How to Use Azure Redis Cache

This guide shows you how to get started using Azure Redis Cache. Microsoft Azure Redis Cache is based on the popular open source Redis Cache. It gives you access to a secure, dedicated Redis cache, managed by Microsoft. A cache created using Azure Redis Cache is accessible from any application within Microsoft Azure.

Microsoft Azure Redis Cache 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 530 GB. 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. For information on pricing, see Cache Pricing Details.

This guide shows you how to use the StackExchange.Redis client using C# code. The scenarios covered include creating and configuring a cache, configuring cache clients, and adding and removing objects from the cache. For more information on using Azure Redis Cache, see Next Steps. For a step-by-step tutorial of building an ASP.NET MVC web app with Redis Cache, see How to create a Web App with Redis Cache.

Get Started with Azure Redis Cache

Getting started with Azure Redis Cache is easy. To get started, you provision and configure a cache. Next, you configure the cache clients so they can access the cache. Once the cache clients are configured, you can begin working with them.

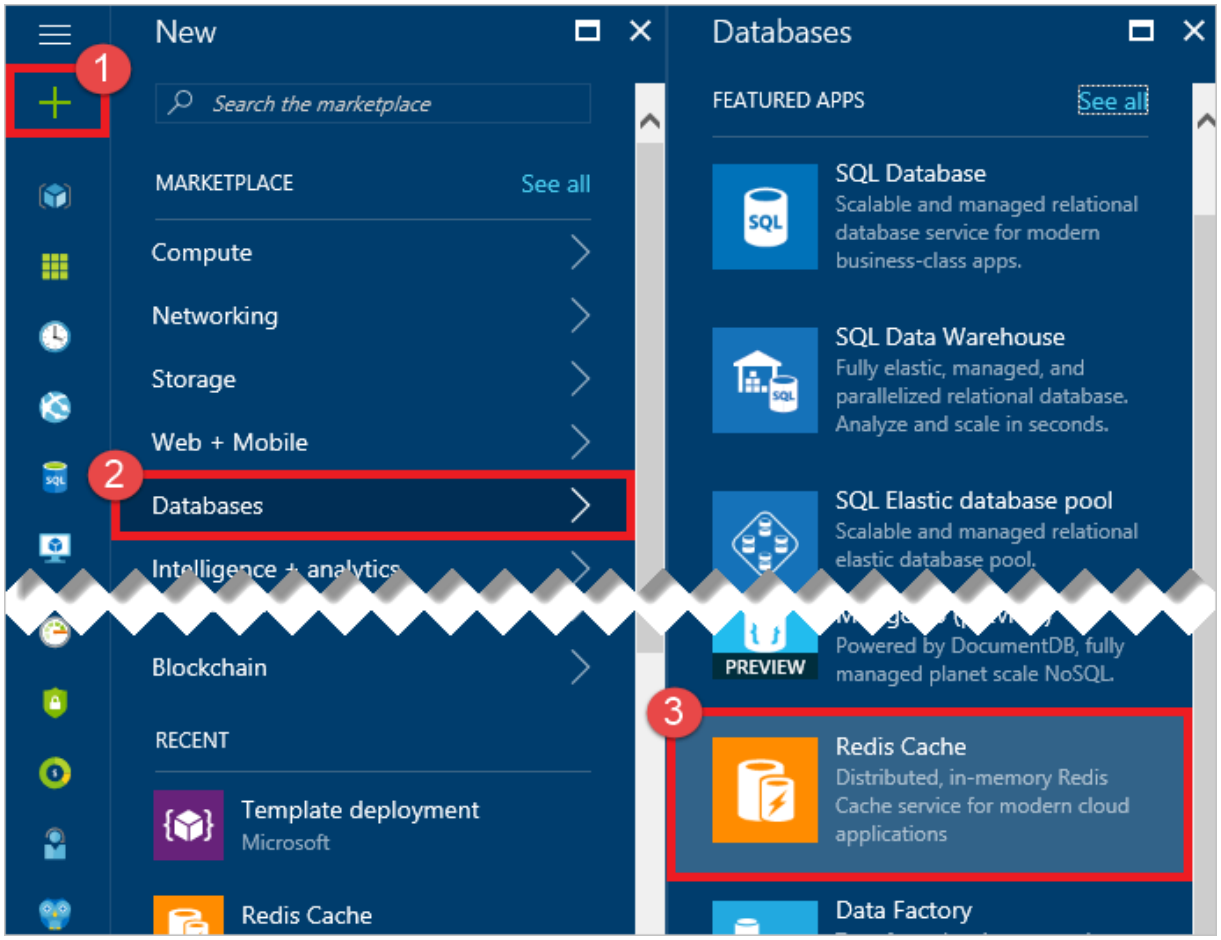
- Create the cache
- Configure the cache clients

