CLOUD COMPUTING SYSTEMS

Lab 4

Nuno Preguiça, João Resende

(nuno.preguica_at_fct.unl.pt, jresende_at_fct.unl.pt)

GOAL

In the end of this lab you should be able to:

- Create a Azure Redis for Cache account @ Azure;
- Use AzureRedisCache to store cache data.

- More professional deployment.

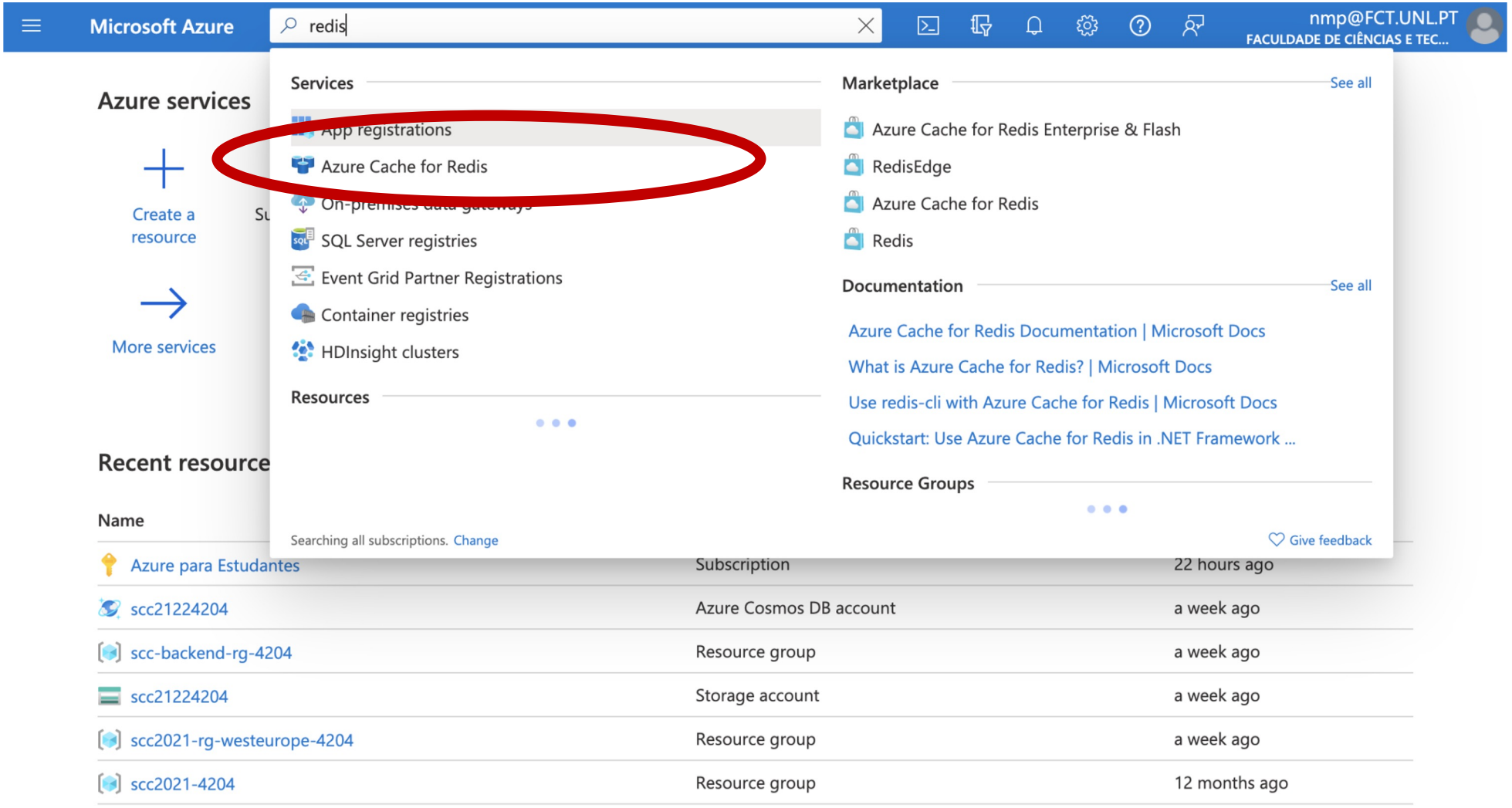
GOAL

In the end of this lab you should be able to:

