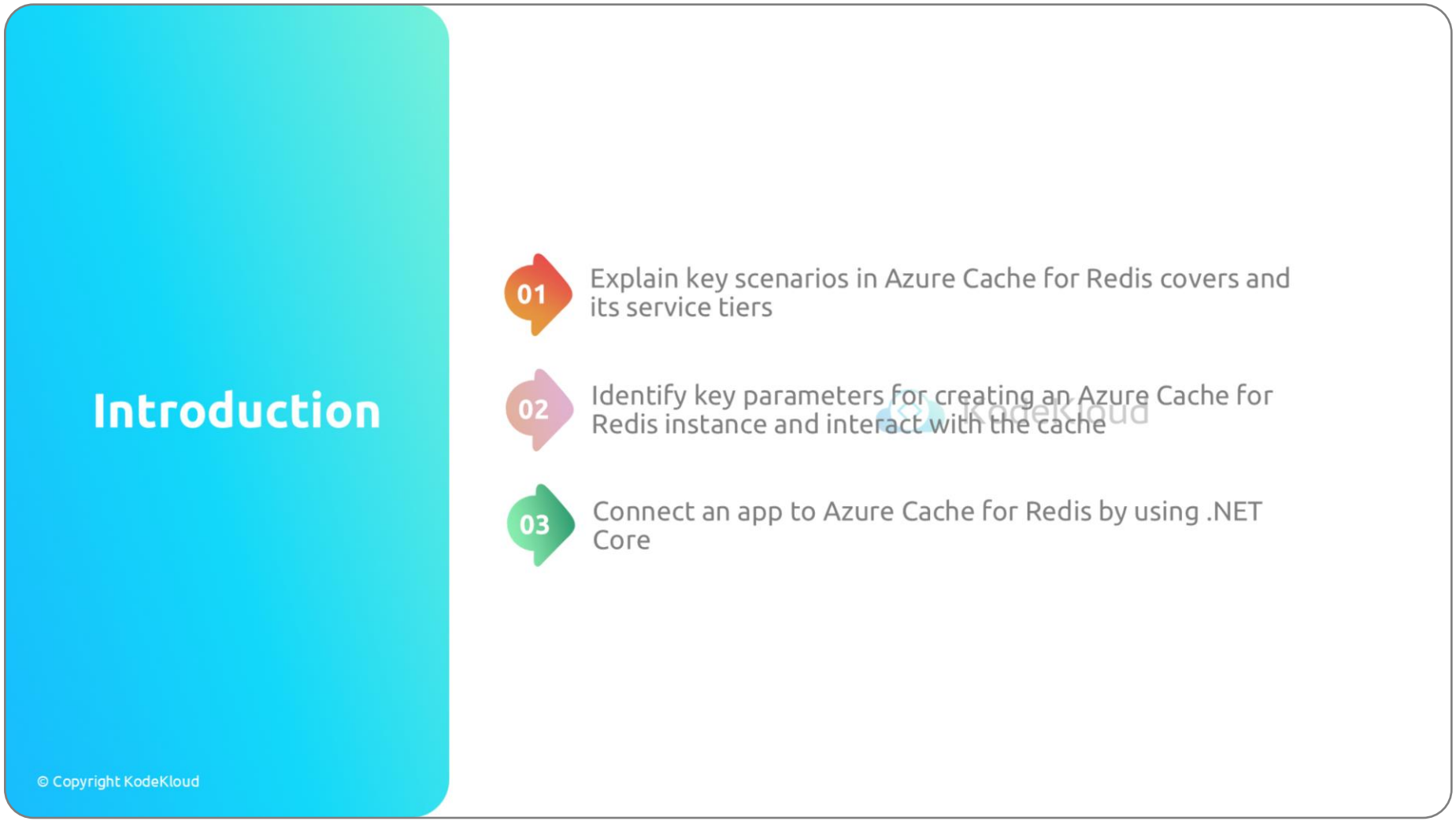
KodeKloud

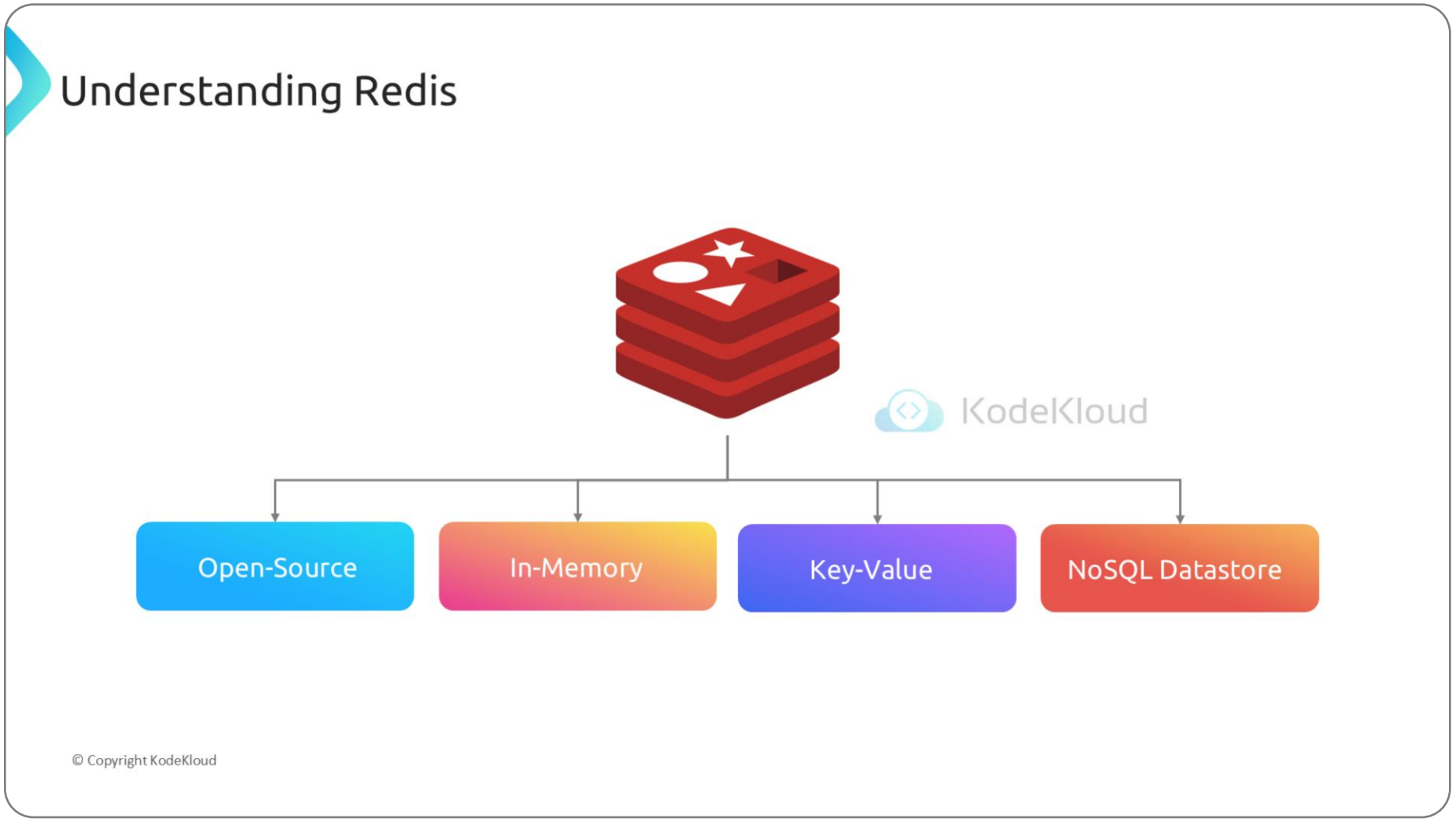
Visit www.kodekloud.com to learn more.

Developing Azure Cache for Redis



Prerequisites
You should be familiar with developer concepts and terminology.
An understanding of cloud computing and some experience with the Azure portal.

Azure Cache for Redis



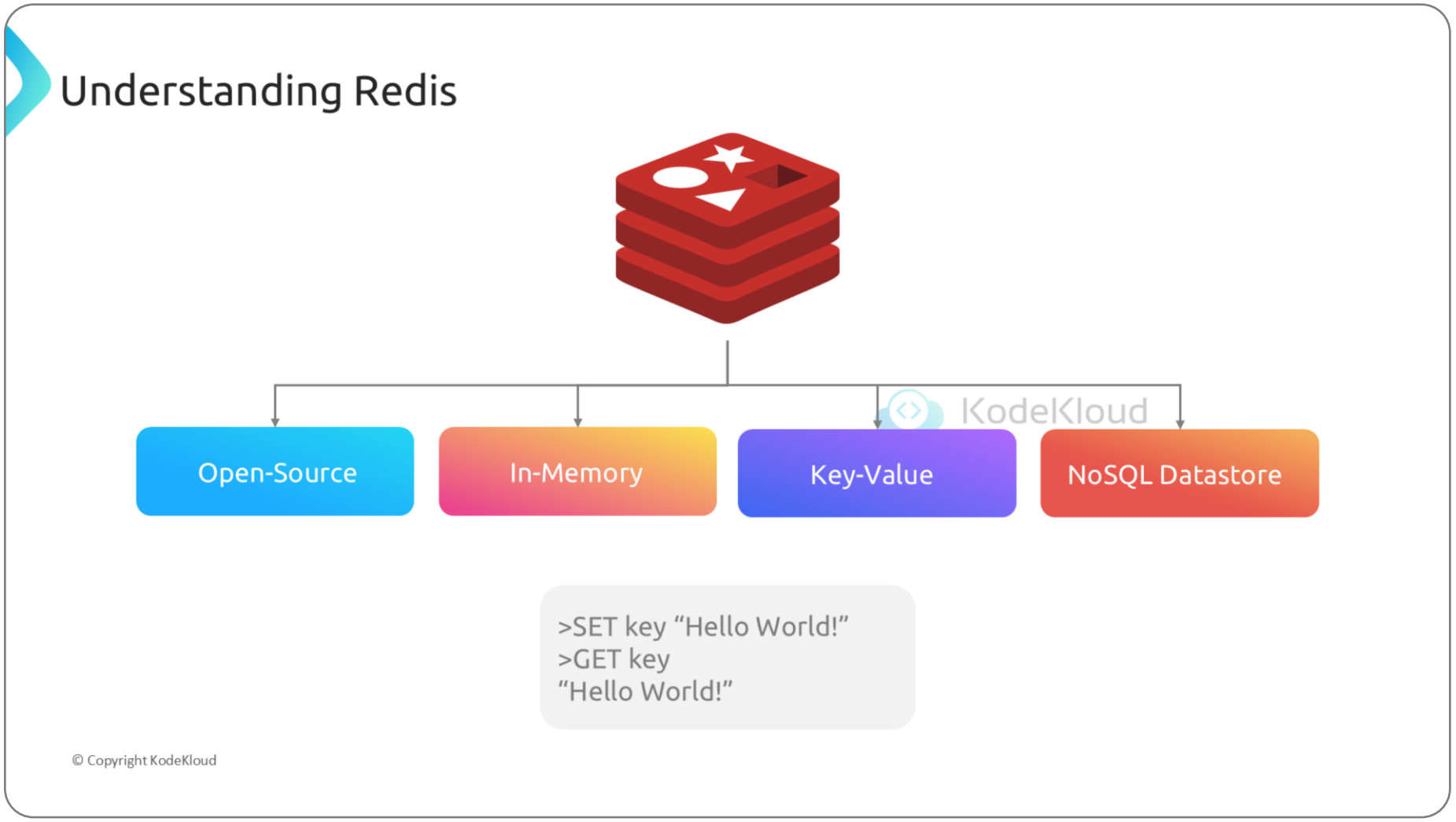
Lets gets started with Core Azure Redis Cache concepts. The first question to ask is what is Redis ? Redis Cache is a high-speed, open-source database system that specializes in caching and storing data in-memory, using a simple key-value structure, and falls into the category of NoSQL databases. That's quite a mouthful, so lets break it down.

Redis, an open-source platform, enjoys immense popularity due to its continuous updates. Being an in-memory database, it offers lightning-fast operations compared to traditional disk-based databases. You might ask what is in-memory database. So, an in-memory database, like Redis, operates by storing data primarily in the computer's main memory (RAM), as opposed to traditional disk-based

databases where data is stored on disk drives. This unique approach of keeping the data in memory allows for extremely fast data access and retrieval. As a result, the read and write operations in an in-memory database are significantly quicker compared to disk-based databases.

