|  |
| --- |
| Photo displaying partial image of two pie charts on a canvas-textured page |
| Unified Redis Platform (URP)  Field Experience Platform |
| |  |  |  | | --- | --- | --- | | Pratik Bhattacharya | 4/23/20 | https://aka.ms/fxpfocs | |

Contents

[Introduction 2](#_Toc39027993)

[Capabilities 3](#_Toc39027994)

[Architecture and Concepts 4](#_Toc39027995)

[Key Isolation 4](#_Toc39027996)

[Hierarchy Cluster 4](#_Toc39027997)

[Geo Replication 4](#_Toc39027998)

[System Design 5](#_Toc39027999)

[Components 5](#_Toc39028000)

[Usage 7](#_Toc39028001)

[Connecting to Cluster 7](#_Toc39028002)

[Creating Redis Database 7](#_Toc39028003)

[Other Connection Multiplexer level operations 7](#_Toc39028004)

[Registration Steps (On-boarding) 9](#_Toc39028005)

[Management Console 11](#_Toc39028006)

[Appendix – 1: Code and Samples 12](#_Toc39028007)

# Introduction

Upon conducting a cost analysis of our Azure Subscriptions, it was seen that Azure Caches for Redis were one of the highest sources of expenditure. It was seen that each service was using its own isolated Redis Cache for each environment. Even with aggressive environment consolidation it was becoming an issue to maintain and pay for all the Redis caches. Furthermore, to satisfy BCDR (Business Continuity Disaster Recovery) multiple Redis Caches were created for the same service in different regions. More analysis also revealed that most of the Caches were not used up to their full potential, both in terms of Storage as well Connection loads. A Premium P1 Redis cache allows storage up to 6 GB but none of our services were storing more than 1 GB of data. A P1 Redis Cache costs roughly $412 per month and with 3 services and each service having 2 caches one would end up spending $2,412 per month.

We came up with the idea of consolidating these Redis Caches to a single unit (per region) to reduce the maintenance overhead and reduce expenditure. We developed the Unified Redis Platform (URP) which can consolidate multiple Redis caches to a single cluster. It provides the isolation required by multiple apps to operate (get/edit/delete/update the cached objects) on the same cluster without interfering with each other.

# Capabilities

URP Capabilities include:

1. Consolidate multiple Azure Cache for Redis instances to a single cluster
2. Automatic key name-spacing for providing isolation between multiple applications
3. Disaster Recovery by keeping Azure Redis Cache in multiple regions in a single cluster
4. Multiple read-regions
5. Geo-replication of data (add/delete/update) across multiple regions
6. Support for all operations available in StackExchange.Redis library. All methods and data types from StackExchange.Redis can be re-used without any code change.
7. Support for auto-retires and timeouts in case of errors
8. Automatic lazy and singleton connection for preventing connection exhaustion issues
9. Secure connection using application specific private Keys
10. Telemetry support for key usage, performance reports and failures
11. Management console for managing keys from backend

# Architecture and Concepts

## Key Isolation

## The major problem in using a unified Redis cache cluster for multiple applications is ensuring that the keys operated by numerous applications don't interfere with each other.

## URP tackles Key isolation by introducing prefix-styled name-spacing for each key stored in the cluster. Each onboarded application is registered with a unique moniker (generally 3 or 4 letters long). For every Key added by the application in the cluster, it gets prefixed with this unique moniker.

## Individual applications do not need to change their application code to add the prefix; URP SDK will automatically add them for all operations.

## Hierarchy Cluster

The concept of Cluster refers to a virtual group of Azure Cache for Redis, which may be spread across multiple regions for disaster recovery and reducing regional latency. Generally, a central team at the organization level manages the Cluster.

