CLOUD COMPUTING SYSTEMS

Lab 2

Nuno Preguiça, João Resende

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

GOAL

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

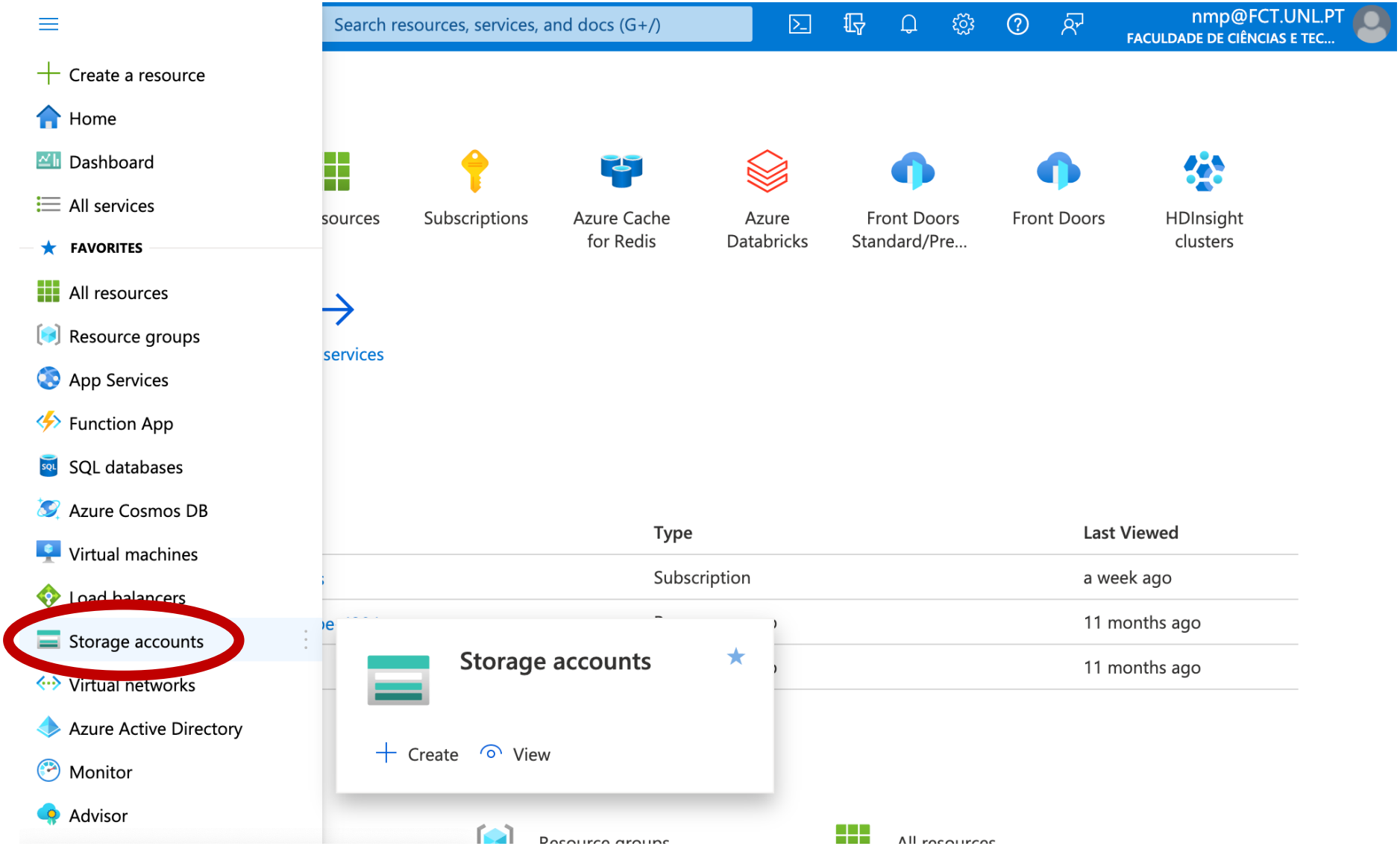
- Create a StorageAccount + Blob Container @ Azure;
- Complete the MediaResource, by storing data at Azure Blob Storage
- Testing web apps