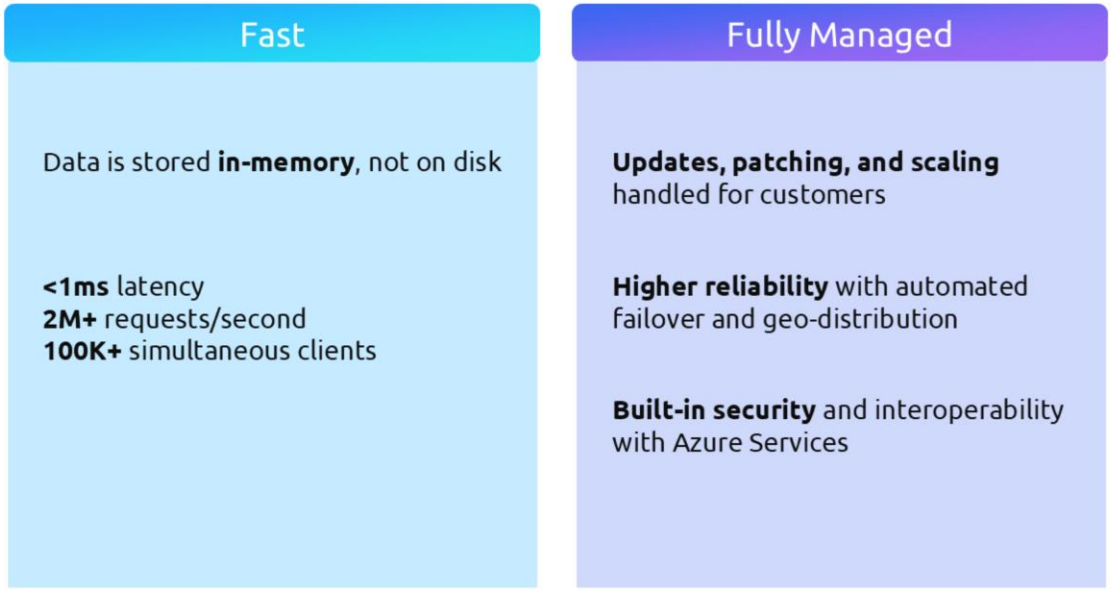
Redis used key-value structure which simplifies data management, making it highly intuitive and efficient. Redis not only leverages this core structure but also incorporates additional data structures, maintaining simplicity while empowering powerful functionalities.

As a NoSQL database, Redis excels in scalability. Frequently used as a cache, its NoSQL capabilities enable handling massive throughputs, making it a compelling choice for various applications seeking robust and scalable solutions.



Okay. So just a really basic example of what Redis looks like. This is using the set and get functionality. Just key value. I set a key basically the word key is the key. And the value is hello world, and then when i get it, i get the value back. Hello world. So that's kind of the basic example.

Azure Cache for Redis



Microsoft has kind of enhanced the open-source Redis by doing a fully managed and hosted version on Azure as a PaaS service. In terms of being fast, you can get sub milisecond service side latency; it can accommodate more than 2 million requests per second with over 100K parallel clients accessing the cache.

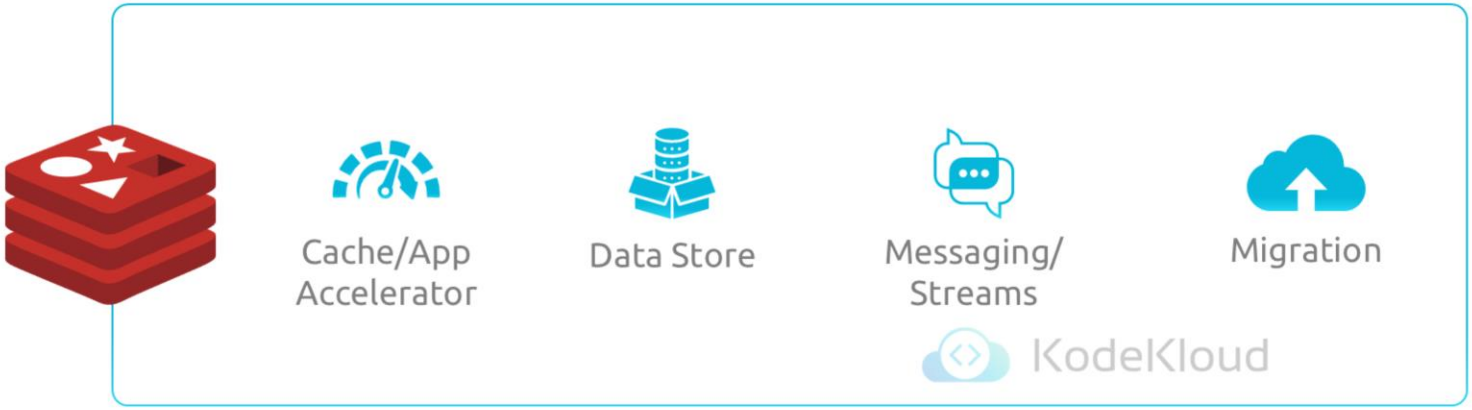
BTW, redis is most loved Database in the stackoverflow survey for 5 years in a row up until 2022.

Because it's a fully managed Service, Microsoft will manage updates, patching, scaling and replication for you.

Redis – Real-World Use Cases

Lets now look at some of the use cases for Redis. This list if off course not exhaustive, but lets talk about the most common ones

Redis – Real-World Use Cases

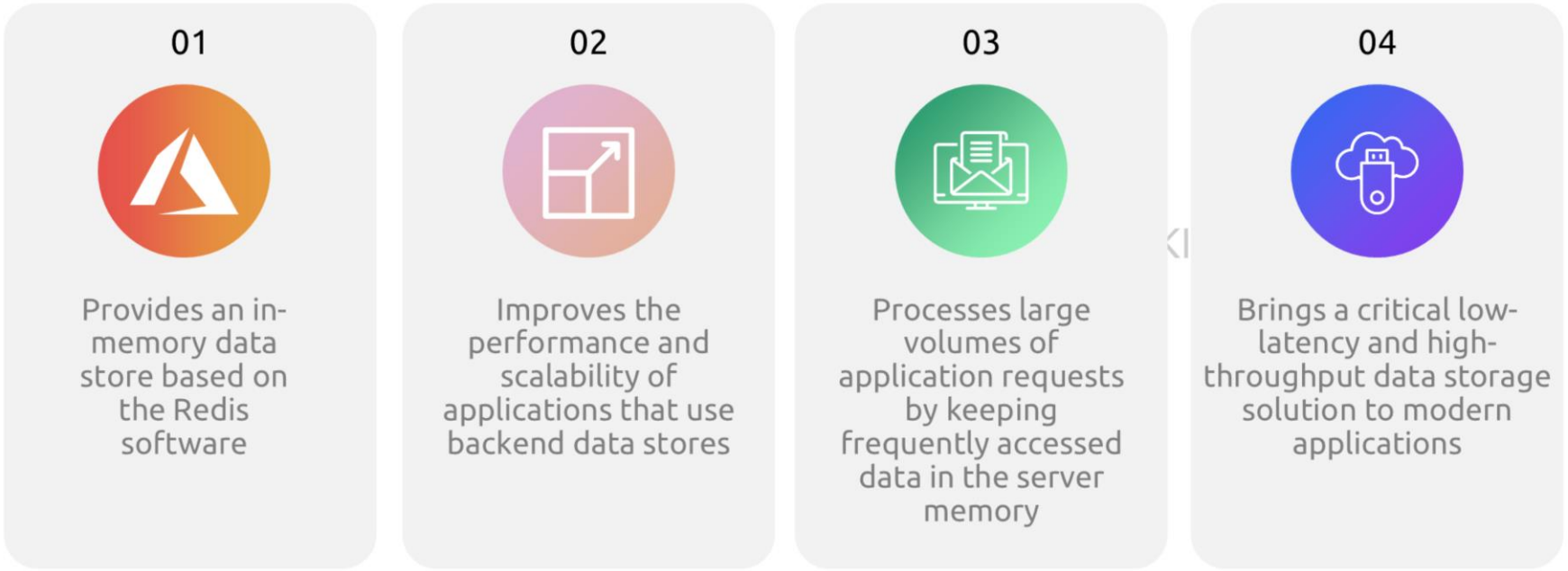


- Reduce database access costs
- API response caching
- ML feature store
- Accelerate stream processing
- Share state across cloud
- Optimize memory for K8s pods

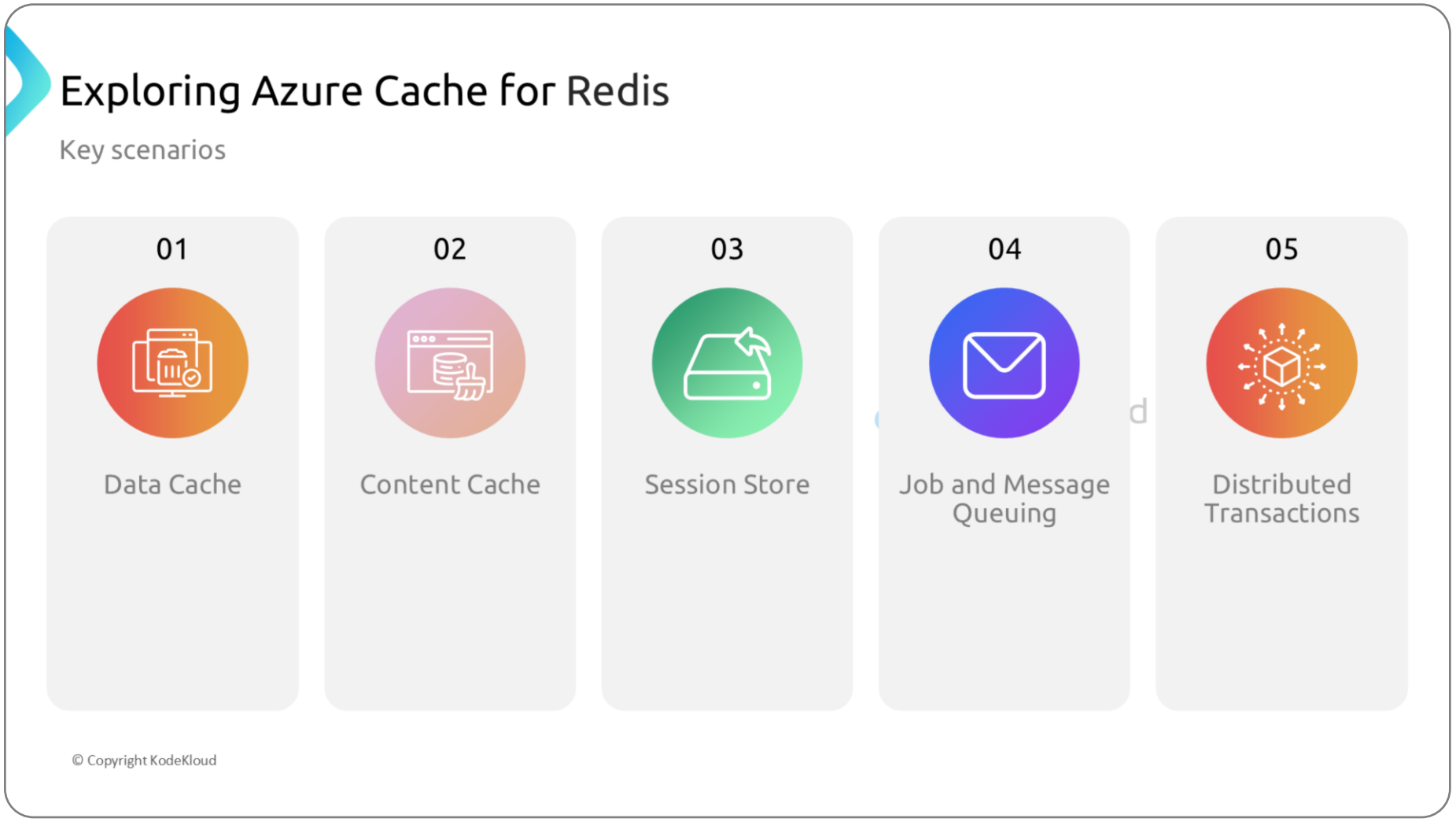
- User session data store
- Accelerate datastore access
- Microservices state store
- Message broker
- Dev/test alternative for CosmosDB
- Share state in hybrid environment

Lets now look at some of the use cases for Redis. This list if off course not exhaustive, but lets talk about the most common ones