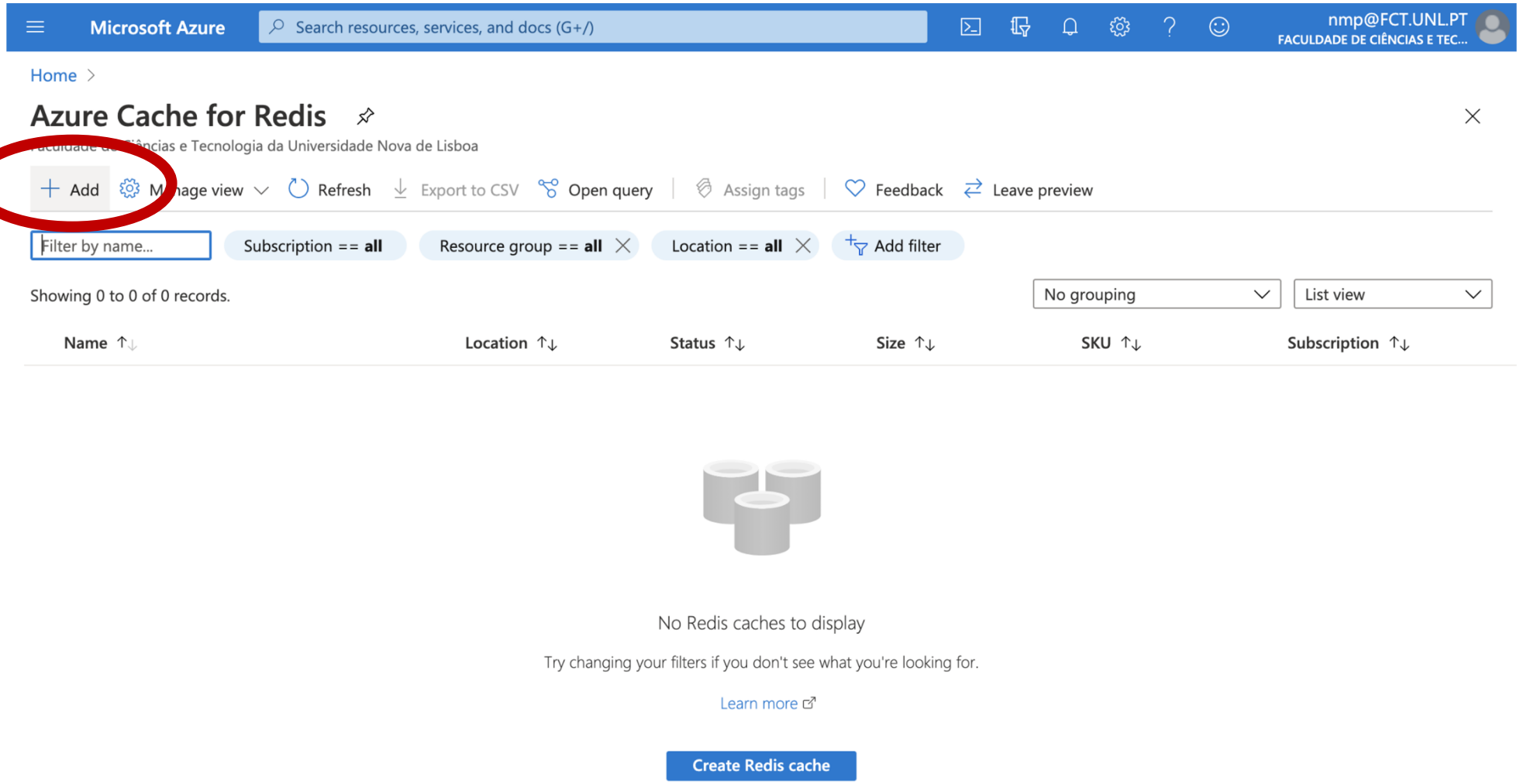
- **Create a Azure Redis for Cache account @ Azure;**
- Use AzureRedisCache to store cache data.

- More professional deployment.