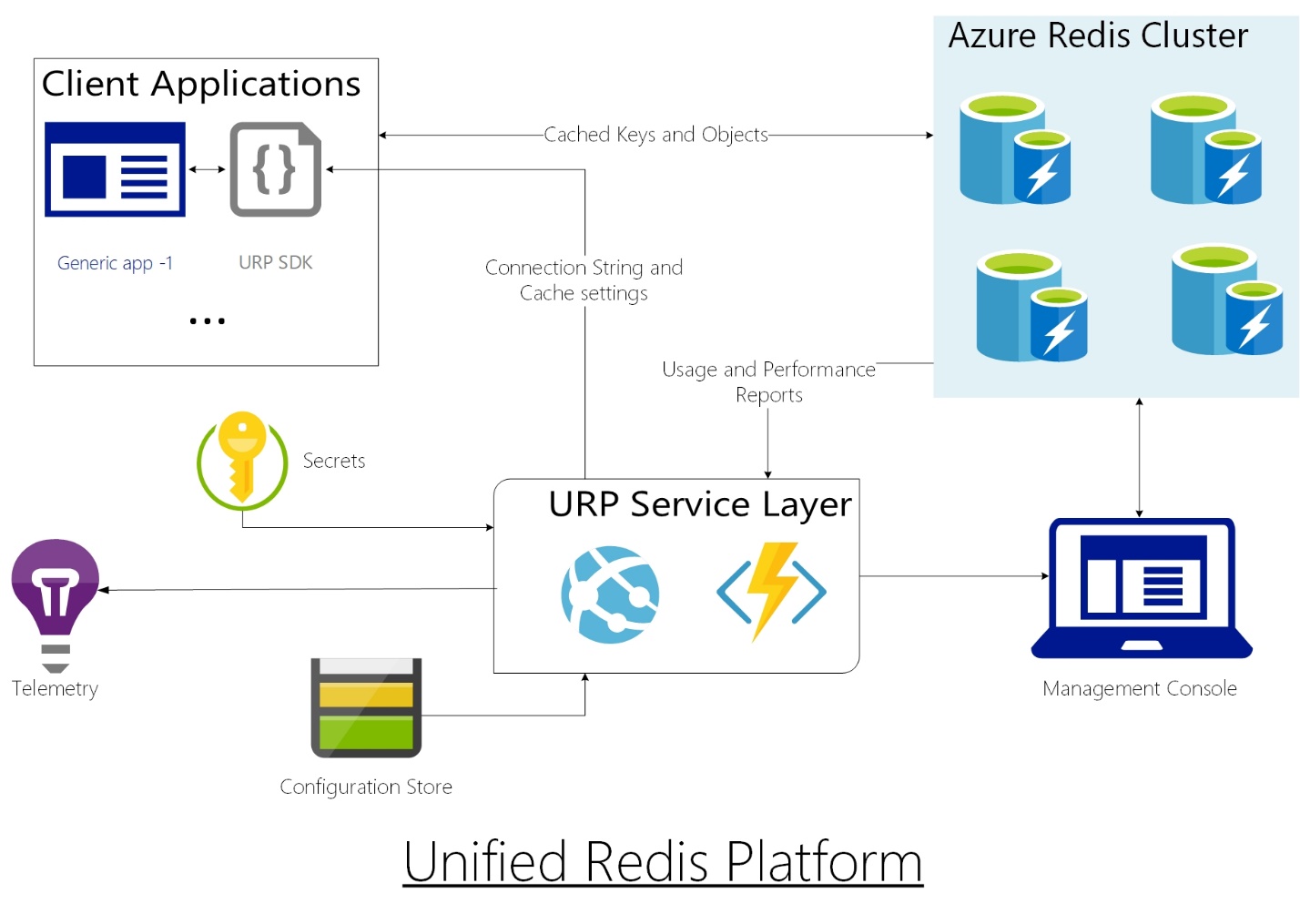
An organization may choose to maintain multiple clusters in extreme scenarios (massive data or extreme client load) or getting higher throughput. We recommend keeping at least 2 Clusters in an organization – one for Production and the other for Pre-Production (with a significant lower tier).

## Geo Replication

When multiple Azure Cache for Redis are present in a single cluster then the URP service will decide the Read region based on the Cache which is geographically closest to the Client Application’s location. Client application can choose to configure the location in URP SDK, or URP service will automatically detect the location of the client application.