GOAL

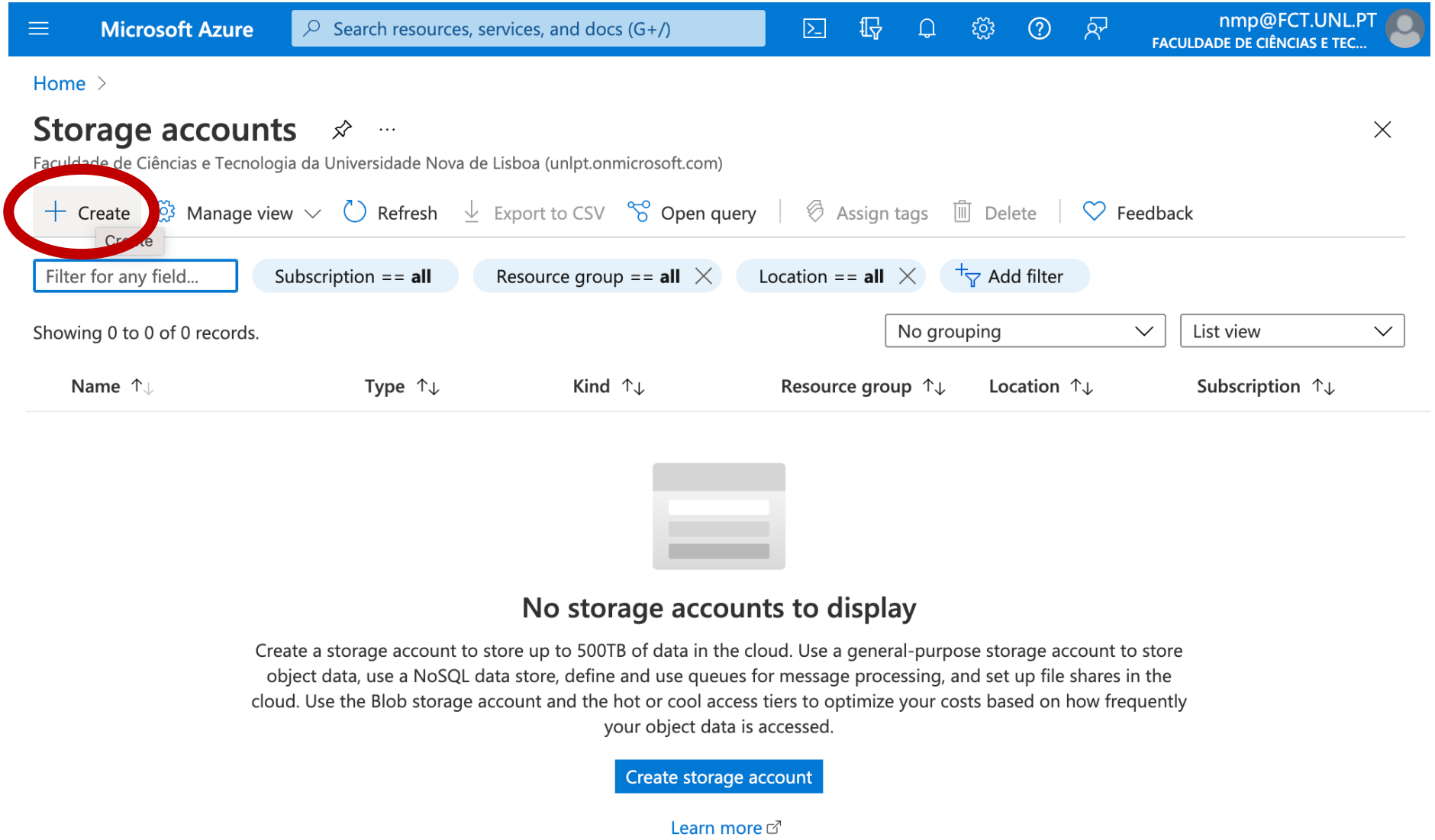
In the end of this lab you should be able to:

