tier. Premium tier Caches have higher throughput and lower latencies.

Azure Cache for Redis performance

The following table shows the maximum bandwidth values observed while testing various sizes of Standard and Premium caches using `redis-benchmark.exe` from an IaaS VM against the Azure Cache for Redis endpoint. For SSL throughput, `redis-benchmark` is used with `stunnel` to connect to the Azure Cache for Redis endpoint.

NOTE

These values are not guaranteed and there is no SLA for these numbers, but should be typical. You should load test your own application to determine the right cache size for your application. These numbers might change as we post newer results periodically.

From this table, we can draw the following conclusions:

- Throughput for the caches that are the same size is higher in the Premium tier as compared to the Standard tier.

For example, with a 6 GB Cache, throughput of P1 is 180,000 requests per second (RPS) as compared to 100,000 RPS for C3.

- With Redis clustering, throughput increases linearly as you increase the number of shards (nodes) in the cluster. For example, if you create a P4 cluster of 10 shards, then the available throughput is $400,000 * 10 = 4$ million RPS.
- Throughput for bigger key sizes is higher in the Premium tier as compared to the Standard Tier.

Pricing Tier	Size	CPU Cores	Available Bandwidth	1-KB Value Size	1-KB Value Size
Standard cache sizes			Megabits per sec (Mb/s) / Megabytes per sec (MB/s)	Requests per second (RPS) Non-SSL	Requests per second (RPS) SSL
C0	250 MB	Shared	100 / 12.5	15,000	7,500
C1	1 GB	1	500 / 62.5	38,000	20,720
C2	2.5 GB	2	500 / 62.5	41,000	37,000
C3	6 GB	4	1000 / 125	100,000	90,000
C4	13 GB	2	500 / 62.5	60,000	55,000
C5	26 GB	4	1,000 / 125	102,000	93,000
C6	53 GB	8	2,000 / 250	126,000	120,000
Premium cache sizes		CPU cores per shard	Megabits per sec (Mb/s) / Megabytes per sec (MB/s)	Requests per second (RPS) Non-SSL, per shard	Requests per second (RPS) SSL, per shard
P1	6 GB	2	1,500 / 187.5	180,000	172,000
P2	13 GB	4	3,000 / 375	350,000	341,000
P3	26 GB	4	3,000 / 375	350,000	341,000
P4	53 GB	8	6,000 / 750	400,000	373,000
P5	120 GB	20	6,000 / 750	400,000	373,000

For instructions on setting up stunnel or downloading the Redis tools such as `redis-benchmark.exe`, see the [How can I run Redis commands?](#) section.

In what region should I locate my cache?

For best performance and lowest latency, locate your Azure Cache for Redis in the same region as your cache client application.

How am I billed for Azure Cache for Redis?

Azure Cache for Redis pricing is [here](#). The pricing page lists pricing as an hourly rate. Caches are billed on a per-minute basis from the time that the cache is created until the time that a cache is deleted. There is no option for

stopping or pausing the billing of a cache.

Can I use Azure Cache for Redis with Azure Government Cloud, Azure China Cloud, or Microsoft Azure Germany?

Yes, Azure Cache for Redis is available in Azure Government Cloud, Azure China 21Vianet Cloud, and Microsoft Azure Germany. The URLs for accessing and managing Azure Cache for Redis are different in these clouds compared with Azure Public Cloud.

CLOUD	DNS SUFFIX FOR REDIS
Public	*.redis.cache.windows.net
US Gov	*.redis.cache.usgovcloudapi.net
Germany	*.redis.cache.cloudapi.de
China	*.redis.cache.chinacloudapi.cn

For more information on considerations when using Azure Cache for Redis with other clouds, see the following links.

- [Azure Government Databases - Azure Cache for Redis](#)
- [Azure China 21Vianet Cloud - Azure Cache for Redis](#)
- [Microsoft Azure Germany](#)

For information on using Azure Cache for Redis with PowerShell in Azure Government Cloud, Azure China 21Vianet Cloud, and Microsoft Azure Germany, see [How to connect to other clouds - Azure Cache for Redis PowerShell](#).