Create a cache

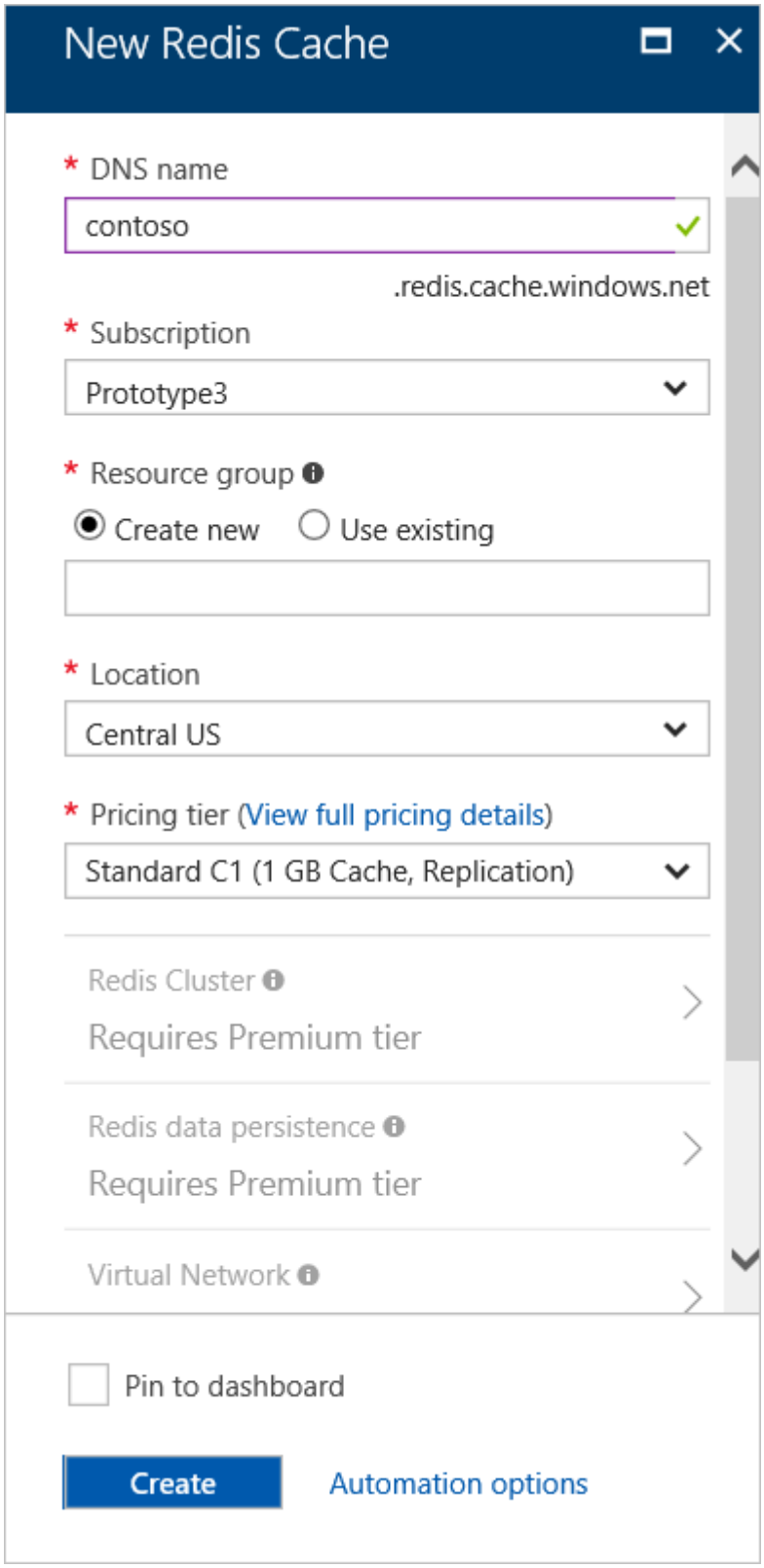
To create a cache, first sign in to the Azure portal, and click New > Databases > Redis Cache.

Note

If you don't have an Azure account, you can Open an Azure account for free in just a couple of minutes.



In the New Redis Cache blade, specify the desired configuration for the cache.