Exploring Azure Cache for Redis



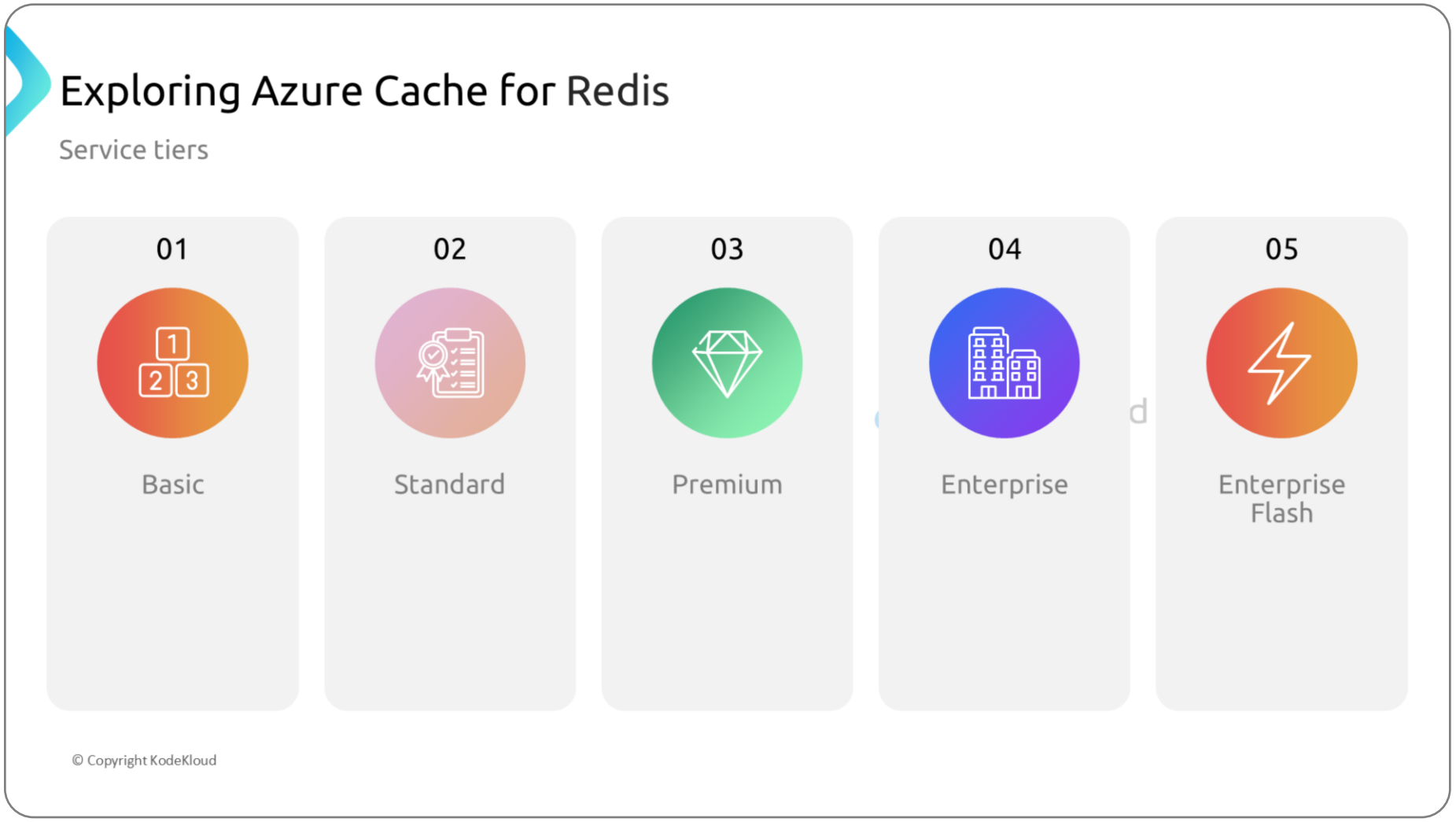
Azure Cache for Redis offers both the Redis open-source (OSS Redis) and a commercial product from Redis Labs (Redis Enterprise) as a managed service. It provides secure and dedicated Redis server instances and full Redis API compatibility. The service is operated by Microsoft, hosted on Azure, and usable by any application within or outside of Azure.



Key scenarios

- Data cache - Databases are often too large to load directly into a cache. It's common to use the cache-aside pattern to load data into the cache only as needed. When the system makes changes to the data, the system can also update the cache, which is then distributed to other clients.
- Content cache - Many web pages are generated from templates that use static content such as headers, footers, banners. These static items shouldn't change often. Using an in-memory cache provides quick access to static content compared to backend datastores.

- Session store - This pattern is commonly used with shopping carts and other user history data that a web application might associate with user cookies. Storing too much in a cookie can have a negative effect on performance as the cookie size grows and is passed and validated with every request. A typical solution uses the cookie as a key to query the data in a 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.
- Job and message queuing - Applications often add tasks to a queue when the operations associated with the request take time to execute. Longer running operations are queued to be processed in sequence, often by another server. This method of deferring work is called task queuing.
- Distributed transactions - Applications sometimes require a series of commands against a backend data-store to execute as a single atomic operation. 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 transaction.



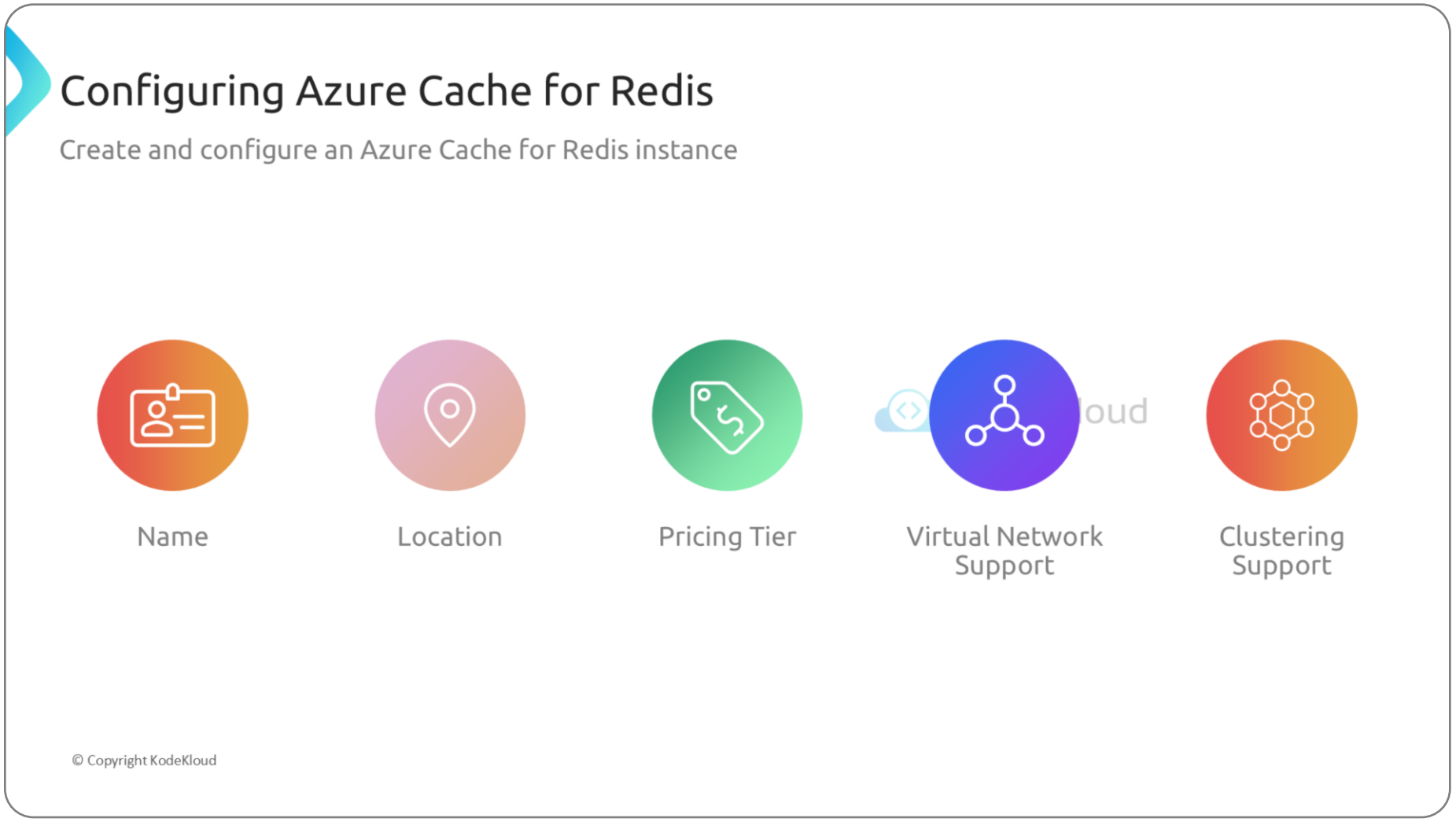
Service tiers

- Basic - An OSS Redis cache running on a single VM. This tier has no service-level agreement (SLA) and is ideal for development/test and non-critical workloads.
- Standard - An OSS Redis cache running on two VMs in a replicated configuration.
- Premium - High-performance OSS Redis caches. This tier offers higher throughput, lower latency, better availability, and more features. Premium caches are deployed on more powerful VMs compared to the VMs for Basic or Standard caches.
- Enterprise - High-performance caches powered by Redis Labs' Redis Enterprise software. This tier supports Redis

modules including RediSearch, RedisBloom, and RedisTimeSeries. Also, it offers even higher availability than the Premium tier.

- Enterprise Flash - Cost-effective large caches powered by Redis Labs' Redis Enterprise software. This tier extends Redis data storage to non-volatile memory, which is cheaper than DRAM, on a VM. It reduces the overall per-GB memory cost.

Configuring Azure Redis Cache



You can create a Redis cache using the Azure portal, the Azure CLI, or Azure PowerShell.

Create and configure an Azure Cache for Redis instance

Name - The Redis cache will need a globally unique name. The name has to be unique within Azure because it is used to generate a public-facing URL to connect and communicate with the service.

Location - You will need to decide where the Redis cache will be physically located by selecting an Azure region. You should always place 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.
Important: Put the Redis cache as close to the data consumer as you can.

Pricing tier – There are three pricing tiers available for an Azure Cache for Redis.
Basic: Basic cache ideal for development/testing. Is limited to a single server, 53 GB of memory, and 20,000 connections. There is no SLA for this service tier.
Standard: Production cache which supports replication and includes an SLA. It supports two servers, and has the same memory/connection limits as the Basic tier.
Premium: Enterprise tier which builds on the Standard tier and includes persistence, clustering, and scale-out cache support. This is the highest performing tier with up to 530 GB of memory and 40,000 simultaneous connections.
You can control the amount of cache memory available on each tier.
Tip: Microsoft recommends you always use Standard or Premium Tier for production systems. The Basic Tier is a single node system with no data replication and no SLA.

The Premium tier allows you to persist data in two ways to provide disaster recovery:
RDB persistence takes a periodic snapshot and can rebuild the cache using the snapshot in case of failure.
AOF persistence saves every write operation to a log that is saved at least once per second. This creates bigger files than RDB but has less data loss.
There are several other settings which are only available to the Premium tier.

Virtual Network support - If you create a premium tier Redis cache, you can deploy it to a virtual network in the cloud. Your cache will be available to only other virtual machines and applications in the same virtual network. This provides a higher level of security when your service and cache are both hosted in Azure, or are connected through an Azure virtual network VPN.

Clustering support - With a premium tier Redis cache, you can implement clustering to automatically split

your dataset among multiple nodes. To implement clustering, you specify the number of shards to a maximum of 10. The cost incurred is the cost of the original node, multiplied by the number of shards.

Configuring Azure Cache for Redis

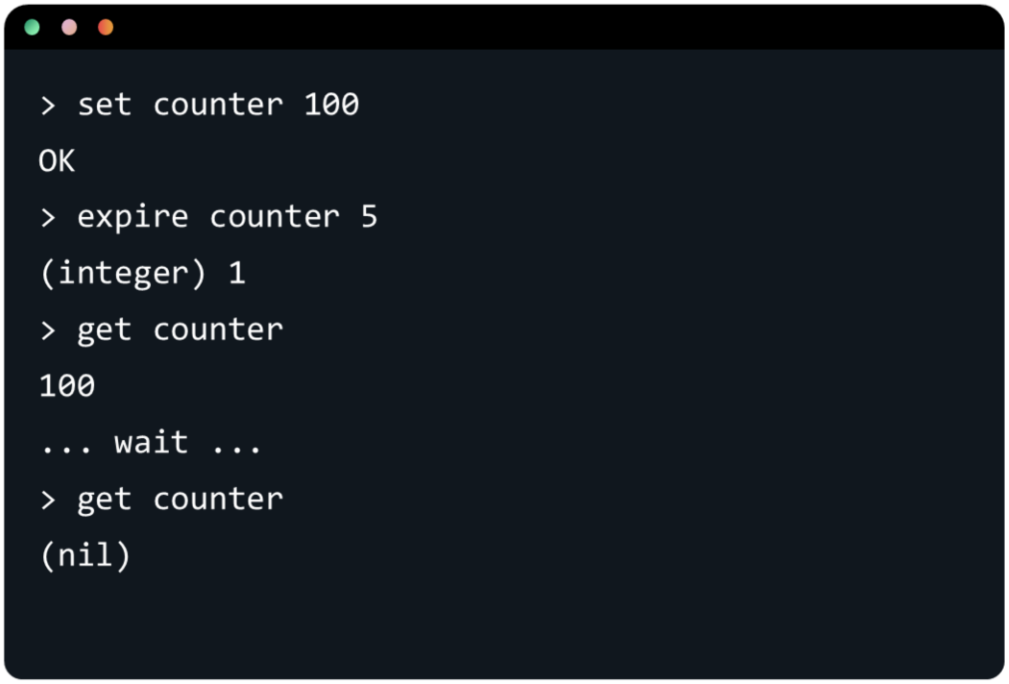
Accessing the Redis instance

Command	Description	Command	Description
ping	Pings the server; returns "PONG"	get [key]	Gets a value from the cache
set [key] [value]	Sets a key/value in the cache; returns "OK" on success	exists [key]	Returns '1' if the **key** exists in the cache, '0' if it doesn't.
incr [key]	Increment the given value associated with **key** by '1'. The value must be an integer or double value. This returns the new value.	incrby [key] [amount]	Increment the given value associated with **key** by the specified amount. The value must be an integer or double value. This returns the new value.
type [key]	Returns the type associated to the value for the given **key**	del [key]	Deletes the value associated with the **key**
flushdb	Deletes all keys and values in the database		

Redis has a command-line tool for interacting with an Azure Cache for Redis as a client. 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.