What do the StackExchange.Redis configuration options do?

StackExchange.Redis has many options. This section talks about some of the common settings. For more detailed information about StackExchange.Redis options, see [StackExchange.Redis configuration](#).

CONFIGURATIONOPTIONS	DESCRIPTION	RECOMMENDATION
AbortOnConnectFail	When set to true, the connection will not reconnect after a network failure.	Set to false and let StackExchange.Redis reconnect automatically.
ConnectRetry	The number of times to repeat connection attempts during initial connect.	See the following notes for guidance.
ConnectTimeout	Timeout in ms for connect operations.	See the following notes for guidance.

Usually the default values of the client are sufficient. You can fine-tune the options based on your workload.

• Retries

- For ConnectRetry and ConnectTimeout, the general guidance is to fail fast and retry again. This guidance is based on your workload and how much time on average it takes for your client to issue a Redis command and receive a response.
- Let StackExchange.Redis automatically reconnect instead of checking connection status and reconnecting yourself. **Avoid using the ConnectionMultiplexer.IsConnected property.**
- Snowballing - sometimes you may run into an issue where you are retrying and the retries snowball and never recovers. If snowballing occurs, you should consider using an exponential backoff retry algorithm

as described in [Retry general guidance](#) published by the Microsoft Patterns & Practices group.

- **Timeout values**

- Consider your workload and set the values accordingly. If you are storing large values, set the timeout to a higher value.
- Set `AbortOnConnectFail` to false and let StackExchange.Redis reconnect for you.
- Use a single `ConnectionMultiplexer` instance for the application. You can use a `LazyConnection` to create a single instance that is returned by a `Connection` property, as shown in [Connect to the cache using the ConnectionMultiplexer class](#).
- Set the `ConnectionMultiplexer.ClientName` property to an app instance unique name for diagnostic purposes.
- Use multiple `ConnectionMultiplexer` instances for custom workloads.
 - You can follow this model if you have varying load in your application. For example:
 - You can have one multiplexer for dealing with large keys.
 - You can have one multiplexer for dealing with small keys.
 - You can set different values for connection timeouts and retry logic for each `ConnectionMultiplexer` that you use.
 - Set the `ClientName` property on each multiplexer to help with diagnostics.
 - This guidance may lead to more streamlined latency per `ConnectionMultiplexer`.

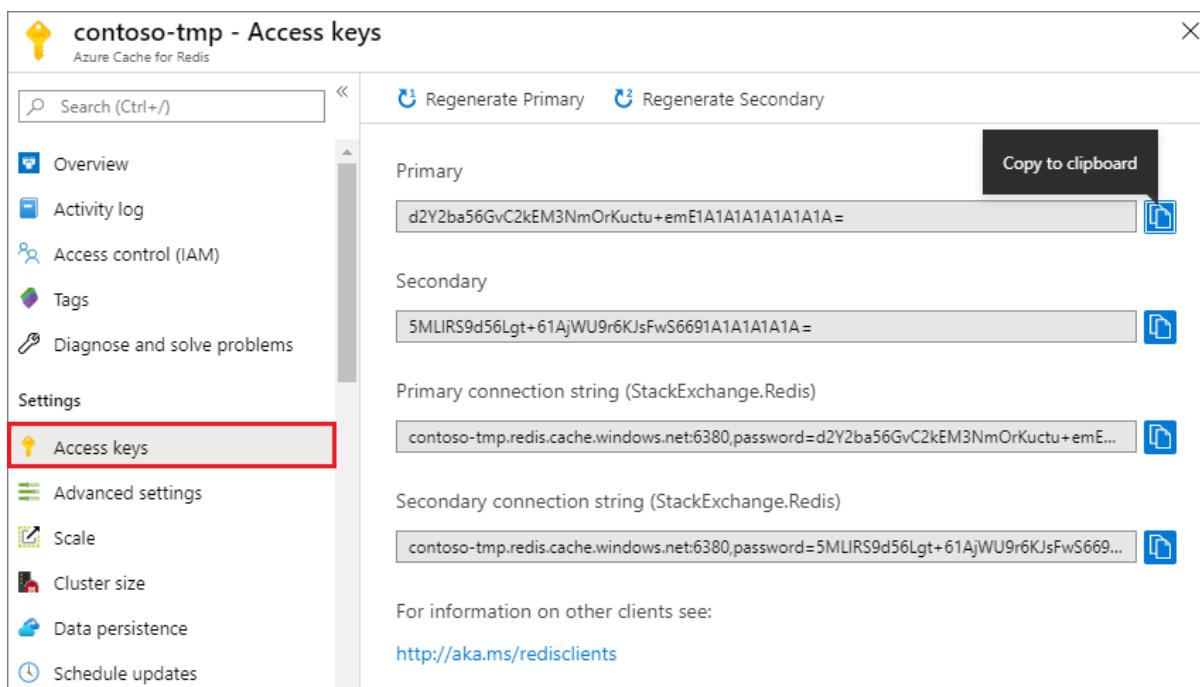
What Azure Cache for Redis clients can I use?