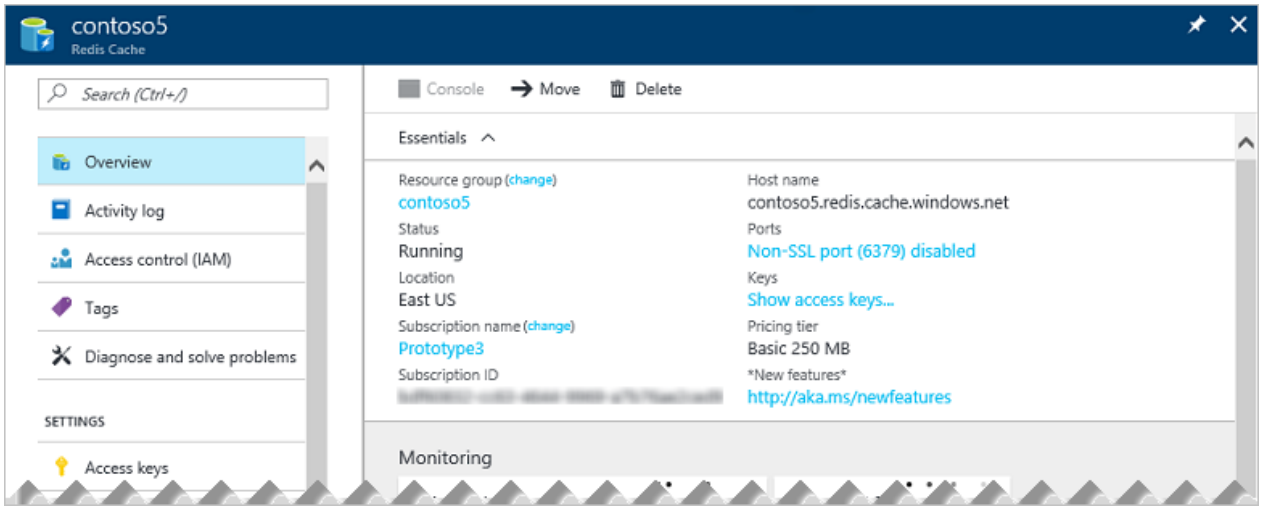


- In Dns name, enter a unique cache name to use for the cache endpoint. The cache name must be a string between 1 and 63 characters and contain only

numbers, letters, and the `-` character. The cache name cannot start or end with the `-` character, and consecutive `-` characters are not valid.

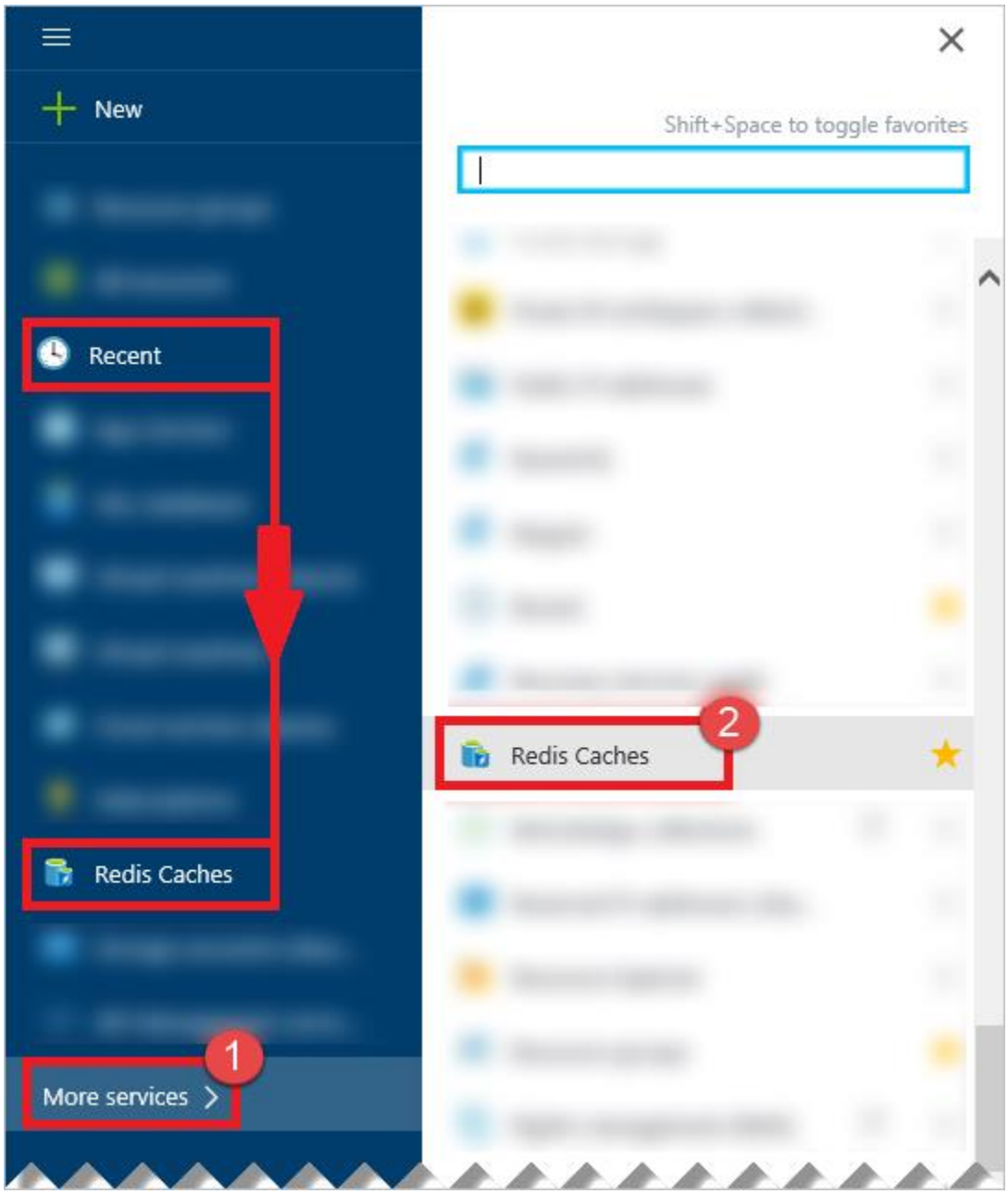
- For **Subscription**, select the Azure subscription that you want to use for the cache. If your account has only one subscription, it will be automatically selected and the **Subscription** drop-down will not be displayed.
- In **Resource group**, select or create a resource group for your cache. For more information, see Using Resource groups to manage your Azure resources.
- Use **Location** to specify the geographic location in which your cache is hosted. For the best performance, Microsoft strongly recommends that you create the cache in the same region as the cache client application.
- Use **Pricing tier** to select the desired cache size and features.
- **Redis cluster** allows you to create caches larger than 53 GB and to shard data across multiple Redis nodes. For more information, see How to configure clustering for a Premium Azure Redis Cache.
- **Redis persistence** offers the ability to persist your cache to an Azure Storage account. For instructions on configuring persistence, see How to configure persistence for a Premium Azure Redis Cache.
- **Virtual Network** provides enhanced security and isolation by restricting access to your cache to only those clients within the specified Azure Virtual Network. You can use all the features of VNet such as subnets, access control policies, and other features to further restrict access to Redis. For more information, see How to configure Virtual Network support for a Premium Azure Redis Cache.
- By default, non-SSL access is disabled for new caches. To enable the non-SSL port, check **Unblock port 6379 (not SSL encrypted)**.