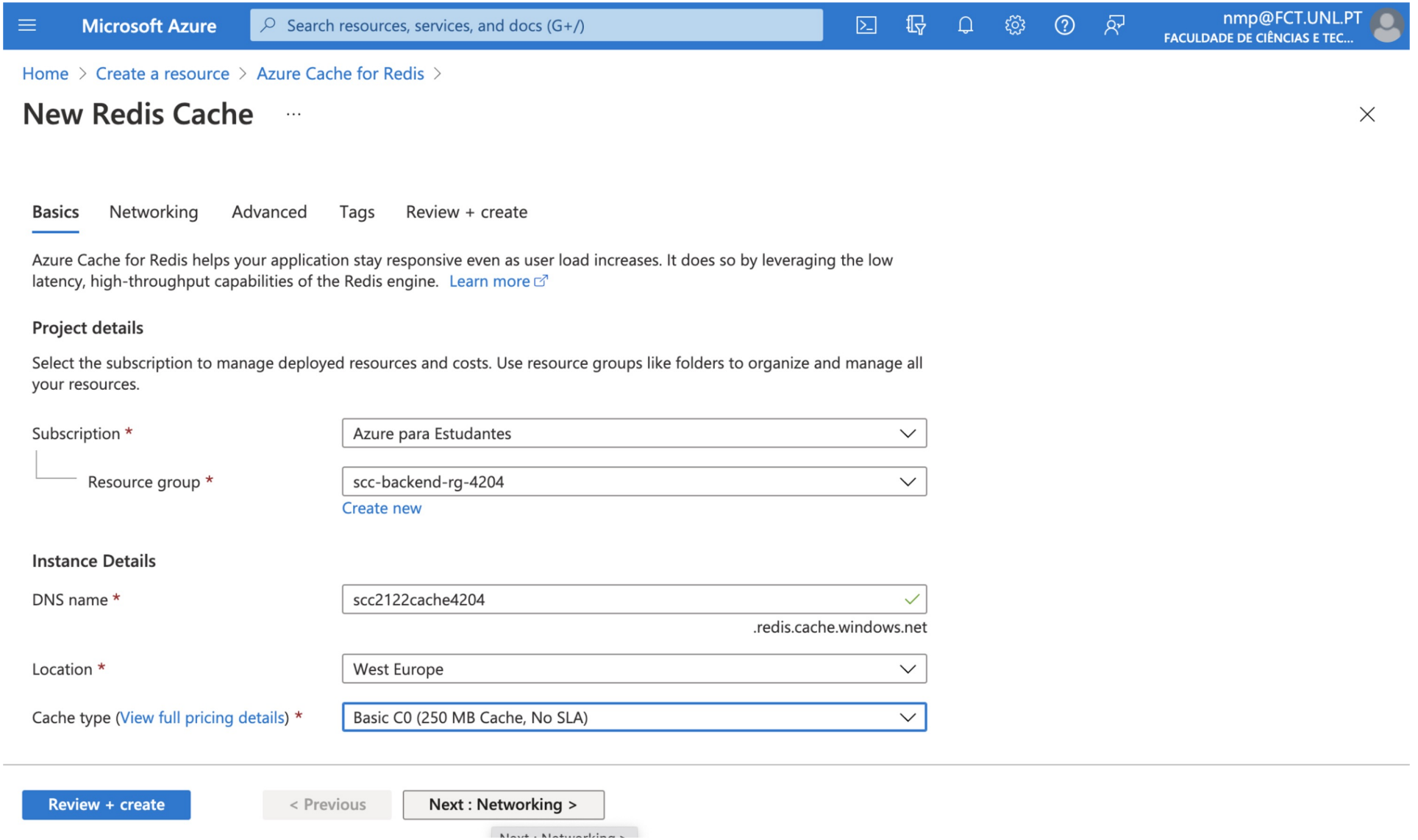
Create Azure Redis for Cache



CREATE AZURE REDIS FOR CACHE (2)