One of the great things about Redis is that there are many clients supporting many different development languages. For a current list of clients, see [Redis clients](#). For tutorials that cover several different languages and clients, see [How to use Azure Cache for Redis](#) and its sibling articles in the table of contents.

Retrieve host name, ports, and access keys from the Azure portal

To connect to an Azure Cache for Redis instance, cache clients need the host name, ports, and a key for the cache. Some clients might refer to these items by slightly different names. You can get the host name, ports, and keys from the [Azure portal](#).

- To get the access keys, from your cache left navigation, select **Access keys**.



- To get the host name and ports, from your cache left navigation, select **Properties**. The host name is of the form `<DNS name>.redis.cache.windows.net`.

The screenshot shows the 'Properties' page for an Azure Cache for Redis named 'contoso-tmp'. The left sidebar lists settings like Access keys, Advanced settings, Scale, Cluster size, Data persistence, Schedule updates, Geo-replication, Virtual Network, Firewall, and Properties. The 'Properties' item is highlighted with a red box. The main pane displays the following details:

Status	Running
Host name	contoso-tmp.redis.cache.windows.net
Non-SSL Port	Disabled
SSL Port	6380
Location	Central US

Copy to clipboard buttons are available for the Host name, Non-SSL Port, and SSL Port fields.

Is there a local emulator for Azure Cache for Redis?

There is no local emulator for Azure Cache for Redis, but you can run the MSOpenTech version of redis-server.exe from the [Redis command-line tools](#) on your local machine and connect to it to get a similar experience to a local cache emulator, as shown in the following example:

```
private static Lazy<ConnectionMultiplexer>
    lazyConnection = new Lazy<ConnectionMultiplexer>
    (() =>
{
    // Connect to a locally running instance of Redis to simulate a local cache emulator experience.
    return ConnectionMultiplexer.Connect("127.0.0.1:6379");
});

public static ConnectionMultiplexer Connection
{
    get
    {
        return lazyConnection.Value;
    }
}
```

You can optionally configure a [redis.conf](#) file to more closely match the [default cache settings](#) for your online Azure Cache for Redis if desired.

How can I run Redis commands?

You can use any of the commands listed at [Redis commands](#) except for the commands listed at [Redis commands not supported in Azure Cache for Redis](#). You have several options to run Redis commands.

- If you have a Standard or Premium cache, you can run Redis commands using the [Redis Console](#). The Redis console provides a secure way to run Redis commands in the Azure portal.
- You can also use the Redis command-line tools. To use them, perform the following steps:
 - Download the [Redis command-line tools](#).
 - Connect to the cache using `redis-cli.exe`. Pass in the cache endpoint using the `-h` switch and the key using `-a` as shown in the following example:
 - `redis-cli -h <Azure Cache for Redis name>.redis.cache.windows.net -a <key>`

NOTE

The Redis command-line tools do not work with the SSL port, but you can use a utility such as `stunnel` to securely connect the tools to the SSL port by following the directions in the [How to use the Redis command-line tool with Azure Cache for Redis](#) article.