In situations where a cluster has multiple Redis Caches, each registered application can either choose to use the Primary Redis Cache for write and update operations or configure the SDK for automatic Geo-Replication. With Geo-Replication enabled, for all Insert, Update and Delete operations the URP SDK will automatically make the required changes in all regions. A downside of this is that until all the regions are updated, the operation won’t be completed. Hence, there is be performance penalty associated with Geo-Replication for Update and Delete operations.

## System Design



### Components

1. URP SDK

It’s an SDK that needs to be installed on the Client’s Application. The SDK is based on .NET Standard 2.0 so is complaint on most of the frameworks. For details about .NET Standard compatibility check this [link](https://docs.microsoft.com/en-us/dotnet/standard/net-standard) - <https://docs.microsoft.com/en-us/dotnet/standard/net-standard>.

The SDK wraps all methods of StackExchange.Redis, hence all operations that teams could perform using StackExchange.Redis can be performed using the URP SDK.

When application is booted, the SDK calls the URP Service and passes the application ID, Cluster ID, and the secret App Key. In return the service sends the connection details of the Redis Caches that the SDK needs to connect for Read and Write operations. The SDK is then responsible for carrying out all the operations on the Redis database.

In addition to the general operations, the SDK is also responsible for Automatic reties, sending usage and performance telemetry and throwing exceptions when time limit is exceeded. The configuration value for all these extra operations can be configured by the Client Application. SDK is also responsible for maintaining Geo-Replication.

1. URP Service

This is an API which maintains the list of all registered applications and connection details (Redis cache connection string) for the clusters. Calls made from URP SDK are authenticated against the App ID, Cluster ID, and the App Secret key. Based on geo-replication settings the Service either returns the lowest latency Redis Cache details or all the cache details in the Cluster.

The service is also responsible for connecting to the caches in the Cluster and collate the telemetry reports and send them to Application Insights.

The services are hosted behind an Azure Front Door which is responsible for load balancing and detecting the geographical location of the calls made from the client. This location is used for detecting the geographically closest Redis Cache in the Cluster.

1. Azure Redis Cluster

This is a group of Azure Cache for Redis where the actual cached data is stored

1. Management Console

Currently the management console is a Console App built on .NET Core which can be used by operations/administrator from each team to manage the cached keys from backend.

# Usage

## Connecting to Cluster

The URP SDK exposes the class UnifiedConnectionMultiplexer which implements IConnectionMultiplexer class from StackExchange.Redis library. This class can be used to connect to the Redis Cluster. In the most general scenario 4 parameters are needed to connect to the cluster

1. Cluster ID – This is the ID of the cluster to which the application has been registered.
2. Application ID – This is a unique ID generated for your application
3. App Key – A secret is generated for each application which is used for authentication
4. [Options] Location – The location of the client application can either be specified when connecting to the cluster. If left blank, then the location is auto detected.

IConnectionMultiplexer \_mux = UnifiedConnectionMultiplexer.Connect(\_clusterName, \_appName, \_appSecret, preferredLocation: \_location);

All existing operations available on IConnectionMultiplexer is also available on UnifiedConnectionMultiplexer.

## Creating Redis Database

Once the connection to the cluster has been established the next step is to create an instance of the Redis Database to perform the required operations. URP SDK exposes the UnifiedRedisDatabase class which is implemented from IDatabase of StackExchange.Redis library, thus providing all operations in StackExchange.Redis.

IDatabase \_database = \_mux.GetDatabase();

All operations available on IDatabase can now be performed. Note that the client application need not add any prefix to the keys, the prefix addition is abstracted and will be taken care by the application. A few examples of using the IDatabase are given below:

await \_database.StringSetAsync(keyName, value);

var value = await \_database.StringGetAsync(keyName);

To see a full list of operations available in StackExchage.Redis library please check the [official documentation](https://stackexchange.github.io/StackExchange.Redis/) of StackExchage.Redis - <https://stackexchange.github.io/StackExchange.Redis/>.