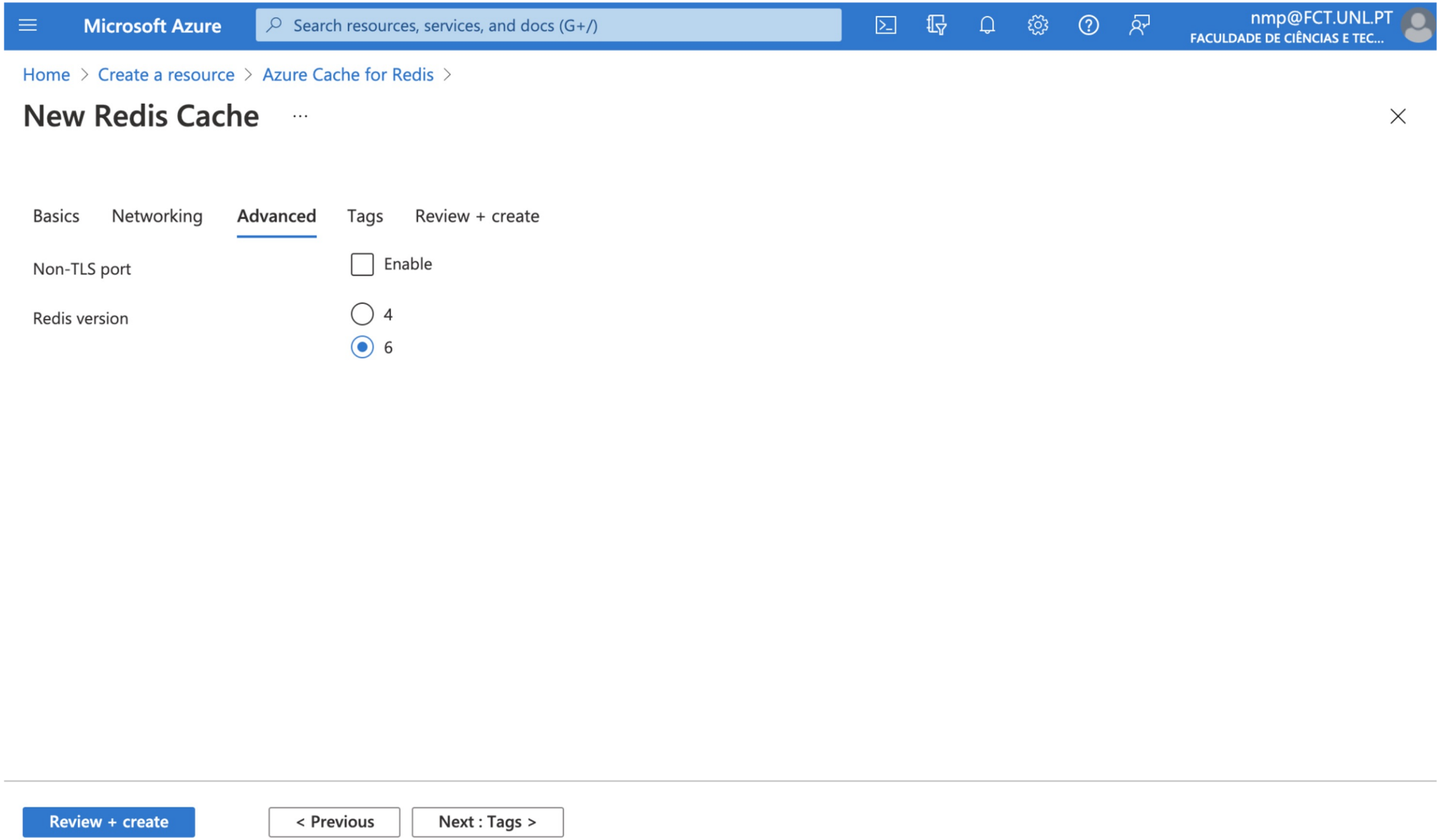


CREATE AZURE REDIS FOR CACHE (3)