Why doesn't Azure Cache for Redis have an MSDN class library reference like some of the other Azure services?

Microsoft Azure Cache for Redis is based on the popular open-source Azure Cache for Redis. It can be accessed by a wide variety of [Redis clients](#) for many programming languages. Each client has its own API that makes calls to the Azure Cache for Redis instance using [Redis commands](#).

Because each client is different, there is not one centralized class reference on MSDN, and each client maintains its own reference documentation. In addition to the reference documentation, there are several tutorials showing how to get started with Azure Cache for Redis using different languages and cache clients. To access these tutorials, see [How to use Azure Cache for Redis](#) and its sibling articles in the table of contents.

Can I use Azure Cache for Redis as a PHP session cache?

Yes, to use Azure Cache for Redis as a PHP session cache, specify the connection string to your Azure Cache for Redis instance in `session.save_path`.

IMPORTANT

When using Azure Cache for Redis as a PHP session cache, you must URL encode the security key used to connect to the cache, as shown in the following example:

```
session.save_path = "tcp://mycache.redis.cache.windows.net:6379?auth=<url encoded primary or secondary key here>;"
```

If the key is not URL encoded, you may receive an exception with a message like: `Failed to parse session.save_path`

For more information about using Azure Cache for Redis as a PHP session cache with the `PhpRedis` client, see [PHP Session handler](#).

What are Redis databases?

Redis Databases are just a logical separation of data within the same Redis instance. The cache memory is shared between all the databases and actual memory consumption of a given database depends on the keys/values stored in that database. For example, a C6 cache has 53 GB of memory, and a P5 has 120 GB. You can choose to put all 53 GB / 120 GB into one database or you can split it up between multiple databases.

NOTE

When using a Premium Azure Cache for Redis with clustering enabled, only database 0 is available. This limitation is an intrinsic Redis limitation and is not specific to Azure Cache for Redis. For more information, see [Do I need to make any changes to my client application to use clustering?](#)

When should I enable the non-SSL port for connecting to Redis?

Redis server does not natively support SSL, but Azure Cache for Redis does. If you are connecting to Azure Cache for Redis and your client supports SSL, like `StackExchange.Redis`, then you should use SSL.

NOTE

The non-SSL port is disabled by default for new Azure Cache for Redis instances. If your client does not support SSL, then you must enable the non-SSL port by following the directions in the [Access ports](#) section of the [Configure a cache in Azure Cache for Redis](#) article.

Redis tools such as `redis-cli` do not work with the SSL port, but you can use a utility such as `stunnel` to securely connect the tools to the SSL port by following the directions in the [Announcing ASP.NET Session State Provider for Redis Preview Release](#) blog post.

For instructions on downloading the Redis tools, see the [How can I run Redis commands?](#) section.

What are some production best practices?

- [StackExchange.Redis best practices](#)
- [Configuration and concepts](#)
- [Performance testing](#)

StackExchange.Redis best practices

- Set `AbortConnect` to false, then let the `ConnectionMultiplexer` reconnect automatically. [See here for details.](#)
- Reuse the `ConnectionMultiplexer` - do not create a new one for each request. The `Lazy<ConnectionMultiplexer>` pattern [shown here](#) is recommended.
- Redis works best with smaller values, so consider chopping up bigger data into multiple keys. In [this Redis discussion](#), 100 kb is considered large. Read [this article](#) for an example problem that can be caused by large values.
- Configure your [ThreadPool settings](#) to avoid timeouts.
- Use at least the default `connectTimeout` of 5 seconds. This interval gives StackExchange.Redis sufficient time to re-establish the connection in the event of a network blip.
- Be aware of the performance costs associated with different operations you are running. For instance, the `KEYS` command is an O(n) operation and should be avoided. The [redis.io site](#) has details around the time complexity for each operation that it supports. Click each command to see the complexity for each operation.