Configuring Azure Cache for Redis

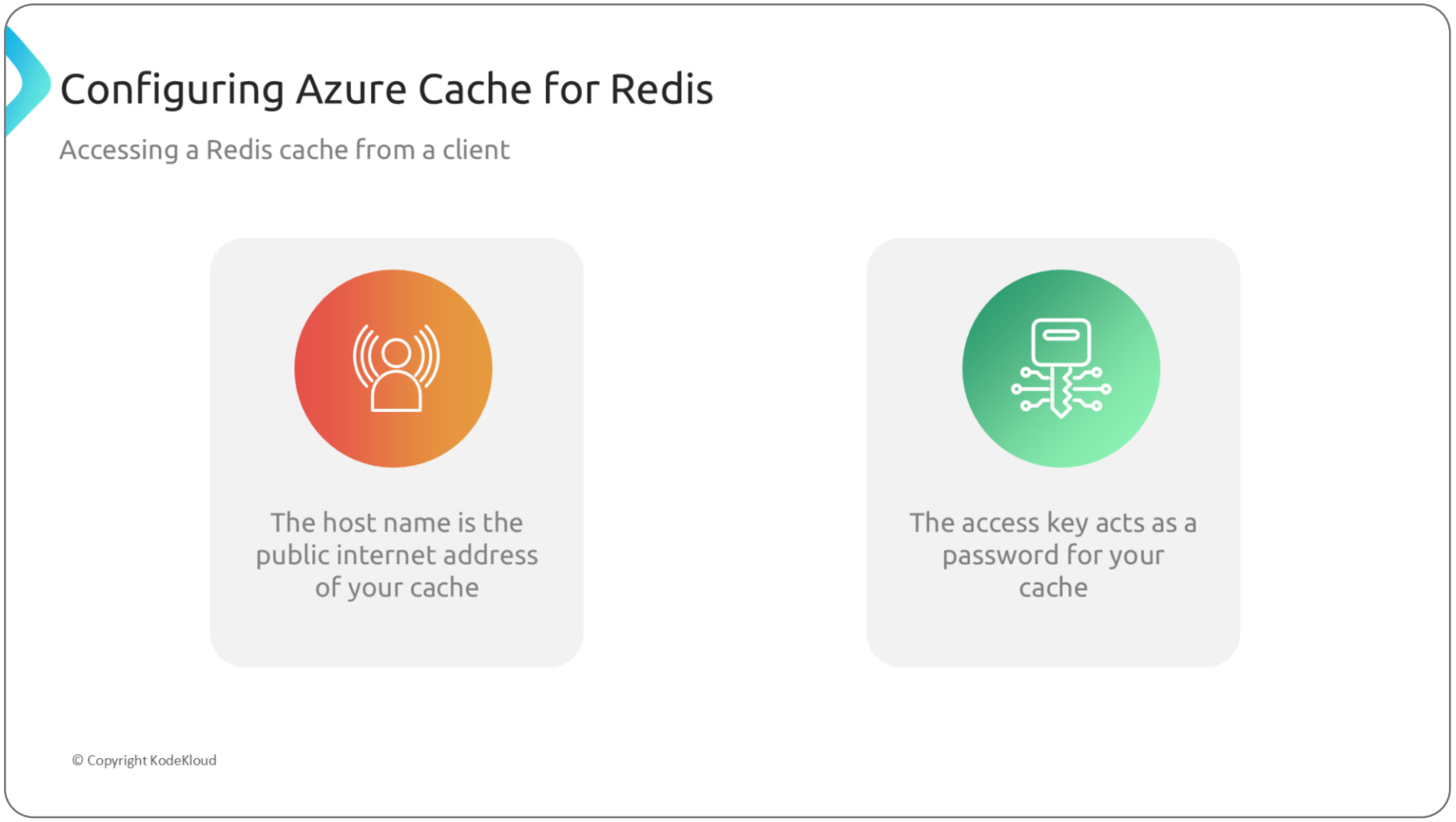
Adding an expiration time to values



KodeKloud

In Redis we expire values when they are stale by applying a time to live (TTL) to a key.
When the TTL elapses, the key is automatically deleted, exactly as if the DEL command were issued.

- Expirations can be set using seconds or milliseconds precision.
- The expire time resolution is always 1 millisecond.



To connect to an Azure Cache for Redis instance, you'll need the host name, port, and an access key for the cache. You can retrieve this information in the Azure portal.
The host name is the public Internet address of your cache, which was created using the name of the cache. For example, sportsresults.redis.cache.windows.net.
The access key acts as a password for your cache. There are two keys created: primary and secondary.

Primary key and secondary key:

You can use either key, two are provided in case you need to change the primary key.
You can switch all of your clients to the secondary key, and regenerate the primary key.
This would block any applications using the original primary key.
Microsoft recommends periodically regenerating the keys - much like you would your personal passwords.

Interacting With Azure Cache for Redis by Using .NET

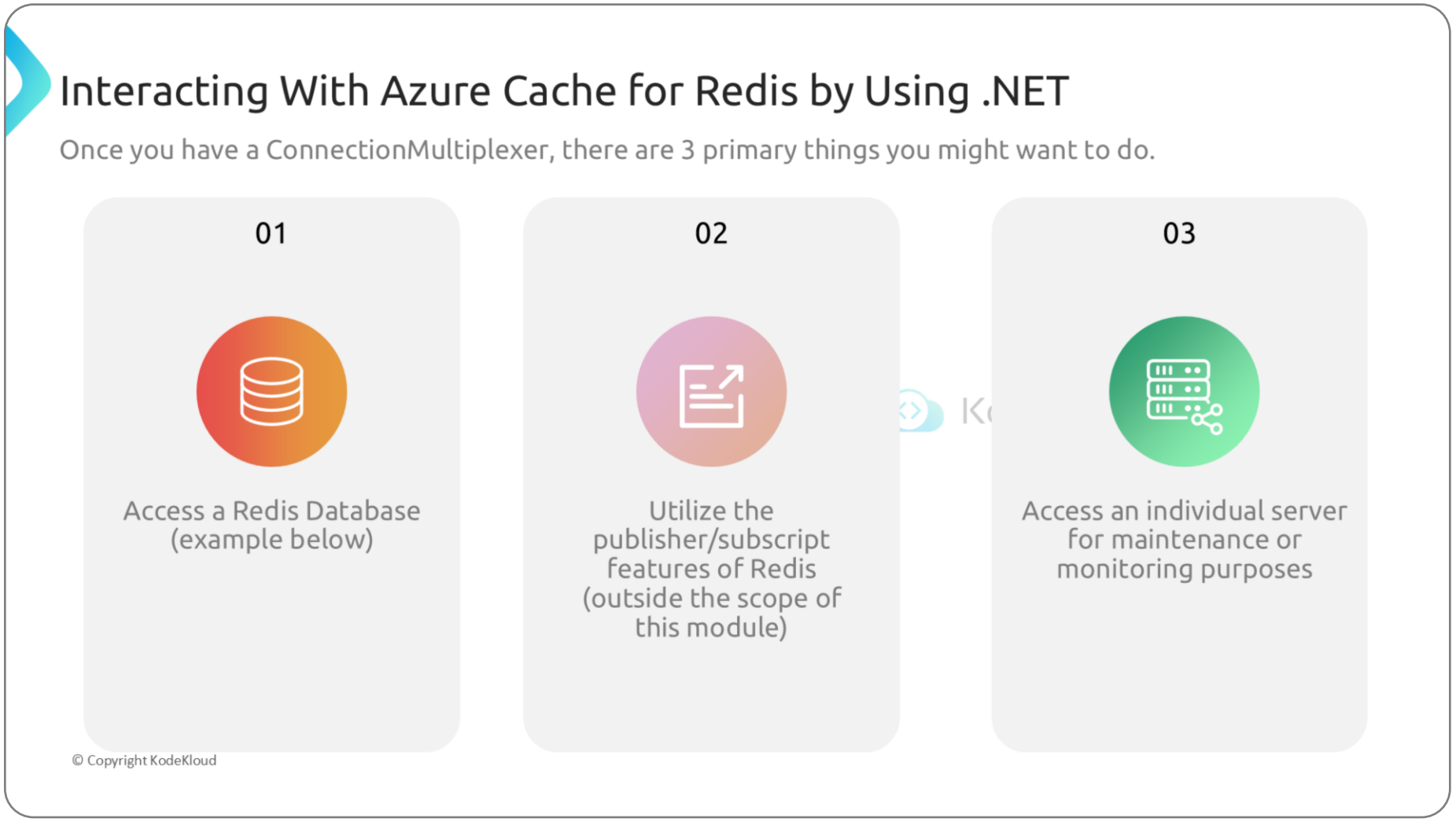
Interacting With Azure Cache for Redis by Using .NET

Executing commands on the Redis cache

```
//Creating a connection


using StackExchange.Redis;

...
var connectionString = "[cache-name].redis.cache.windows.net:6380,password=[password-
here],ssl=True,abortConnect=False";

var redisConnection = ConnectionMultiplexer.Connect(connectionString);
```


Executing commands on the Redis cache
A popular high-performance Redis client for the .NET language is StackExchange.Redis. The package is available through NuGet and can be added to your .NET code using the command line or IDE.

The main connection object in StackExchange.Redis is the StackExchange.Redis.ConnectionMultiplexer class.



Once you have a ConnectionMultiplexer, there are 3 primary things you might want to do:
Access a Redis Database (example below)
Make use of the publisher/subscript features of Redis. (Outside the scope of this module.)
Access an individual server for maintenance or monitoring purposes.

The Redis database is represented by the IDatabase type. You can retrieve one using the GetDatabase() method.

Once you have a IDatabase object, you can execute methods to interact with the cache.

The StringSet method returns a bool indicating whether the value was set (true) or not (false). We can then retrieve the value with the StringGet method

Interacting With Azure Cache for Redis by Using .NET

```
// Accessing a database
IDatabase db = redisConnection.GetDatabase();

// Example of storing a key/value in the cache
bool wasSet = db.StringSet("favorite:flavor", "i-love-rocky-road");

// Retrieving the value
string value = db.StringGet("favorite:flavor");
Console.WriteLine(value); // displays: ""i-love-rocky-road""
```


Once you have a ConnectionMultiplexer, there are 3 primary things you might want to do:
Access a Redis Database (example below)
Make use of the publisher/subscript features of Redis. (Outside the scope of this module.)
Access an individual server for maintenance or monitoring purposes.

The Redis database is represented by the IDatabase type. You can retrieve one using the GetDatabase() method.

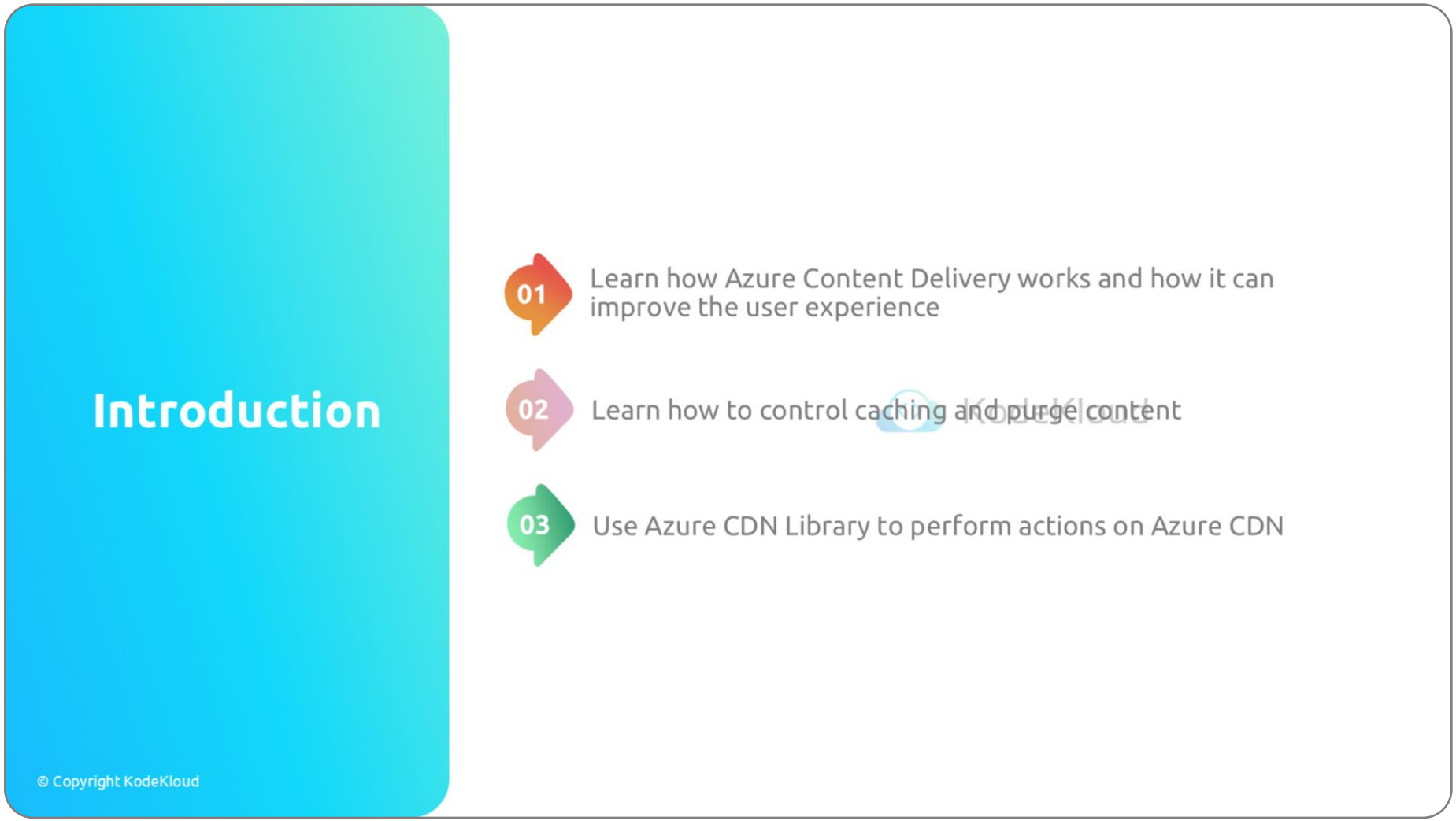
Once you have a IDatabase object, you can execute methods to interact with the cache.