- **Create a StorageAccount + Blob Container @ Azure;**
- Complete the MediaResource, by storing data at Azure Blob Storage
- Testing web apps

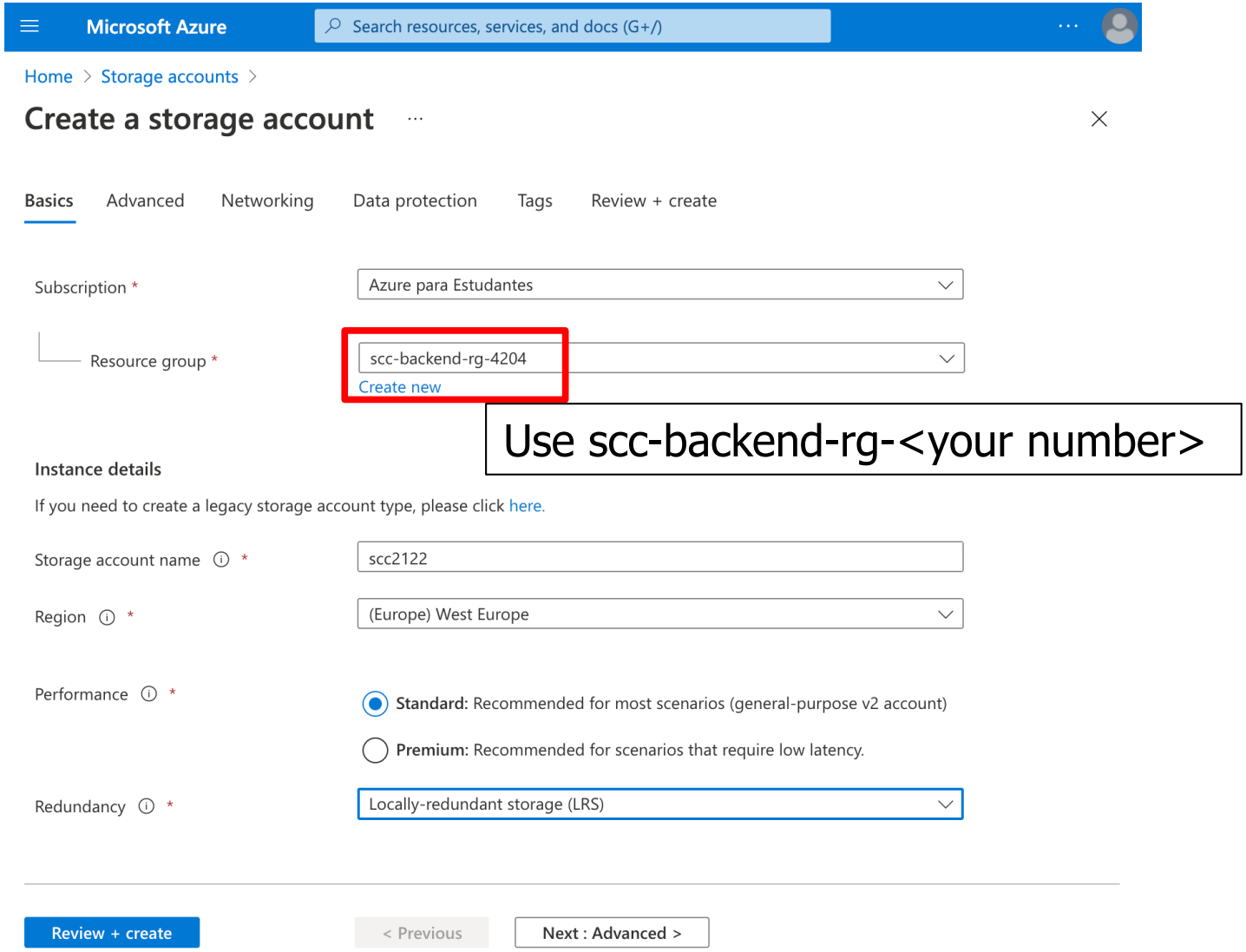
CREATE STORAGE ACCOUNT (1)