Configuration and concepts

- Use Standard or Premium Tier for Production systems. The Basic Tier is a single node system with no data replication and no SLA. Also, use at least a C1 cache. C0 caches are typically used for simple dev/test scenarios.
- Remember that Redis is an **In-Memory** data store. Read [this article](#) so that you are aware of scenarios where data loss can occur.
- Develop your system such that it can handle connection blips [due to patching and failover](#).

Performance testing

- Start by using `redis-benchmark.exe` to get a feel for possible throughput before writing your own perf tests. Because `redis-benchmark` does not support SSL, you must [enable the Non-SSL port through the Azure portal](#) before you run the test. For examples, see [How can I benchmark and test the performance of my cache?](#)
- The client VM used for testing should be in the same region as your Azure Cache for Redis instance.
- We recommend using Dv2 VM Series for your client as they have better hardware and should give the best results.
- Make sure your client VM you choose has at least as much computing and bandwidth capability as the cache you are testing.
- Enable VRSS on the client machine if you are on Windows. [See here for details.](#)
- Premium tier Redis instances have better network latency and throughput because they are running on better hardware for both CPU and Network.

What are some of the considerations when using common Redis commands?

- Avoid using certain Redis commands that take a long time to complete, unless you fully understand the impact of these commands. For example, do not run the [KEYS](#) command in production. Depending on the number of keys, it could take a long time to return. Redis is a single-threaded server and it processes commands one at a time. If you have other commands issued after KEYS, they will not be processed until Redis processes the KEYS command. The [redis.io site](#) has details around the time complexity for each operation that it supports. Click each command to see the complexity for each operation.
- Key sizes - should I use small key/values or large key/values? It depends on the scenario. If your scenario requires larger keys, you can adjust the ConnectionTimeout, then retry values and adjust your retry logic. From a Redis server perspective, smaller values give better performance.
- These considerations don't mean that you can't store larger values in Redis; you must be aware of the following considerations. Latencies will be higher. If you have one set of data that is larger and one that is smaller, you can use multiple ConnectionMultiplexer instances, each configured with a different set of timeout and retry values, as described in the previous [What do the StackExchange.Redis configuration options do](#) section.

How can I benchmark and test the performance of my cache?

- [Enable cache diagnostics](#) so you can [monitor](#) the health of your cache. You can view the metrics in the Azure portal and you can also [download and review](#) them using the tools of your choice.
- You can use redis-benchmark.exe to load test your Redis server.
- Ensure that the load testing client and the Azure Cache for Redis are in the same region.
- Use redis-cli.exe and monitor the cache using the INFO command.
- If your load is causing high memory fragmentation, you should scale up to a larger cache size.
- For instructions on downloading the Redis tools, see the [How can I run Redis commands?](#) section.

The following commands provide an example of using redis-benchmark.exe. For accurate results, run these commands from a VM in the same region as your cache.

- Test Pipelined SET requests using a 1k payload

```
redis-benchmark.exe -h **yourcache**.redis.cache.windows.net -a **yourAccesskey** -t SET -n 1000000 -d 1024 -P 50
```

- Test Pipelined GET requests using a 1k payload. NOTE: Run the SET test shown above first to populate cache

```
redis-benchmark.exe -h **yourcache**.redis.cache.windows.net -a **yourAccesskey** -t GET -n 1000000 -d 1024 -P 50
```

Important details about ThreadPool growth

The CLR ThreadPool has two types of threads - "Worker" and "I/O Completion Port" (IOCP) threads.

- Worker threads are used for things like processing the `Task.Run(...)`, or `ThreadPool.QueueUserWorkItem(...)` methods. These threads are also used by various components in the CLR when work needs to happen on a background thread.
- IOCP threads are used when asynchronous IO happens, such as when reading from the network.

The thread pool provides new worker threads or I/O completion threads on demand (without any throttling) until it reaches the "Minimum" setting for each type of thread. By default, the minimum number of threads is set to the number of processors on a system.

Once the number of existing (busy) threads hits the "minimum" number of threads, the ThreadPool will throttle the rate at which it injects new threads to one thread per 500 milliseconds. Typically, if your system gets a burst of work needing an IOCP thread, it will process that work quickly. However, if the burst of work is more than the configured "Minimum" setting, there will be some delay in processing some of the work as the ThreadPool waits for one of two things to happen.