The StringSet method returns a bool indicating whether the value was set (true) or not (false). We can then retrieve the value with the StringGet method

Interacting With Azure Cache for Redis by Using .NET

Method	Description
CreateBatch	Creates a group of operations that will be sent to the server as a single unit, but not necessarily processed as a unit
CreateTransaction	Creates a group of operations that will be sent to the server as a single unit and processed on the server as a single unit
KeyDelete	Delete the key/value
KeyExists	Returns whether the given key exists in cache
KeyExpire	Sets a time-to-live (TTL) expiration on a key
KeyRename	Renames a key
KeyTimeToLive	Returns the TTL for a key
KeyType	Returns the string representation of the type of value stored at key. Different types that can be returned: string, list, set, zset, and hash

Developing for Storage on CDNs

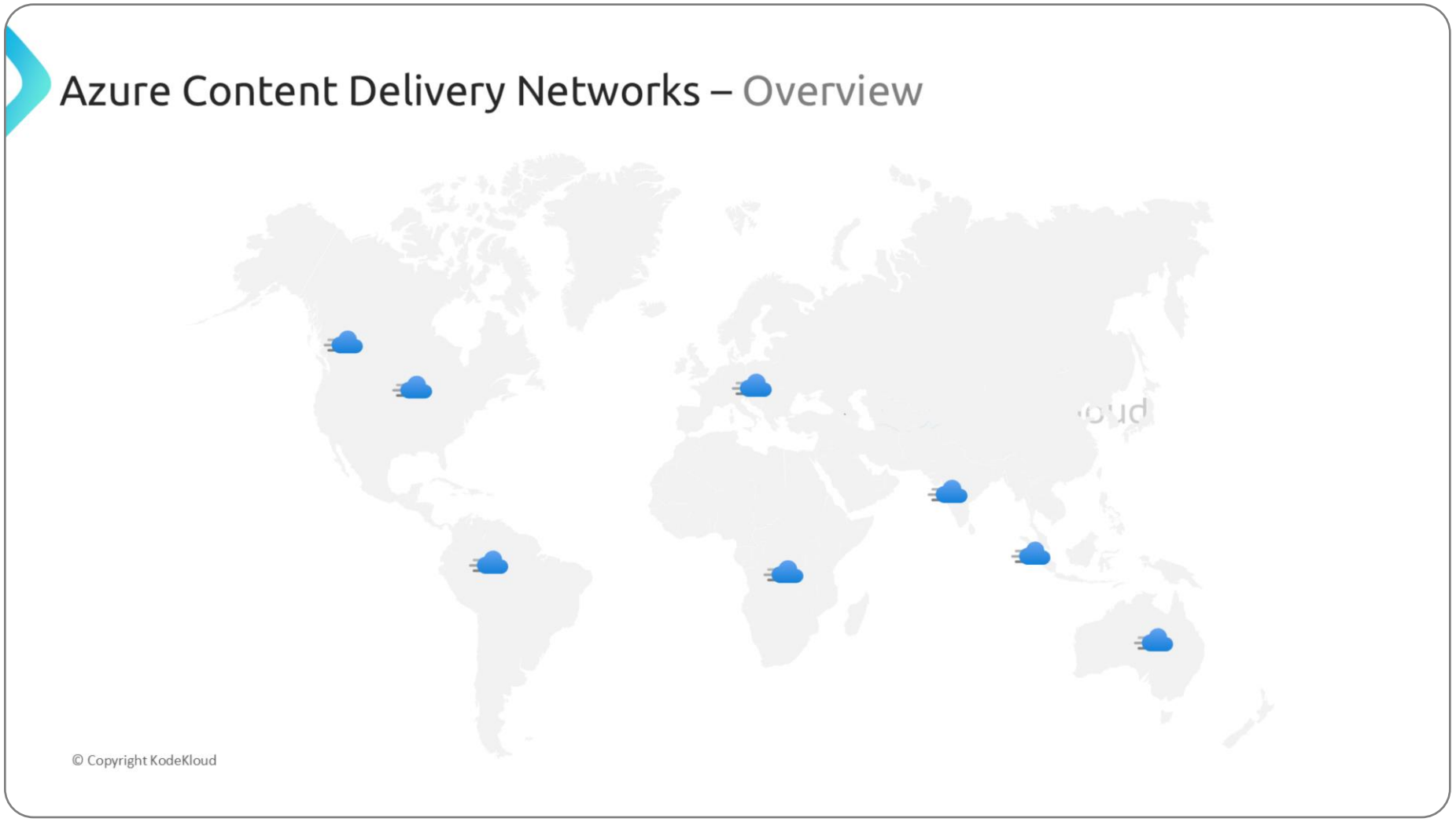


Prerequisites
You should be familiar with developer concepts and terminology.
An understanding of cloud computing and some experience with the Azure portal.

Exploring Azure CDN

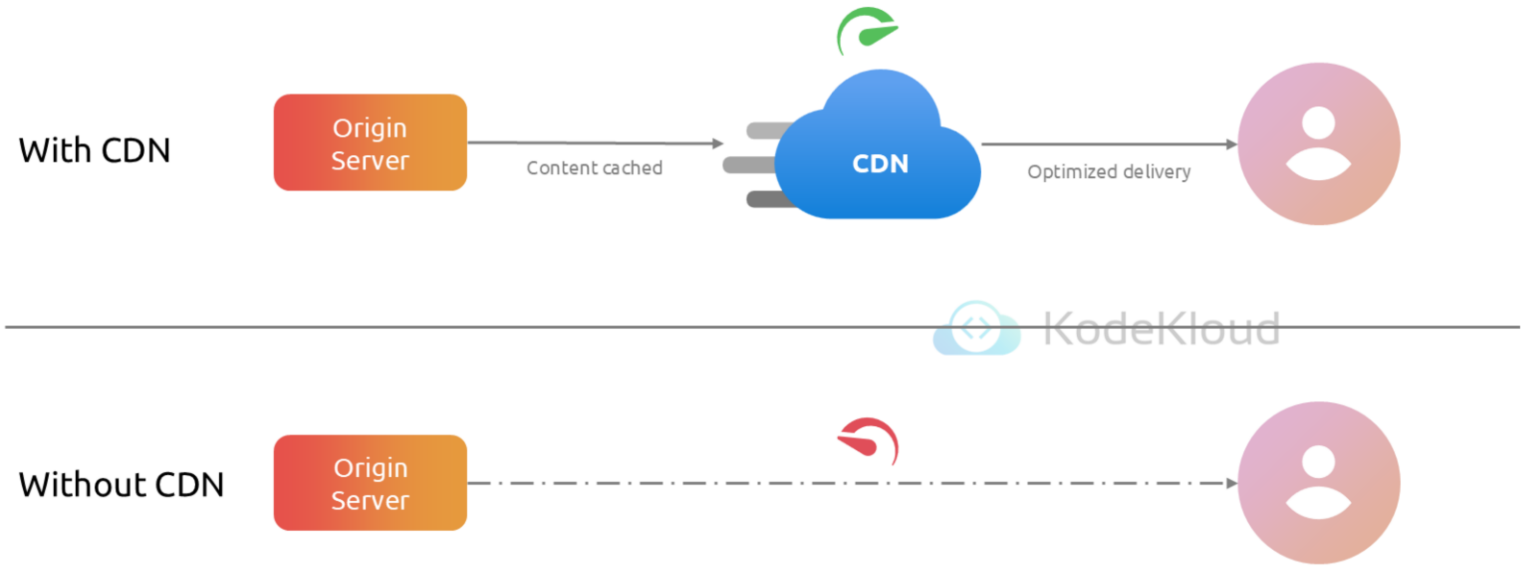
Azure Content Delivery Networks – Overview

Azure Content Delivery Networks (1 of 4)
Overview
Azure Content Delivery Network (CDN) delivers high-bandwidth content to users by caching it at strategically placed physical nodes worldwide.
It also accelerates dynamic content that cannot be cached by utilizing various network optimizations through CDN Points of Presence (POPs).



Azure Content Delivery Networks (1 of 4)
Overview
Azure Content Delivery Network (CDN) delivers high-bandwidth content to users by caching it at strategically placed physical nodes worldwide.

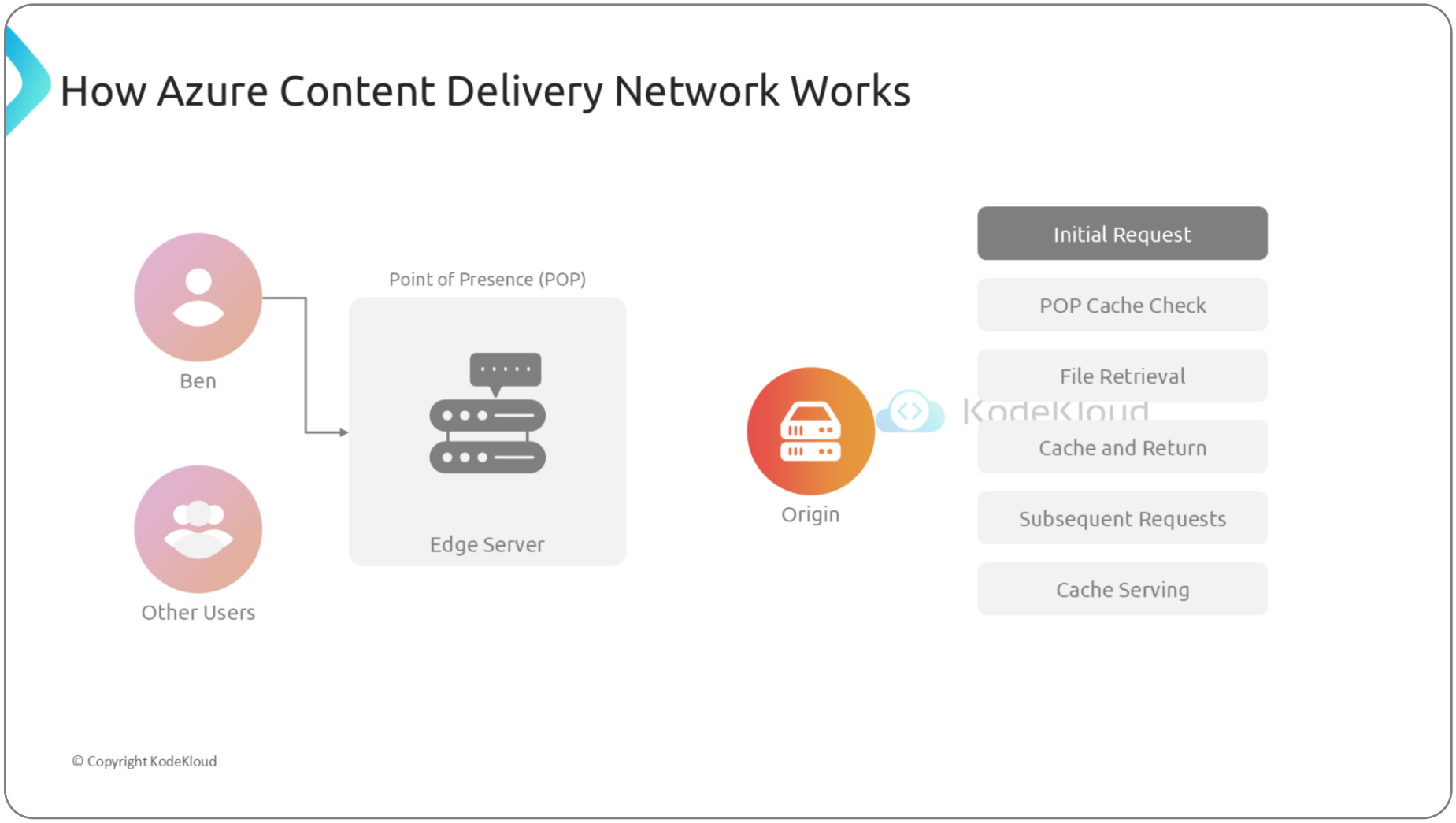
Azure Content Delivery Networks – Overview



It also accelerates dynamic content that cannot be cached by utilizing various network optimizations through CDN Points of Presence (POPs).