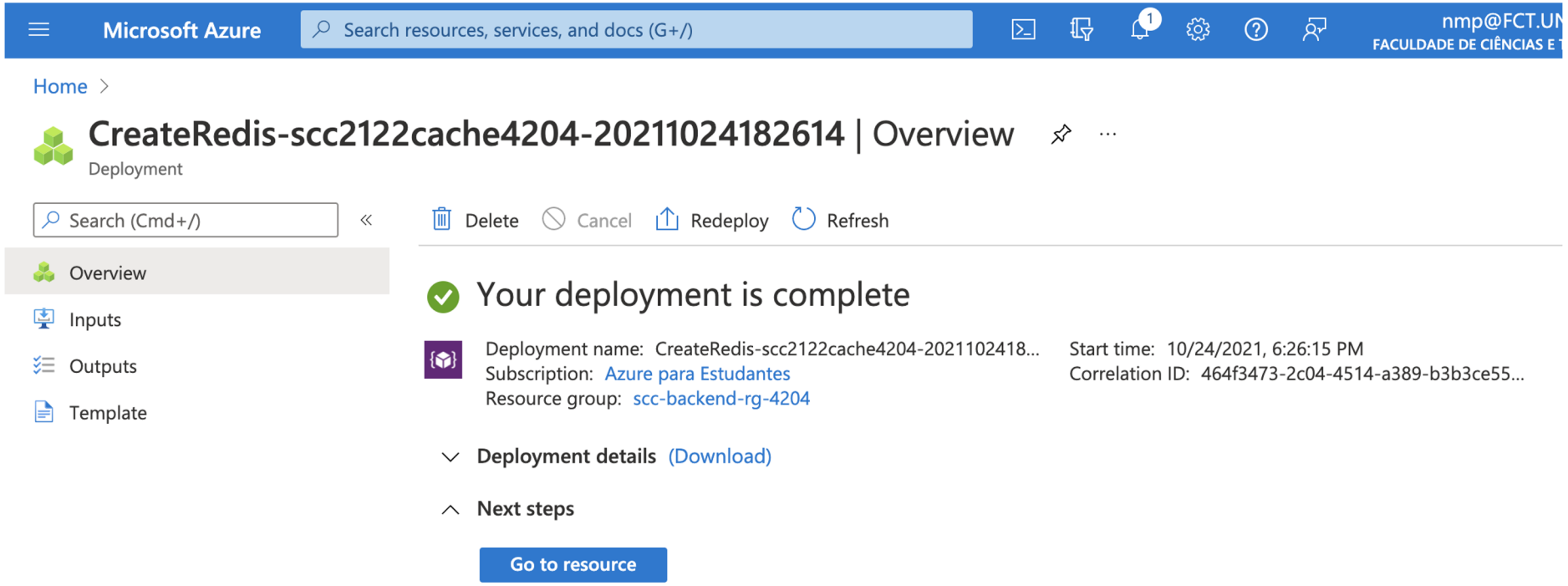


Check features and pricing at:
https://azure.microsoft.com/pt-pt/pricing/details/cache/

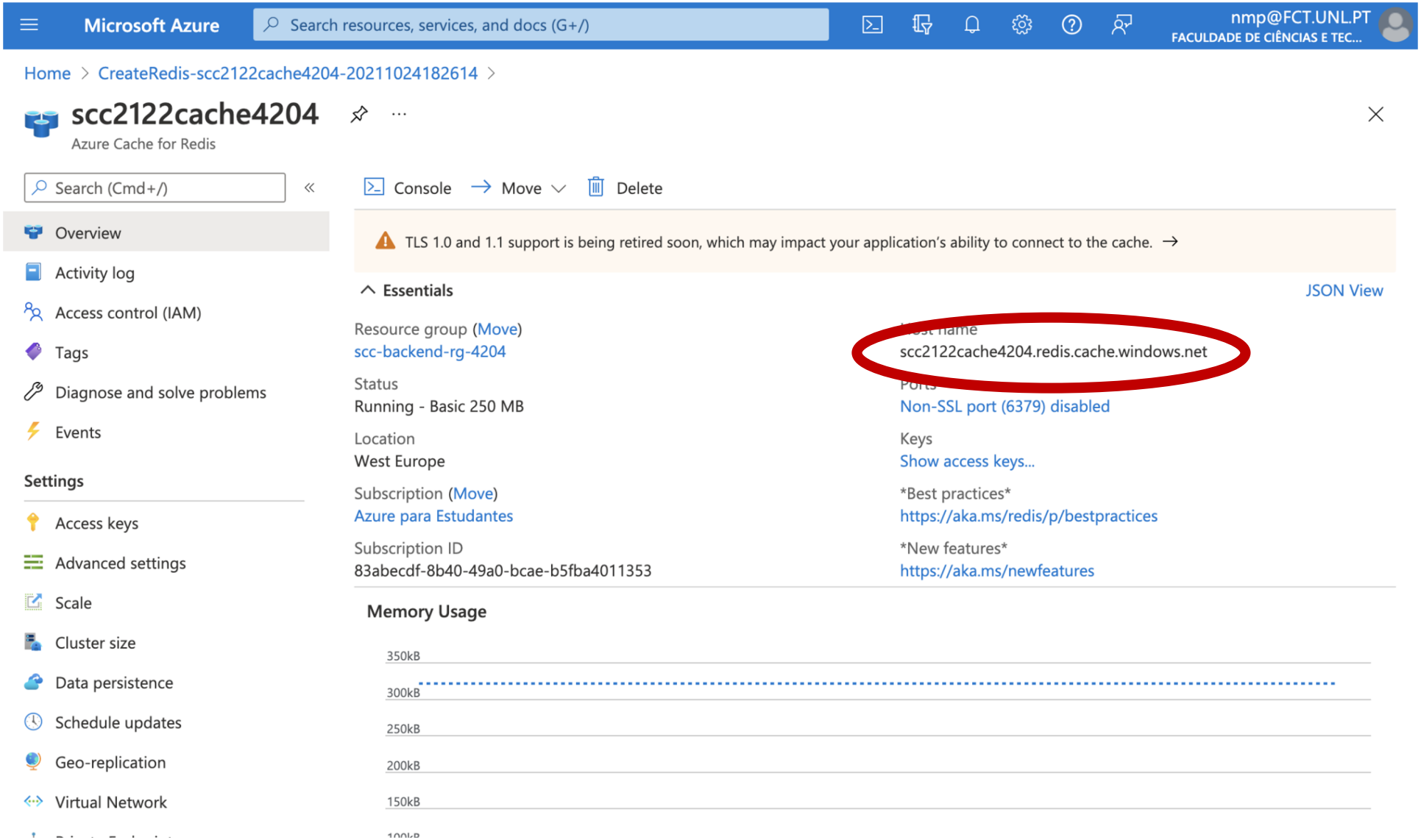
Create Azure Redis for Cache (4)