1. An existing thread becomes free to process the work.

- No existing thread becomes free for 500 ms, so a new thread is created.

Basically, it means that when the number of Busy threads is greater than Min threads, you are likely paying a 500-ms delay before network traffic is processed by the application. Also, it is important to note that when an existing thread stays idle for longer than 15 seconds (based on what I remember), it will be cleaned up and this cycle of growth and shrinkage can repeat.

If we look at an example error message from StackExchange.Redis (build 1.0.450 or later), you will see that it now prints ThreadPool statistics (see IOCP and WORKER details below).

```
System.TimeoutException: Timeout performing GET MyKey, inst: 2, mgr: Inactive,
queue: 6, qu: 0, qs: 6, qc: 0, wr: 0, wq: 0, in: 0, ar: 0,
IOCP: (Busy=6,Free=994,Min=4,Max=1000),
WORKER: (Busy=3,Free=997,Min=4,Max=1000)
```

In the previous example, you can see that for IOCP thread there are six busy threads and the system is configured to allow four minimum threads. In this case, the client would have likely seen two 500-ms delays, because $6 > 4$.

Note that StackExchange.Redis can hit timeouts if growth of either IOCP or WORKER threads gets throttled.

Recommendation

Given this information, we strongly recommend that customers set the minimum configuration value for IOCP and WORKER threads to something larger than the default value. We can't give one-size-fits-all guidance on what this value should be because the right value for one application will likely be too high or low for another application. This setting can also impact the performance of other parts of complicated applications, so each customer needs to fine-tune this setting to their specific needs. A good starting place is 200 or 300, then test and tweak as needed.

How to configure this setting:

- We recommend changing this setting programmatically by using the [ThreadPool.SetMinThreads \(...\)](#) method in `global.asax.cs`. For example:

```
private readonly int minThreads = 200;
void Application_Start(object sender, EventArgs e)
{
    // Code that runs on application startup
    AreaRegistration.RegisterAllAreas();
    RouteConfig.RegisterRoutes(RouteTable.Routes);
    BundleConfig.RegisterBundles(BundleTable.Bundles);
    ThreadPool.SetMinThreads(minThreads, minThreads);
}
```

NOTE

The value specified by this method is a global setting, affecting the whole AppDomain. For example, if you have a 4-core machine and want to set `minWorkerThreads` and `minIoThreads` to 50 per CPU during run-time, you would use `ThreadPool.SetMinThreads(200, 200)`.

- It is also possible to specify the minimum threads setting by using the [minIoThreads or minWorkerThreads configuration setting](#) under the `<processModel>` configuration element in `Machine.config`, usually located at `%SystemRoot%\Microsoft.NET\Framework\[versionNumber]\CONFIG\`. **Setting the number of minimum threads in this way is generally not recommended, because it is a System-wide setting.**

NOTE

The value specified in this configuration element is a *per-core* setting. For example, if you have a 4-core machine and want your *minIoThreads* setting to be 200 at runtime, you would use `<processModel minIoThreads="50"/>`.

Enable server GC to get more throughput on the client when using StackExchange.Redis

Enabling server GC can optimize the client and provide better performance and throughput when using StackExchange.Redis. For more information on server GC and how to enable it, see the following articles:

- [To enable server GC](#)
- [Fundamentals of Garbage Collection](#)
- [Garbage Collection and Performance](#)

Performance considerations around connections

Each pricing tier has different limits for client connections, memory, and bandwidth. While each size of cache allows *up to* a certain number of connections, each connection to Redis has overhead associated with it. An example of such overhead would be CPU and memory usage as a result of TLS/SSL encryption. The maximum connection limit for a given cache size assumes a lightly loaded cache. If load from connection overhead *plus* load from client operations exceeds capacity for the system, the cache can experience capacity issues even if you have not exceeded the connection limit for the current cache size.