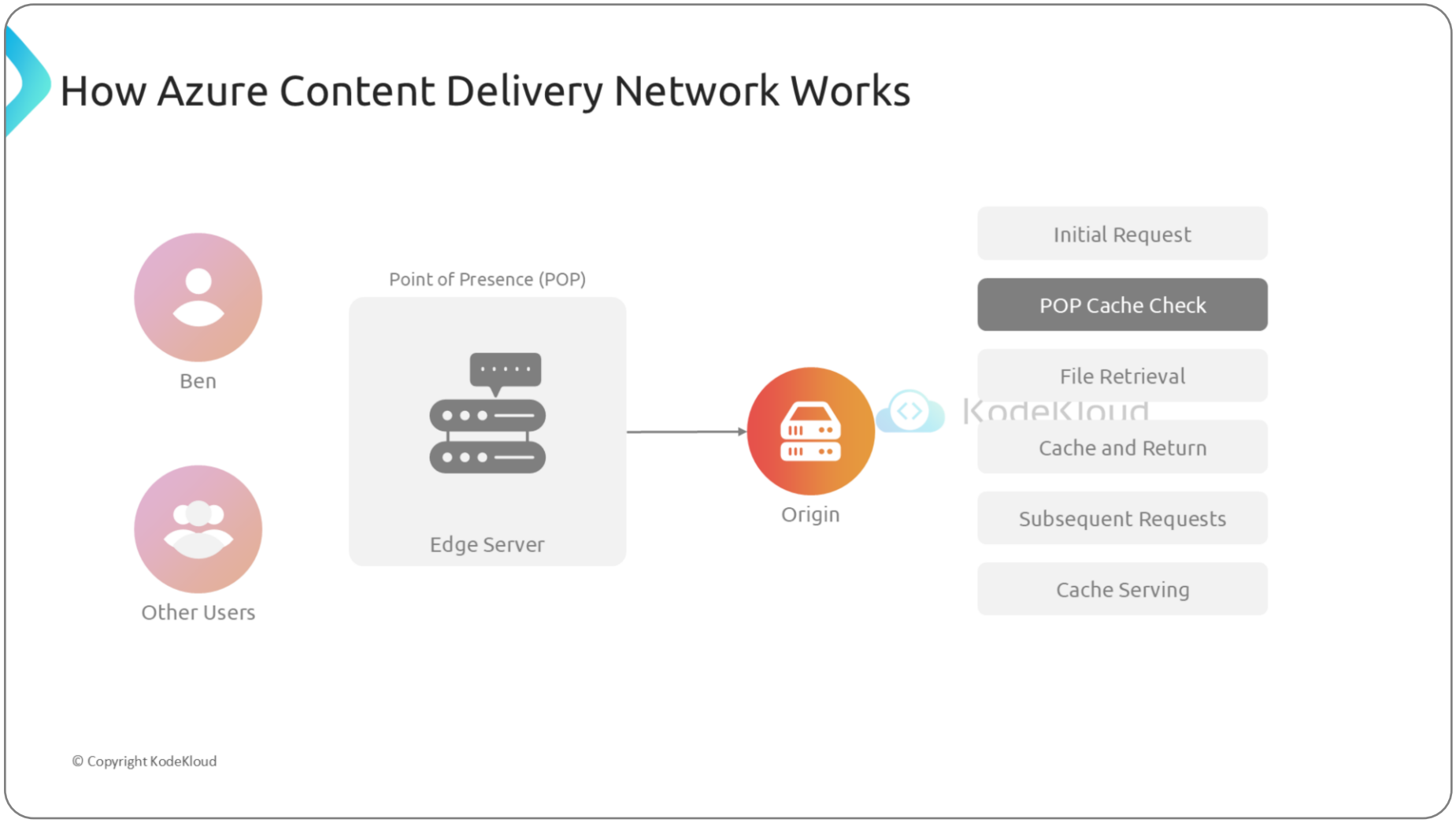
Azure Content Delivery Networks – Benefits

Benefits
Enhanced Performance: Improves user experience by delivering content faster.
Scalability: Efficiently handles sudden high traffic loads.
Traffic Distribution: Reduces load on the origin server by serving content directly from edge servers.

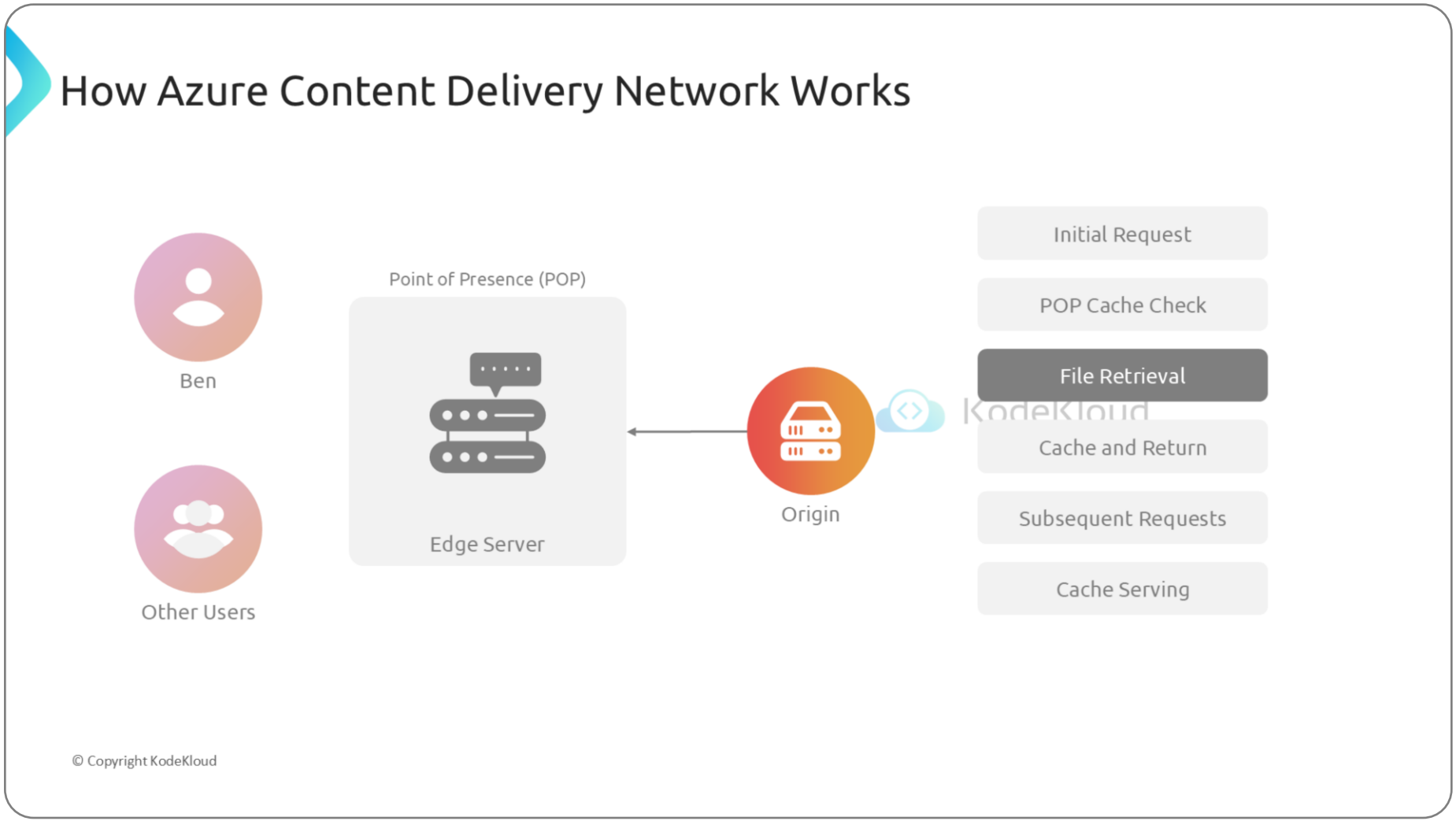


How Azure Content Delivery Network Works?

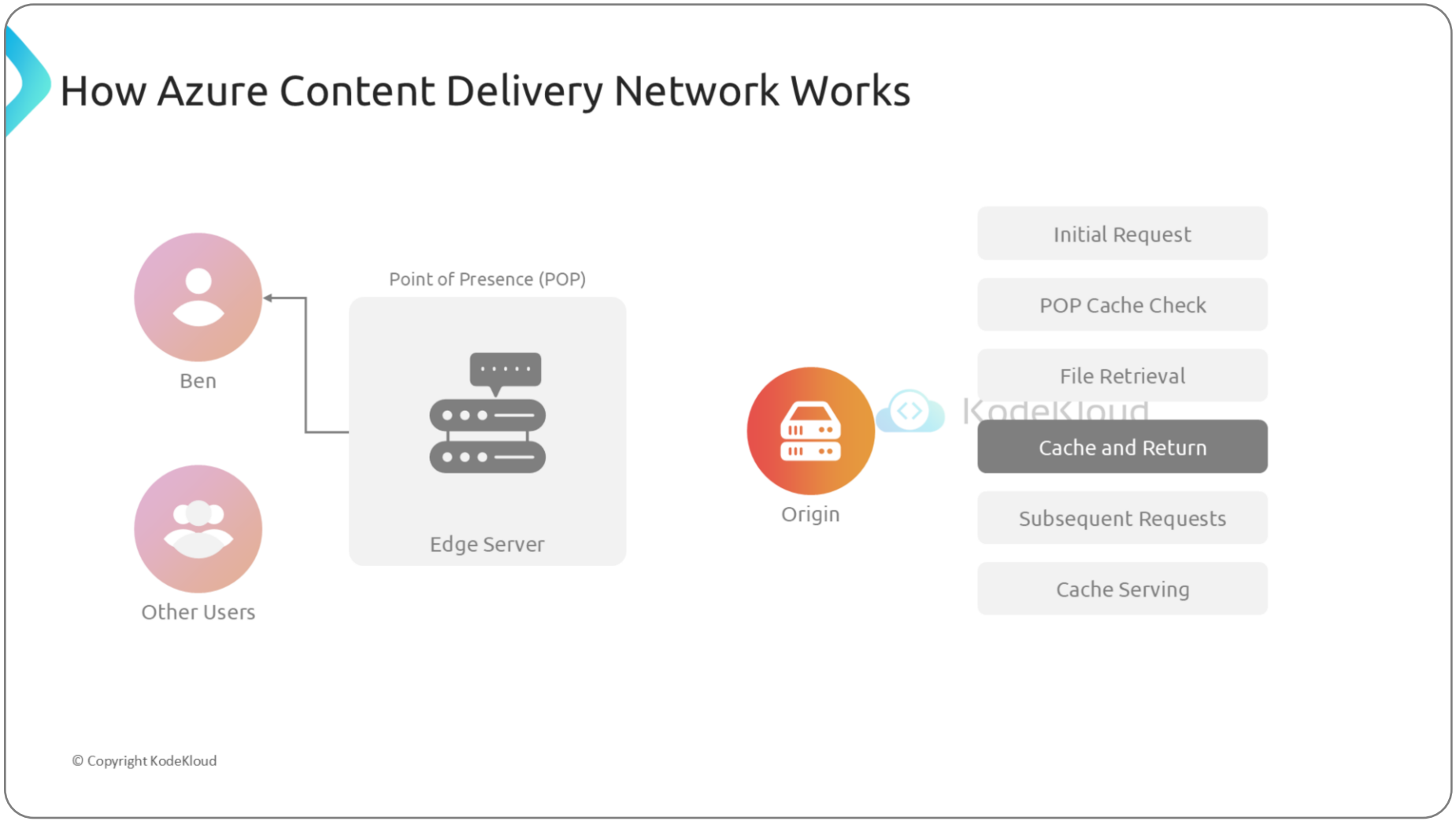
Initial Request: Ben requests a file using a URL with a special domain name, such as <endpoint name>.azureedge.net.