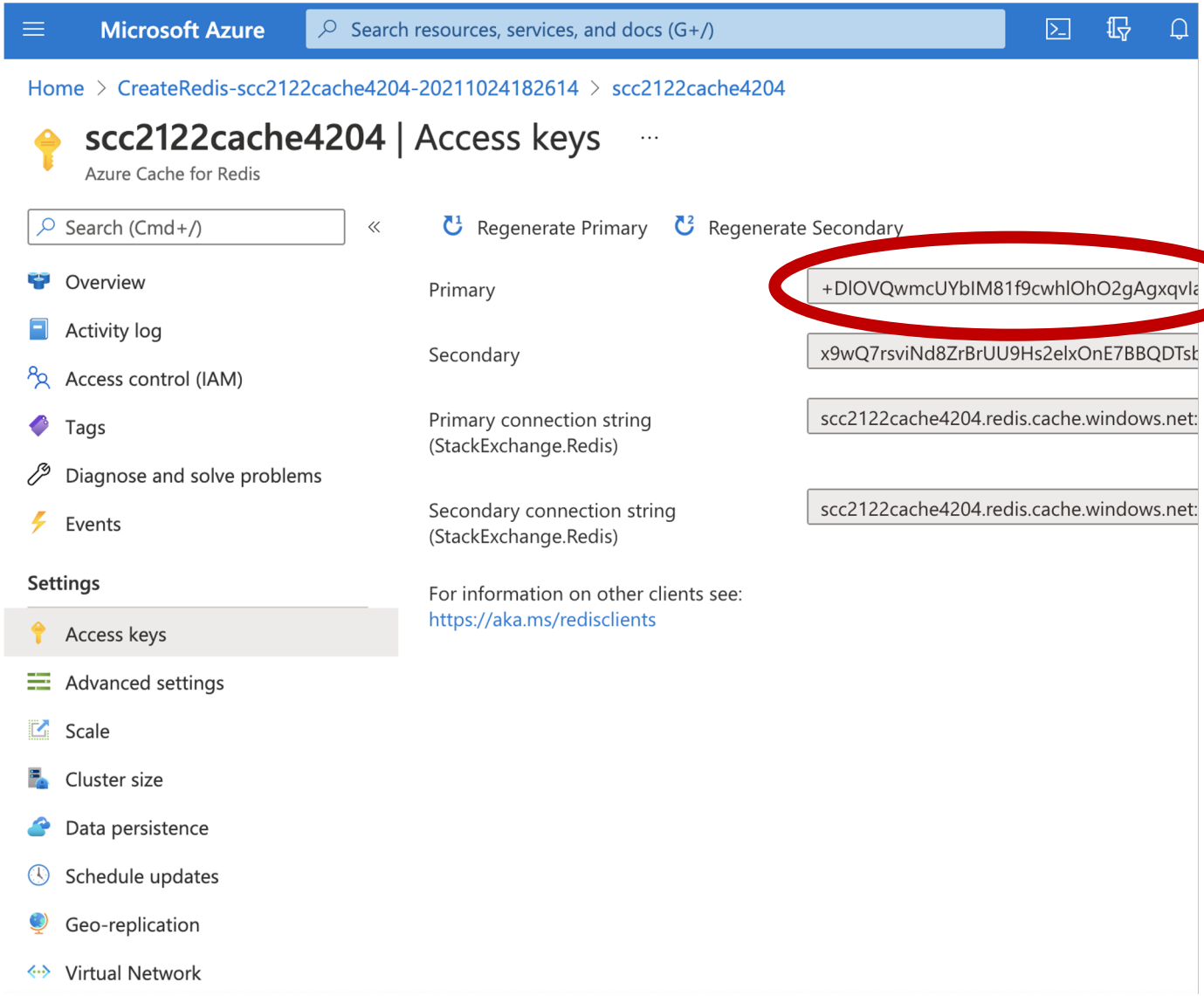
CREATE AZURE REDIS FOR CACHE (5)



Create Azure Redis for Cache: URL



CREATE AZURE REDIS FOR CACHE: KEY



GOAL

In the end of this lab you should be able to:

- Create a Azure Redis for Cache account @ Azure;
- **Use AzureRedisCache to store cache data.**

- More professional deployment.