Once the new cache options are configured, click **Create**. It can take a few minutes for the cache to be created. To check the status, you can monitor the progress on the startboard. After the cache has been created, your new cache has a **Running** status and is ready for use with default settings.



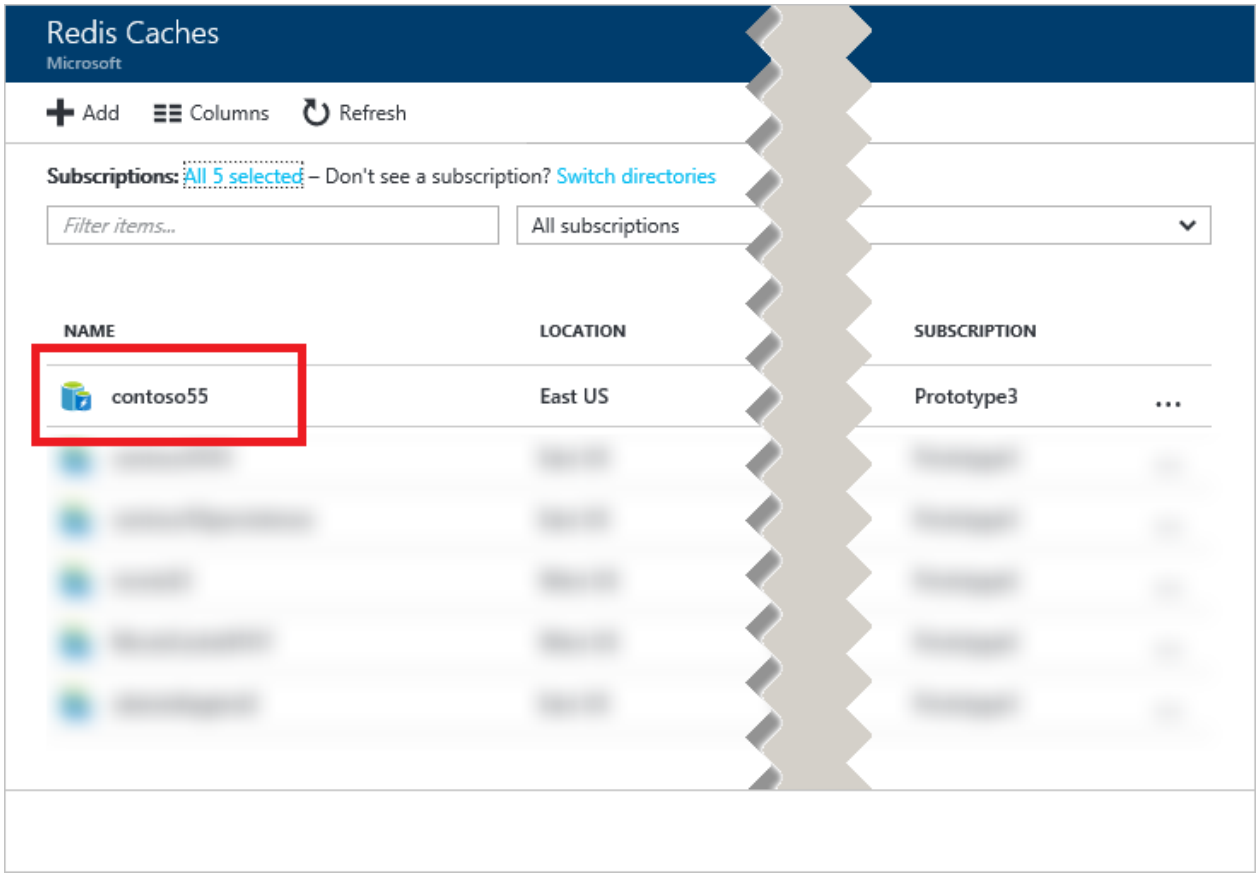
To access your cache after it's created

Caches can be accessed in the Azure portal using the Browse blade.

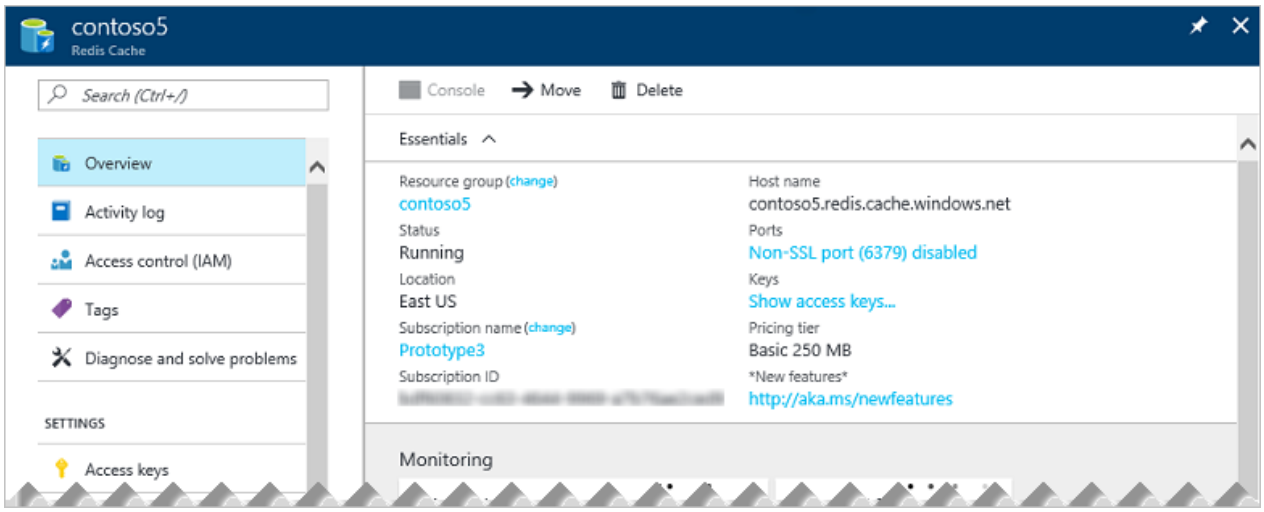


To view your caches, click More services > Redis Caches. If you have recently browsed to a Redis Cache, you can click Redis Caches directly from the list without clicking More services.

Select the desired cache to view and configure the settings for that cache.



You can view and configure your cache from the Redis Cache blade.



For more information about configuring your cache, see How to configure Azure Redis Cache.