## Other Connection Multiplexer level operations

Apart from the operations available in IConnectionMultiplexer there are a few extra operations which are also available in UnifiedConnectionMultiplexer class which can be used by the client applications. For using these extra methods, client applications must use the IUnifiedConnectionMultiplexer interface. UnifiedConnectionMultiplexer class implements both IConnectionMultiplexer and IUnifiedConnectionMultiplexer interface. Here are the list of operations:

1. Task<List<RedisKey>> GetKeysAsync(string pattern = "") - Gets all the keys from the Redis cluster (includes all caches in a cluster). Applications can also search for keys with the given pattern.
2. List<RedisKey> GetKeys(string pattern = "") - Synchronous version of the above
3. Task FlushAsync(string pattern = "", CommandFlags flags = CommandFlags.None)- Can be used to delete all keys from the Primary Redis Cache in the cluster. A pattern can be provided for deleting keys with the given pattern. Here primary Redis cache indicates the cache which is geographically closest to the client application.
4. void Flush(string pattern = "", CommandFlags flags = CommandFlags.None) - Synchronous version of the above
5. Task FlushSecondaryAsync(string pattern = "", CommandFlags flags = CommandFlags.None) - Used for clearing all keys from the Secondary Redis caches in the cluster. Pattern search can be used.
6. void FlushSecondary(string pattern = "", CommandFlags flags = CommandFlags.None) - Synchronous version of the above.

SDK Details

Name – [Microsoft.UnifiedRedisPlatform.Core](https://dev.azure.com/MicrosoftIT/OneITVSO/_packaging?_a=package&feed=ProfessionalServices&package=Microsoft.UnifiedRedisPlatform.Core&version=1.1.0-rc3&protocolType=NuGet&view=versions) (Always check for the Latest version)

Feed – [ProfessionalServices](https://dev.azure.com/MicrosoftIT/OneITVSO/_packaging?_a=connect&feed=ProfessionalServices)

SDK-Link - [ProfessionalServices (azure.com)](https://dev.azure.com/MicrosoftIT/OneITVSO/_packaging?_a=package&feed=ProfessionalServices&package=Microsoft.UnifiedRedisPlatform.Core&version=1.2.0-rc-1.0&protocolType=NuGet)

ASP.NET Core Extension SDK – [Microsoft.Extensions.Caching.UnifiedRedisPlatform 1.0.0-rc-1.0 (azure.com)](https://dev.azure.com/MicrosoftIT/OneITVSO/_packaging?_a=package&feed=ProfessionalServices&package=Microsoft.Extensions.Caching.UnifiedRedisPlatform&protocolType=NuGet&version=1.0.0-rc-1.0)

# Registration Steps (On-boarding)

Cluster Registration

For registering a new cluster please contact the following:

1. Pratik Bhattacharya – [pratikb@microsoft.com](mailto:pratikb@microsoft.com)
2. Rohit J. Kshirsagar – [rokshi@microsoft.com](mailto:rokshi@microsoft.com)

Application Registration

For registering an existing application please contact your cluster administrator. If you are not sure about your cluster admin contact the above people. The following information needs to be provided for registering your application

|  |  |
| --- | --- |
| **Information** | **Description** |
| Application Name | For uniformity please provide the Service Tree Component Name of your application. If multiple applications are sharing the same component, then add proper suffix to the Component Name |
| Support Contact | Security Group (with at least 1 FTE) that can be contacted for any updates, announcements, or emergencies |
| Write Policy | For enabling Geo-Replication with various options. Possible values include   1. Write-Through – All update and delete operations will be done to all caches in the cluster 2. Delete-Through – Only delete operations will be replicated 3. Not-Managed – Update and delete operations will be performed only on the Primary Redis Cache |
| Retry Options | Following details are needed   1. Maximum Retry Count [Default - 50] 2. Backoff interval – Time gap between 2 attempts in milliseconds after an unsuccessful attempt (provide a maximum and minimum value). [Default – 5000-15000] 3. Timeout – Maximum waiting period in milliseconds after which timeout exception will be thrown if response is not received [Default – 15000) 4. Enable Hard Timeout – If false then timeout exceptions won’t be thrown |
| Logging | Mention if telemetry needs to be captured. If telemetry details are not captured, then usage details and performance reports won’t be available for your application  Note: Cached values are not captured only the key names are captured in telemetry. |

After successful registration of your application, the Cluster ID, Application ID, and secret App Key will be sent to you.