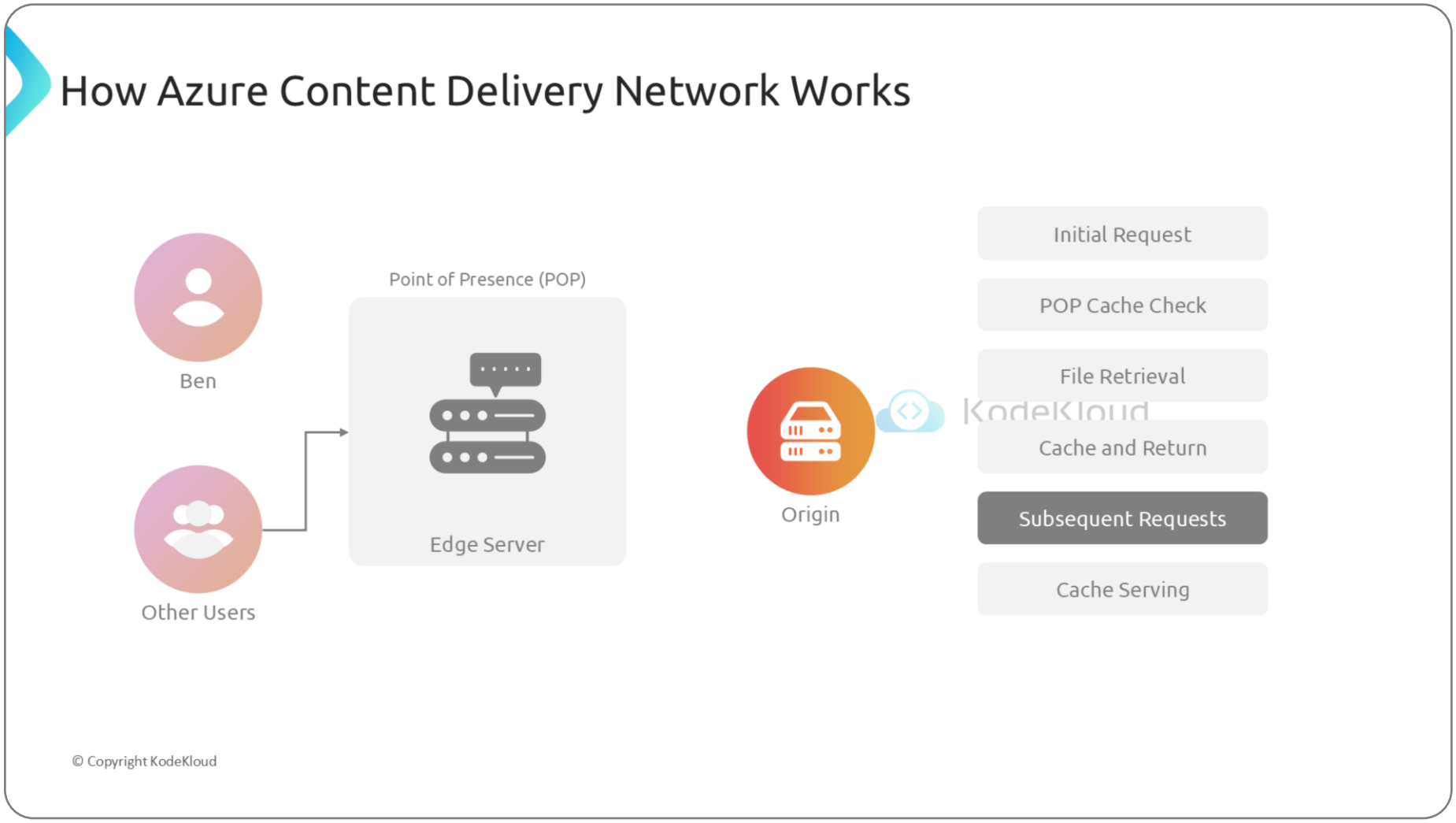
POP Cache Check: If edge servers in the Point of Presence (POP) don’t have the file cached, the POP requests the file from the origin server.



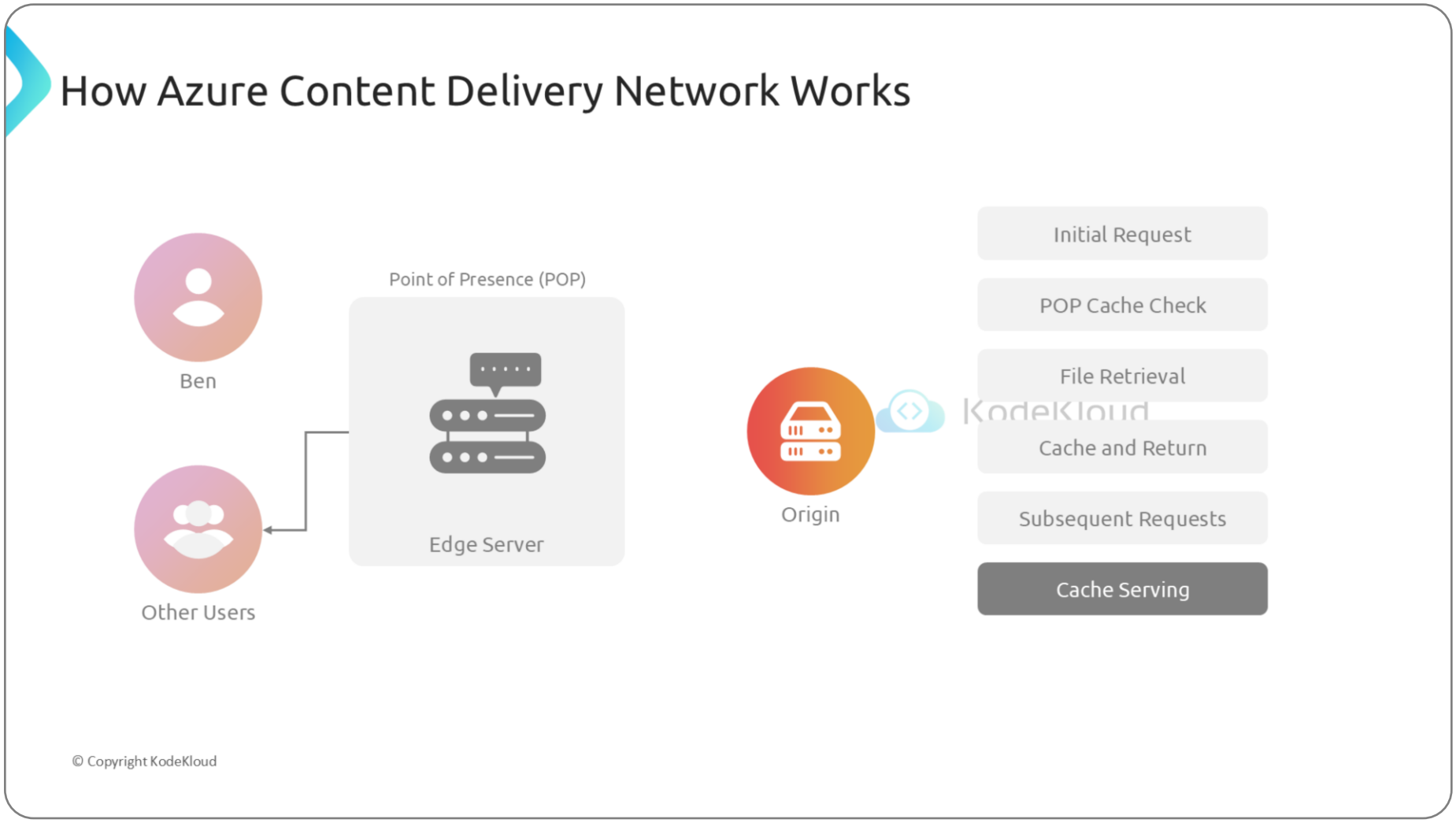
File Retrieval: The origin server sends the file to the edge server in the POP.



Cache and Return: The edge server caches the file and returns it to Alice.



Subsequent Requests: Additional users can request the same file using the same URL, and the file is served from the cache.



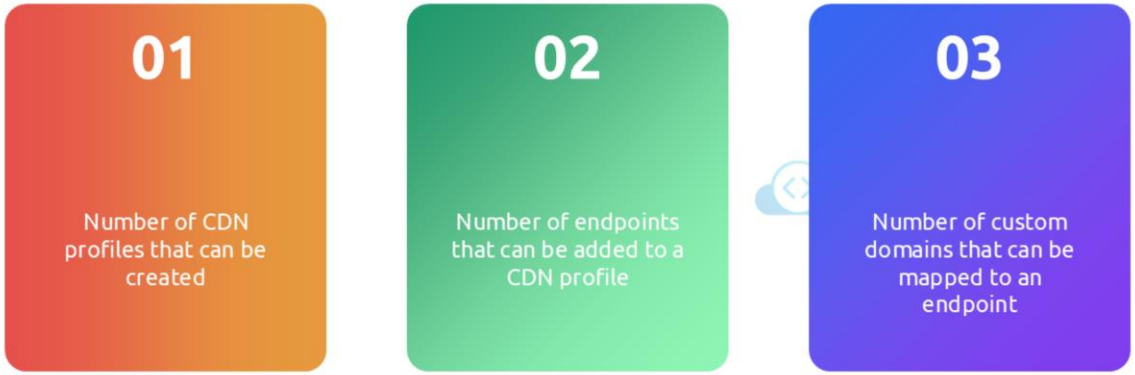
Cache Serving: If the Time-To-Live (TTL) for the file hasn't expired, the POP edge server continues to serve the file directly from the cache.

Azure Content Delivery Networks – Requirements

Requirements

- At least one CDN profile must be created, which is a collection of CDN endpoints.
- Each CDN endpoint represents a specific configuration of content delivery behavior and access.
- Multiple profiles can be used to organize CDN endpoints by internet domain, web application, or other criteria.

Azure Content Delivery Networks – Limitations

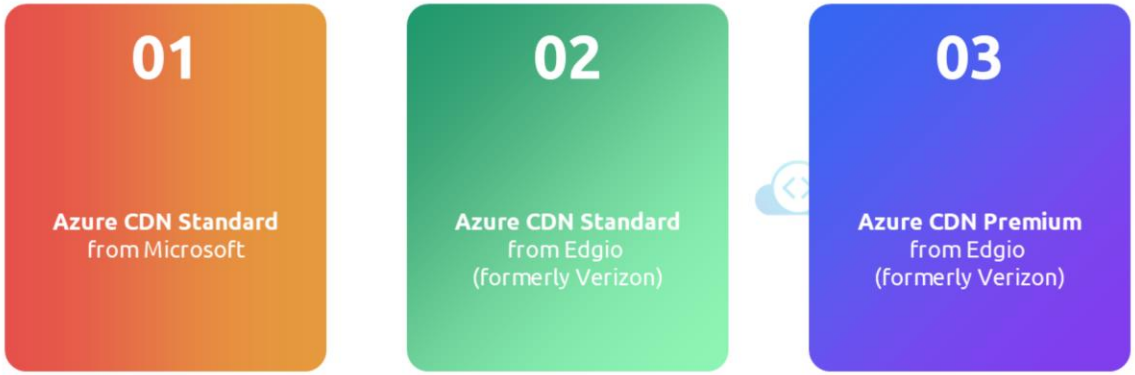


Limitations

Subscription Limits: Each Azure subscription has default limits on the following:

- Number of CDN profiles that can be created.
- Number of endpoints that can be added to a CDN profile.
- Number of custom domains that can be mapped to an endpoint.

Azure Content Delivery Networks – Products

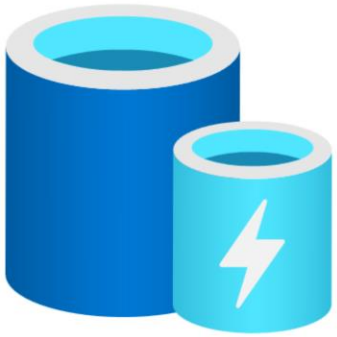


Azure Content Delivery Network (CDN) includes three products:

- Azure CDN Standard from Microsoft
- Azure CDN Standard from Edgio (formerly Verizon)
- Azure CDN Premium from Edgio (formerly Verizon)

Controlling Caching Behavior

Controlling Caching Behavior



Efficient Content Delivery

Cache Management

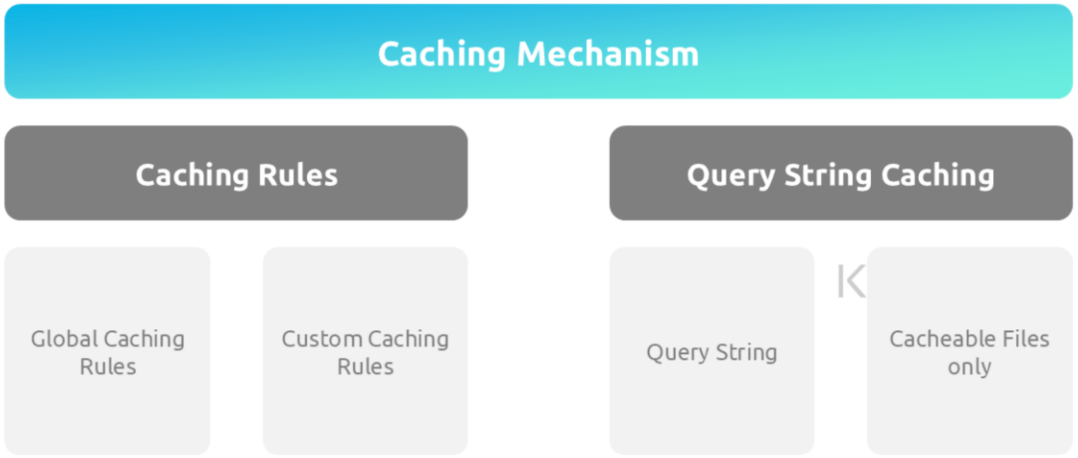
Prevents Outdated Resources

Controlling caching behavior in any content delivery system Caching plays a vital role in ensuring efficient content delivery However, cache management is necessary to ensure the freshness of content and avoid outdated resources being served to users

Controlling Caching Behavior

- Caching Importance: Since cached resources can become outdated, it’s crucial to manage when and how content is refreshed.
- Efficiency: To optimize time and bandwidth, cached resources are not compared to the origin server’s version with every access.
- Freshness Assumption: A cached resource deemed "fresh" is considered the most current version and is served directly to the client.
- Freshness Criteria: A resource is considered "fresh" when its age is below the threshold defined by the cache settings.