For more information about the different connections limits for each tier, see [Azure Cache for Redis pricing](#). For more information about connections and other default configurations, see [Default Redis server configuration](#).

How do I monitor the health and performance of my cache?

Microsoft Azure Cache for Redis instances can be monitored in the [Azure portal](#). You can view metrics, pin metrics charts to the Startboard, customize the date and time range of monitoring charts, add and remove metrics from the charts, and set alerts when certain conditions are met. For more information, see [Monitor Azure Cache for Redis](#).

The Azure Cache for Redis **Resource menu** also contains several tools for monitoring and troubleshooting your caches.

- **Diagnose and solve problems** provides information about common issues and strategies for resolving them.
- **Resource health** watches your resource and tells you if it's running as expected. For more information about the Azure Resource health service, see [Azure Resource health overview](#).
- **New support request** provides options to open a support request for your cache.

These tools enable you to monitor the health of your Azure Cache for Redis instances and help you manage your caching applications. For more information, see the "Support & troubleshooting settings" section of [How to configure Azure Cache for Redis](#).

Why am I seeing timeouts?

Timeouts happen in the client that you use to talk to Redis. When a command is sent to the Redis server, the command is queued up and Redis server eventually picks up the command and executes it. However the client can time out during this process and if it does an exception is raised on the calling side. For more information on troubleshooting timeout issues, see [client-side troubleshooting](#) and [StackExchange.Redis timeout exceptions](#).

Why was my client disconnected from the cache?

The following are some common reason for a cache disconnect.

- Client-side causes
 - The client application was redeployed.
 - The client application performed a scaling operation.
 - In the case of Cloud Services or Web Apps, this may be due to autoscaling.

- The networking layer on the client side changed.
 - Transient errors occurred in the client or in the network nodes between the client and the server.
 - The bandwidth threshold limits were reached.
 - CPU bound operations took too long to complete.
- Server-side causes
 - On the standard cache offering, the Azure Cache for Redis service initiated a fail-over from the primary node to the secondary node.
 - Azure was patching the instance where the cache was deployed
 - This can be for Redis server updates or general VM maintenance.

Which Azure Cache offering is right for me?

IMPORTANT

As per last year's [announcement](#), Azure Managed Cache Service and Azure In-Role Cache service **have been retired** on November 30, 2016. Our recommendation is to use [Azure Cache for Redis](#). For information on migrating, see [Migrate from Managed Cache Service to Azure Cache for Redis](#).

Azure Cache for Redis

Azure Cache for Redis is Generally Available in sizes up to 120 GB and has an availability SLA of 99.9%. The new [premium tier](#) offers sizes up to 1.2 TB and support for clustering, VNET, and persistence, with a 99.9% SLA.

Azure Cache for Redis gives customers the ability to use a secure, dedicated Azure Cache for Redis, managed by Microsoft. With this offer, you get to leverage the rich feature set and ecosystem provided by Redis, and reliable hosting and monitoring from Microsoft.

Unlike traditional caches that deal only with key-value pairs, Redis is popular for its highly performant data types. Redis also supports running atomic operations on these types, like appending to a string; incrementing the value in a hash; pushing to a list; computing set intersection, union and difference; or getting the member with highest ranking in a sorted set. Other features include support for transactions, pub/sub, Lua scripting, keys with a limited time-to-live, and configuration settings to make Redis behave more like a traditional cache.

Another key aspect to Redis success is the healthy, vibrant open- source ecosystem built around it. This is reflected in the diverse set of Redis clients available across multiple languages. This ecosystem and wide range of clients allow Azure Cache for Redis to be used by nearly any workload you would build inside of Azure.

For more information about getting started with Azure Cache for Redis, see [How to Use Azure Cache for Redis](#) and [Azure Cache for Redis documentation](#).

Managed Cache service

[Managed Cache service was retired November 30, 2016](#).

To view archived documentation, see [Archived Managed Cache Service Documentation](#).

In-Role Cache

[In-Role Cache was retired November 30, 2016](#).

To view archived documentation, see [Archived In-Role Cache Documentation](#).