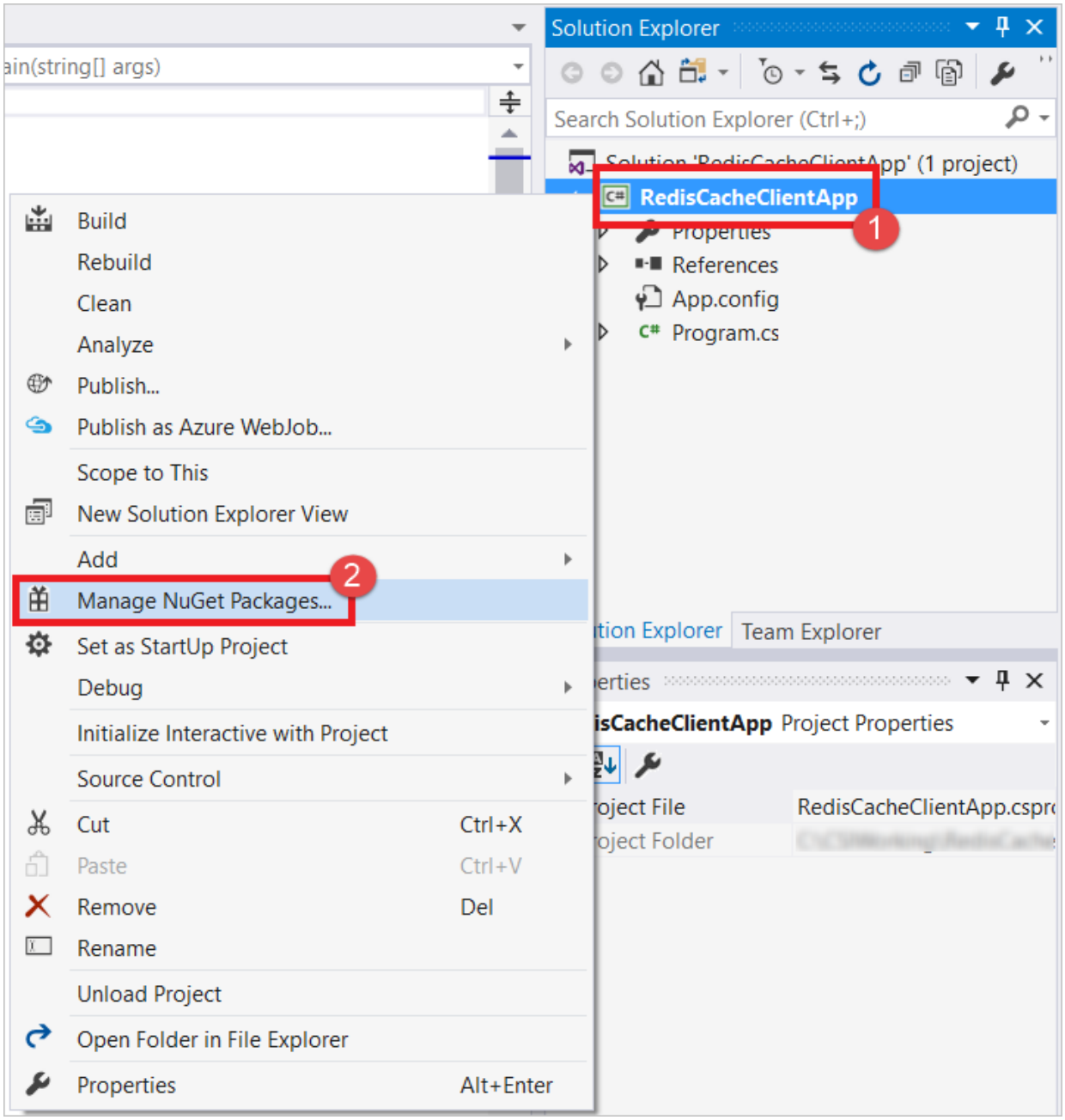
Configure the cache clients

.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.Redis cache client documentation.

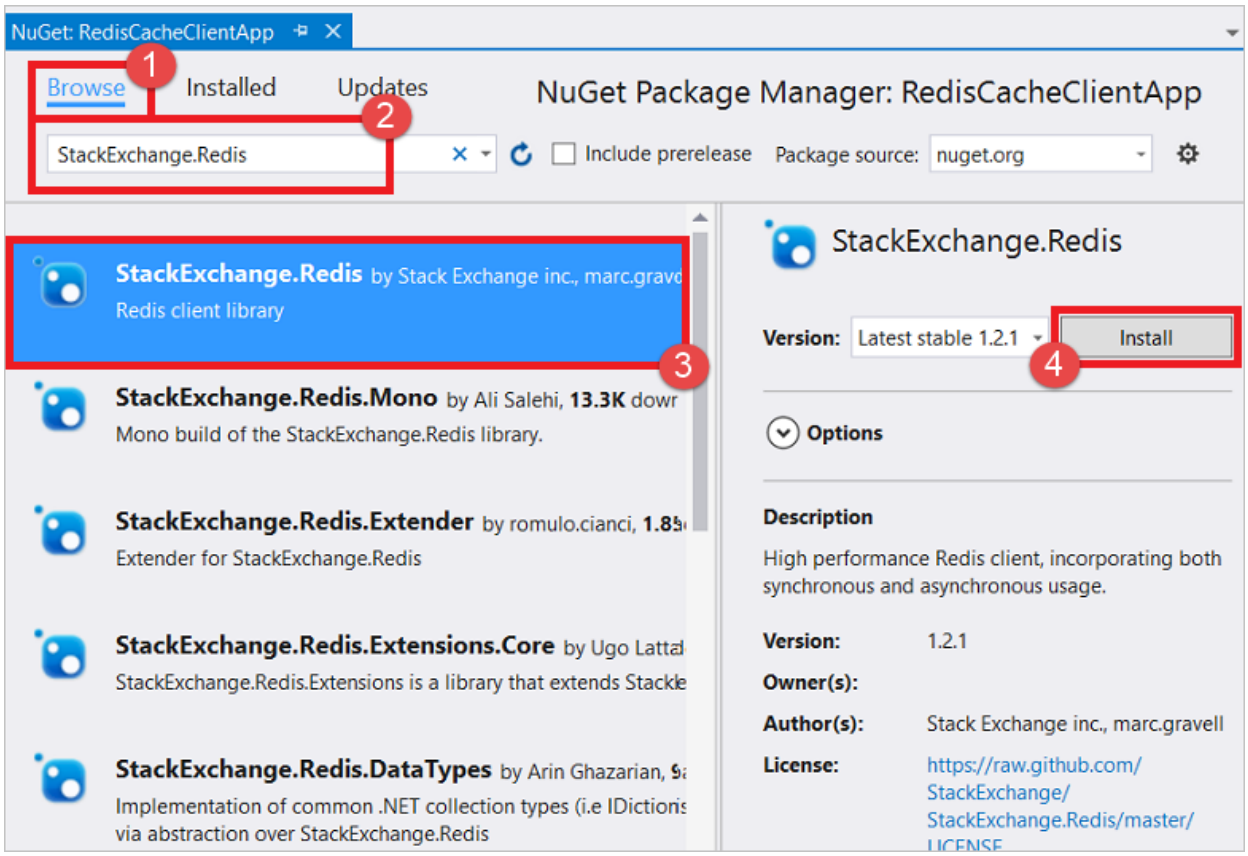
To configure a client application in Visual Studio using the StackExchange.Redis NuGet package, right-click the project in Solution Explorer and choose Manage NuGet Packages.