Use these values to connect to the URP Cluster. For security purposes we recommend storing the secret App key in Key Vault, or use other form of encryption.

# Management Console

The management console can be used by Administrators and DRIs for connecting to the Redis Cluster for performing maintenance duties. All operations performed in the Management Console have isolation safety, i.e. operations performed by one team won’t interfere with cached keys and objects of other applications.

Currently the Management Console is a C# Console Application. We are working towards improving the experience.

The following operations are allowed from the Management Console:

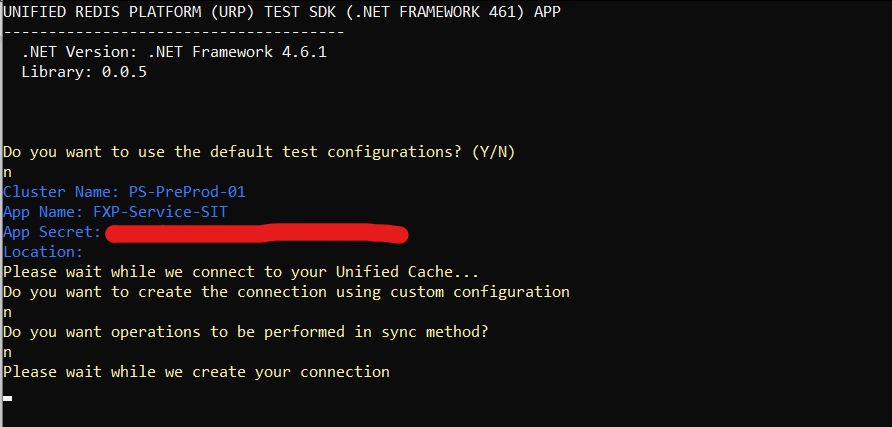
1. Check value of a key (only string data type)
2. Add a new key (only string data type)
3. Delete a key
4. See all keys
5. Flush the database
6. Ping the cluster

The tool can be downloaded from the below location:

<https://microsoft-my.sharepoint.com/:u:/p/pratikb/EcrSBYzUXYlDgNTxa9ggqOUBbFJdCnPNh5W6cRSSPLO34Q?e=GfIR8l>

Unzip the files and open “URP Client” link.

Provide the Cluster ID, App ID, and App Secret to connect to your application



# Appendix – 1: Code and Samples

Code Links

1. SDK - <https://dev.azure.com/MicrosoftIT/OneITVSO/_git/PS-GCM-FExP-FxpTools?path=%2Fsrc%2FUnifiedRedisPlatform%2Fsrc%2Fsdk%2FMicrosoft.UnifiedRedisPlatform.SDK>
2. IDistributedCache SDK -
3. Service/API - <https://dev.azure.com/MicrosoftIT/OneITVSO/_git/PS-GCM-FExP-FxpTools?path=%2Fsrc%2FUnifiedRedisPlatform%2Fsrc%2Fservice%2FMicrosoft.UnifiedRedisPlatform.Service>

Sample Applications

1. .NET Core - <https://dev.azure.com/MicrosoftIT/OneITVSO/_git/PS-GCM-FExP-FxpTools?path=%2Fsrc%2FUnifiedRedisPlatform%2Fsrc%2Ftests%2FMicrosoft.UnifiedRedisPlatform.TestApps%2FTestConsoleCore.SDK>
2. .NET Framework (4.6.1) - <https://dev.azure.com/MicrosoftIT/OneITVSO/_git/PS-GCM-FExP-FxpTools?path=%2Fsrc%2FUnifiedRedisPlatform%2Fsrc%2Ftests%2FMicrosoft.UnifiedRedisPlatform.TestApps%2FTestConsole461.SDK>

Migrating from StackExchange.Redis to URP SDK (real applications)

1. FXP Service - <https://dev.azure.com/MicrosoftIT/OneITVSO/_git/PS-GCM-FExP-FxpService/pullrequest/454618?_a=files>
2. FXP Configuration Service - <https://dev.azure.com/MicrosoftIT/OneITVSO/_git/PS-GCM-FExP-Confit/pullrequest/454619>