Accessing Azure Cache for Redis: useful links

We will use the library provided by Redis.

Java Docs available at:

https://www.javadoc.io/doc/redis.clients/jedis/4.2.3

Overview on how to use at:

https://docs.microsoft.com/en-us/azure/azure-cache-for-redis/cache-java-get-started

MAVEN DEPENDENCIES

```
<dependency>
        <groupId>redis.clients</groupId>
        <artifactId>jedis</artifactId>
        <version>4.2.3</version>
</dependency>
<dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-core</artifactId>
        <version>2.13.4</version>
</dependency>
<dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-databind</artifactId>
        <version>2.13.4</version>
</dependency>
```

Step 1: create client to Redis (1)

Alternatives:

1. Create a simple client to Redis. A client has a connection to the Redis server and it is not thread-safe.

- Not a good option for using in an application server.

2. Use a pool of clients to Redis, that can be shared by multiple threads.

STEP 1: CREATE CLIENT TO REDIS (2)

```
private static final String RedisHostname = ... ;
private static final String RedisKey = ...;

private static JedisPool instance;

public synchronized static JedisPool getCachePool() {
   if( instance != null)
            return instance;
   final JedisPoolConfig poolConfig = new JedisPoolConfig();
   poolConfig.setMaxTotal(128);
   poolConfig.setMaxIdle(128);
   poolConfig.setMinIdle(16);
   ...
   poolConfig.setBlockWhenExhausted(true);
   instance = new JedisPool(poolConfig, RedisHostname, 6380, 1000, RedisKey, true);
   return instance;
}
```

new JedisPool(config, hostname, port, timeout, key, TLS)

STEP 1: CREATE CLIENT TO REDIS (2)

```
try (Jedis jedis = RedisCache.getCachePool().getResource()) {

        // Execute operations with a client to the Redis server

}
```

Step 2: Set/get a value to a key

```
ObjectMapper mapper = new ObjectMapper();
try (Jedis jedis = RedisCache.getCachePool().getResource()) {
    jedis.set("user:"+id, mapper.writeValueAsString(u));
    String res = jedis.get("user:"+id);
    System.out.println(res);

    Long cnt = jedis.lpush("MostRecentUsers", mapper.writeValueAsString(u));
    if (cnt > 5)
        jedis.ltrim("MostRecentUsers", 0, 4);

    List<String> lst = jedis.lrange("MostRecentUsers", 0, -1);
    for( String s : lst)
            System.out.println(s);

    cnt = jedis.incr("NumUsers");
    System.out.println( "Num users : " + cnt);
}
```

STEP 2: ADD ELEMENT TO LIST AND KEEP LIMIT

```
ObjectMapper mapper = new ObjectMapper();
try (Jedis jedis = RedisCache.getCachePool().getResource()) {
    jedis.set("user:"+id, mapper.writeValueAsString(u));
    String res = jedis.get("user:"+id);
    System.out.println(res);

    Long cnt = jedis.lpush("MostRecentUsers", mapper.writeValueAsString(u));
    if (cnt > 5)
        jedis.ltrim("MostRecentUsers", 0, 4);

    List<String> lst = jedis.lrange("MostRecentUsers", 0, -1);
    for( String s : lst)
            System.out.println(s);

    cnt = jedis.incr("NumUsers");
    System.out.println( "Num users : " + cnt);
}
```

NOTE: this is just an example for the use of a list... in your project it probably does not make sense to have a list of Users.