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 Redis Cache with the StackExchange.Redis cache 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 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
```

Once your client project is configured for caching, you can use the techniques described in the following sections for working with your cache.

Working with Caches

The steps in this section describe how to perform common tasks with Cache.

- Connect to the cache
- Add and retrieve objects from the cache
- Work with .NET objects in the cache

Connect to the cache

To programmatically work with a cache, you need a reference to the cache. Add the following to the top of any file from which you want to use the StackExchange.Redis client to access an Azure Redis Cache.

```
using StackExchange.Redis;
```

Note

The StackExchange.Redis client requires .NET Framework 4 or higher.

The connection to the Azure Redis Cache is managed by the `ConnectionMultiplexer` class. This class should be shared and reused throughout your client application, and does not need to be created on a per operation basis.

To connect to an Azure Redis Cache and be returned an instance of a connected `ConnectionMultiplexer`, call the static `Connect` method and pass in the cache endpoint and key. Use the key generated from the Azure portal as the password parameter.

```
ConnectionMultiplexer connection =
ConnectionMultiplexer.Connect("contoso5.redis.cache.windows.net,abortConnect=false
,ssl=true,password=...");
```

Important

Warning: Never store credentials in source code. To keep this sample simple, I'm showing them in the source code. See How Application Strings and Connection Strings Work for information on how to store credentials.

If you don't want to use SSL, either set `ssl=false` or omit the `ssl` parameter.

Note

The non-SSL port is disabled by default for new caches. For instructions on enabling the non-SSL port, see Access Ports.

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 only a single connected `ConnectionMultiplexer` instance. In these examples `abortConnect` is set to false, which means that the call succeeds even if a connection to the Azure Redis Cache 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.

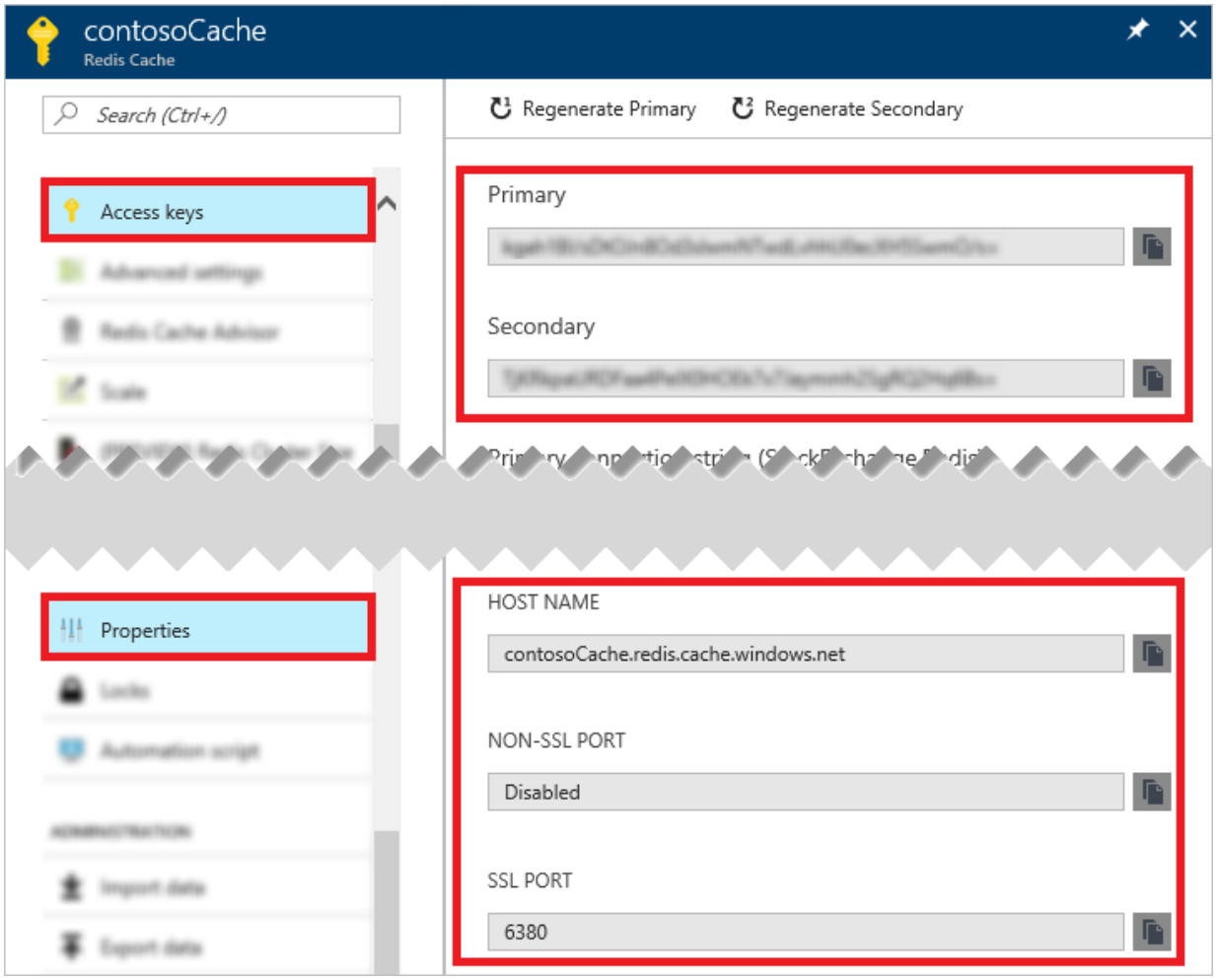
```
private static Lazy<ConnectionMultiplexer> lazyConnection = new
Lazy<ConnectionMultiplexer>(() =>
{
    return
ConnectionMultiplexer.Connect("contoso5.redis.cache.windows.net,abortConnect=false
,ssl=true,password=...");
});