Controlling Caching Behavior



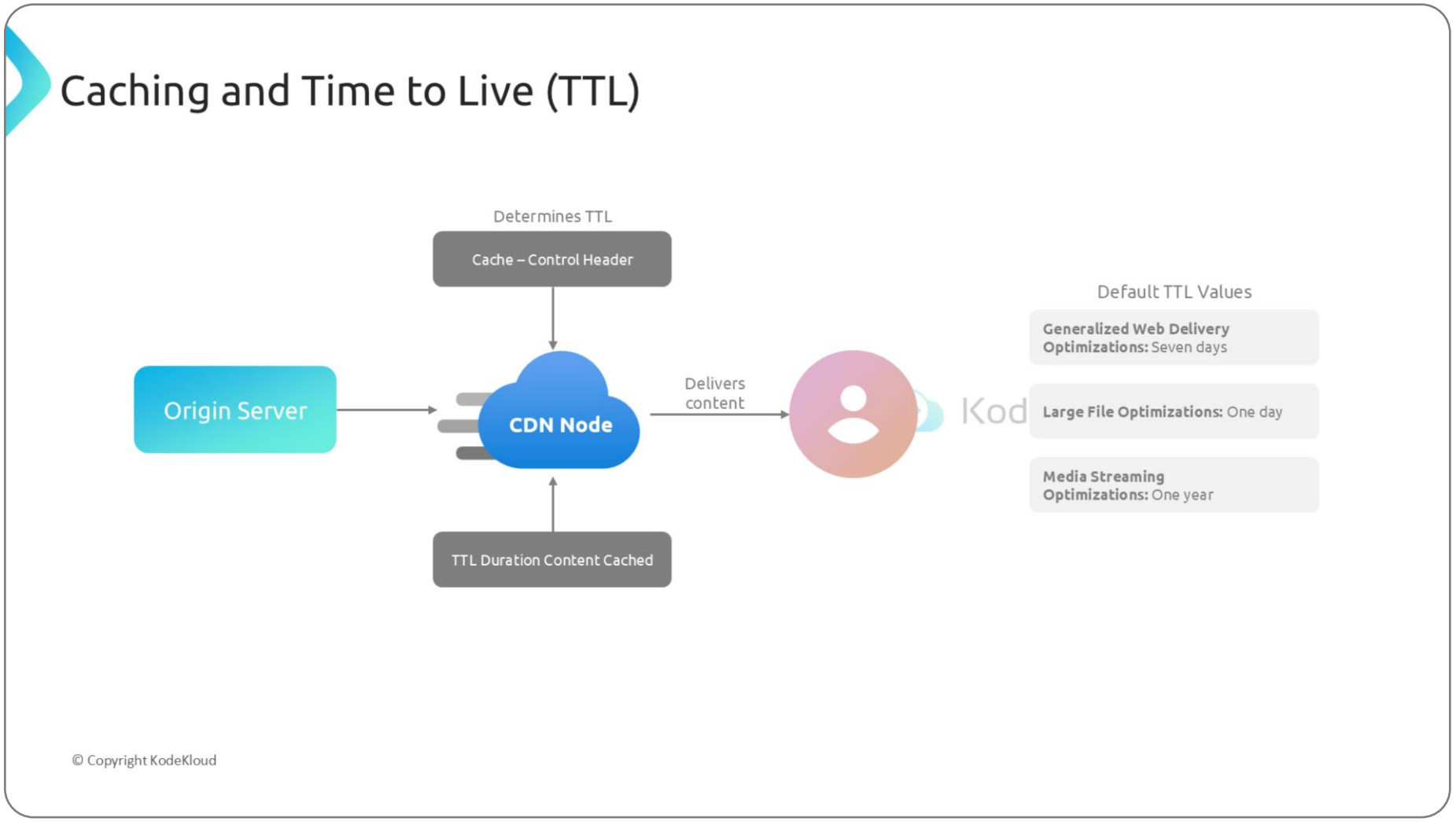
Controlling Caching Behavior
Caching Mechanisms: Azure CDNs offer two main caching mechanisms: caching rules and query string caching. The availability of these settings depends on the CDN tier selected.

Caching Rules: Azure CDN provides both global and custom caching rules.

Global Caching Rules: Set a global caching rule for each endpoint in your profile, impacting all requests to that endpoint.

Custom Caching Rules: Configure one or more custom caching rules for each endpoint in your profile. These rules override the global caching rule, if applied.

Query String Caching: Modify how Azure CDN handles caching for requests that include query strings. If a file isn’t cacheable, the query string caching setting will not have any effect, as it is governed by caching rules and CDN default behaviors.



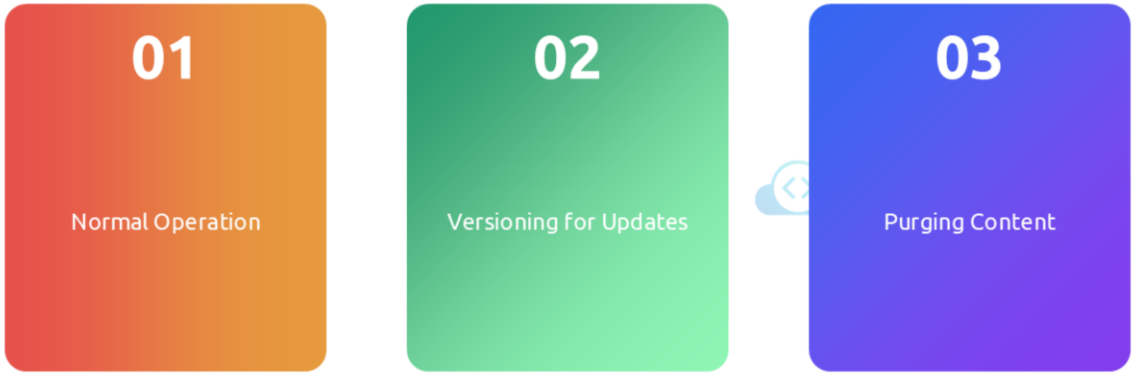
Caching and Time to Live (TTL)

TTL Management: When you publish a website through Azure CDN, the files are cached until their Time to Live (TTL) expires. The Cache-Control header in the HTTP response from the origin server determines the TTL duration.

Default TTL Values: If no TTL is set on a file, Azure CDN applies a default value, which can be overridden by configuring caching rules in Azure. The default TTL values are:

Generalized Web Delivery Optimizations: Seven days
Large File Optimizations: One day
Media Streaming Optimizations: One year

Controlling Caching behavior – Content Update



Content Updating

Normal Operation:

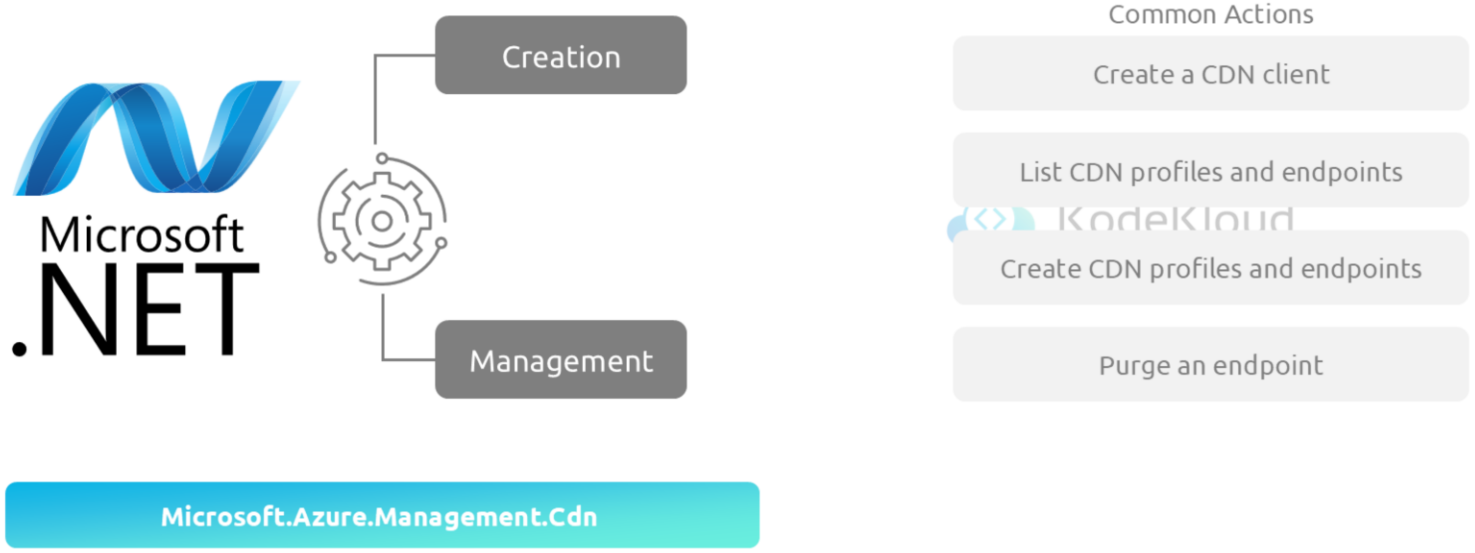
An Azure CDN edge node will serve an asset until its TTL expires.
Once the TTL expires and a client requests the same asset, the edge node reconnects to the origin server.
The node retrieves a fresh copy of the asset, resetting the TTL in the process.

Versioning for Updates: To ensure users always receive the latest version of an asset, consider adding a version string to the asset URL. This approach prompts the CDN to fetch the new asset immediately.

Purging Content: You can purge cached content from edge nodes, which forces the CDN to refresh the content on the next client request.

Using .NET Azure Content Delivery Network Library

Interacting With Azure Content Delivery Networks by Using .NET



You can use the Azure CDN Library for .NET to automate the creation and management of CDN profiles and endpoints.

Install the Microsoft.Azure.Management.Cdn package directly from the Visual Studio Package Manager Console or via the .NET Core CLI.

Common Actions:
•Create a CDN client

- List CDN profiles and endpoints
- Create CDN profiles and endpoints
- Purge an endpoint

Interacting With Azure Content Delivery Networks by Using .NET

Create a CDN client

```
static void Main(string[] args)
{
    // Create CDN client
    CdnManagementClient cdn = new CdnManagementClient(new TokenCredentials(authResult.AccessToken))
        { SubscriptionId = subscriptionId };
}
```



```
static void Main(string[] args)
{
   // Create CDN client
   CdnManagementClient cdn = new CdnManagementClient(new TokenCredentials(authResult.AccessToken))
      { SubscriptionId = subscriptionId };
}
```

Interacting With Azure Content Delivery Networks by Using .NET

List CDN profiles and endpoints

```
private static void ListProfilesAndEndpoints(CdnManagementClient cdn)
{
    // List all the CDN profiles in this resource group
    var profileList = cdn.Profiles.ListByResourceGroup(resourceGroupName);
    foreach (Profile p in profileList)
    {
        Console.WriteLine("CDN profile {0}", p.Name);

        //List all the CDN endpoints on this CDN profile
        Console.WriteLine("Endpoints:");
        var endpointList = cdn.Endpoints.ListByProfile(p.Name, resourceGroupName);
        foreach (Endpoint e in endpointList)
        {
            Console.WriteLine("-{0} ({1})", e.Name, e.HostName);
        }
        Console.WriteLine();
    }
}
```



```
private static void ListProfilesAndEndpoints(CdnManagementClient cdn)
{
   // List all the CDN profiles in this resource group
   var profileList = cdn.Profiles.ListByResourceGroup(resourceGroupName);
   foreach (Profile p in profileList)
   {
      Console.WriteLine("CDN profile {0}", p.Name);
```

```
        //List all the CDN endpoints on this CDN profile
        Console.WriteLine("Endpoints:");
        var endpointList = cdn.Endpoints.ListByProfile(p.Name, resourceGroupName);
        foreach (Endpoint e in endpointList)
        {
           Console.WriteLine("-{0} ({1})", e.Name, e.HostName);
        }
        Console.WriteLine();
    }
}
```

KodeKloud

Visit www.kodekloud.com to learn more.