Step 2: Get the elements of a list

```
ObjectMapper mapper = new ObjectMapper();
try (Jedis jedis = RedisCache.getCachePool().getResource()) {
    jedis.set("user:"+id, mapper.writeValueAsString(u));
    String res = jedis.get("user:"+id);
    System.out.println(res);

    Long cnt = jedis.lpush("MostRecentUsers", mapper.writeValueAsString(u));
    if (cnt > 5)
        jedis.ltrim("MostRecentUsers", 0, 4);

    List<String> lst = jedis.lrange("MostRecentUsers", 0, -1);
    for( String s : lst)
            System.out.println(s);

    cnt = jedis.incr("NumUsers");
    System.out.println( "Num users : " + cnt);
}
```

Step 2: Increment and get the value of a counter

```
ObjectMapper mapper = new ObjectMapper();
try (Jedis jedis = RedisCache.getCachePool().getResource()) {
    jedis.set("user:"+id, mapper.writeValueAsString(u));
    String res = jedis.get("user:"+id);
    System.out.println(res);

    Long cnt = jedis.lpush("MostRecentUsers", mapper.writeValueAsString(u));
    if (cnt > 5)
        jedis.ltrim("MostRecentUsers", 0, 4);

    List<String> lst = jedis.lrange("MostRecentUsers", 0, -1);
    for( String s : lst)
            System.out.println(s);

    cnt = jedis.incr("NumUsers");
    System.out.println( "Num users : " + cnt);
}
```

REDIS DATATYPES

https://redis.io/topics/data-types

- String
- List of strings (with insertion on head or tail)
- Set of strings
- Sorted set of strings
- Hashes (similar to a struct/map)
- Bit array
- HyperLogLogs (probabilistic data structure for estimating the number of elements in a set)
- …

USEFUL REDIS METHOD

- expire(key, seconds)

Allows to set an expire time for a key. After the time for the key expires, the key is automatically deleted from the cache.

SOME NOTES

You should use keys that are unique for a given data type.

What should you keep in cache?

- Everything that might get accessed later;
- In case of doubt, just keep everything… it is a cache: what is not being used gets discarded.

CODE PROVIDED

The code provided (lab4.zip) is a Maven project with a single class that stores a User object in a Redis list and the number of users created in a counter.

For testing it in the command line, just run:

```
mvn compile assembly:single
```

to compile and create a single file with all compiled classes and dependencies.

Run the program as follows:

```
java -cp target/scc2223-lab4-1.0-jar-with-dependencies.jar scc.utils.TestCache
```

TODO

Extend your backend to use the Azure Cache whenever that makes sense.

NOTE: in the end, do not forget to delete the Azure Cache for Redis resources.

GOAL

In the end of this lab you should be able to:

- Create a Azure Redis for Cache account @ Azure;
- Use AzureRedisCache to store cache data.

- **More professional deployment.**

PROBLEMS

- Creating resources manually is time consuming
- Setting keys in the code is bad practice

PROBLEMS

- **Creating resources manually is time consuming**
- Setting keys in the code is bad practice

AUTOMATED CREATION OF RESOURCES

Possible to create resources:

- Programatically using management API;
- Using az command to create scripts.

AUTOMATED CREATION OF RESOURCES USING MANAGEMENT API

Check scc-mgt.zip code at CLIP.

Code creates:

- Blob storage accounts, containers;
- Cosmos DB accounts, DB, containers;
- Redis servers.

Configure the code to change names, regions used, etc.

Check comment with TODO.

AUTOMATED CREATION OF RESOURCES USING MANAGEMENT API (2)

Program also create two file per regions – e.g. westeurope:

- azurekeys-westeurope.props
  - Property file with URLs and keys for created resources
- azureprops-westeurope.sh
  - Shell script with commands to update application settings for both app service and Azure functions.
  - After starting the web app, you can run the commands.

APPLICATION SETTINGS

Command to set application settings in Azure:

az functionapp config appsettings set --name **app_name** --resource-group **res_group_name** --settings "**PROP**=**VAL**"

PROBLEMS

- Creating resources manually is time consuming
- **Setting keys in the code is bad practice**

Application settings can be accessed using system environment variable – System.getenv(variable name)

ACCESSING KEY IN APPLICATION SETTINGS

```
// Get connection string in the storage access keys page
String storageConnectionString = System.getenv("BlobStoreConnection");

// Get container client
BlobContainerClient containerClient = new BlobContainerClientBuilder()
                                        .connectionString(storageConnectionString)
                                        .containerName("images")
                                        .buildClient();
```

CODE PROVIDED

The code provided (scc-mgt.zip) is a Maven project with a class that creates resources in Azure. Check TODO to configure for your needs.

For testing it in the command line, just run:

```
mvn compile assembly:single
```

to compile and create a single file with all compiled classes and dependencies.

Run the program as follows to create resources:

```
java -cp target/scc2223-mgt-1.0-jar-with-dependencies.jar scc.mgt.AzureManagement
```

Run the program as follows to delete resources:

```
java -cp target/scc2223-mgt-1.0-jar-with-dependencies.jar scc.mgt.AzureManagement -delete
```