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

For more information on advanced connection configuration options, see StackExchange.Redis configuration model.

To connect to an Azure Redis Cache instance, cache clients need the host name, ports, and keys of the cache. Some clients may refer to these items by slightly different names. You can retrieve this information in the Azure portal or by using command-line tools such as Azure CLI.

Retrieve host name, ports, and access keys using the Azure Portal

To retrieve host name, ports, and access keys using the Azure Portal, browse to your cache in the Azure portal and click Access keys and Properties in the Resource menu.



Once the connection is established, return a reference to the redis cache 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.

```
// Connection refers to a property that returns a ConnectionMultiplexer
// as shown in the previous example.
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");
```

Azure Redis caches have a configurable number of databases (default of 16) that can be used to logically separate the data within a Redis cache. For more information, see What are Redis databases? and Default Redis server configuration.

Now that you know how to connect to an Azure Redis Cache instance and return a reference to the cache database, let's look at working with the cache.

Add and retrieve objects from the cache

Items can be stored in and retrieved from a cache by using the `StringSet` and `StringGet` methods.

```
// If key1 exists, it is overwritten.
cache.StringSet("key1", "value1");

string value = cache.StringGet("key1");
```

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.

When calling `StringGet`, if the object exists, it is returned, and if it does not, `null` is returned. If `null` is returned, you can retrieve the value from the desired data source and store it in the cache for subsequent use. This usage pattern is known as the cache-aside pattern.

```
string value = cache.StringGet("key1");
if (value == null)
{
    // The item keyed by "key1" is not in the cache. Obtain
    // it from the desired data source and add it to the cache.
    value = GetValueFromDataSource();

    cache.StringSet("key1", value);
}
```

You can also use `RedisValue`, as shown in the following example. `RedisValue` has implicit operators for working with integral data types, and can be useful if `null` is an expected value for a cached item.

```
RedisValue value = cache.StringGet("key1");
if (!value.HasValue)
{
    value = GetValueFromDataSource();
    cache.StringSet("key1", value);
}
```

To specify the expiration of an item in the cache, use the `TimeSpan` parameter of `StringSet`.

```
cache.StringSet("key1", "value1", TimeSpan.FromMinutes(90));
```

Work with .NET objects in the cache

Azure Redis Cache 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.NET and serialize to and from JSON. The following example shows a get and set using an `Employee` object instance.

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

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

// Store to cache
cache.StringSet("e25", JsonConvert.SerializeObject(new Employee(25, "Clayton
Gragg")));

// Retrieve from cache
Employee e25 = JsonConvert.DeserializeObject<Employee>(cache.StringGet("e25"));
```