CREATE STORAGE ACCOUNT (2)

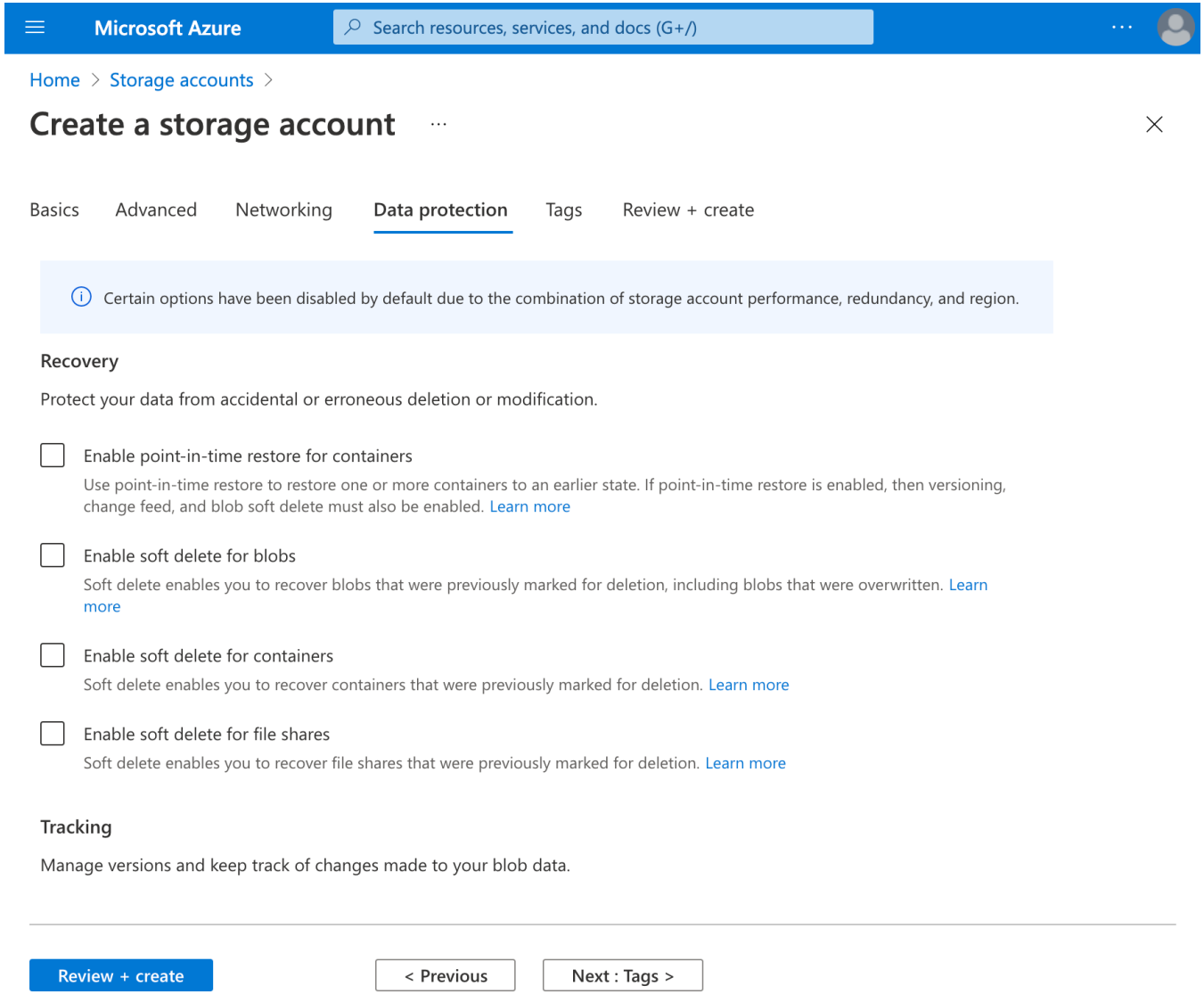


CREATE STORAGE ACCOUNT (3)



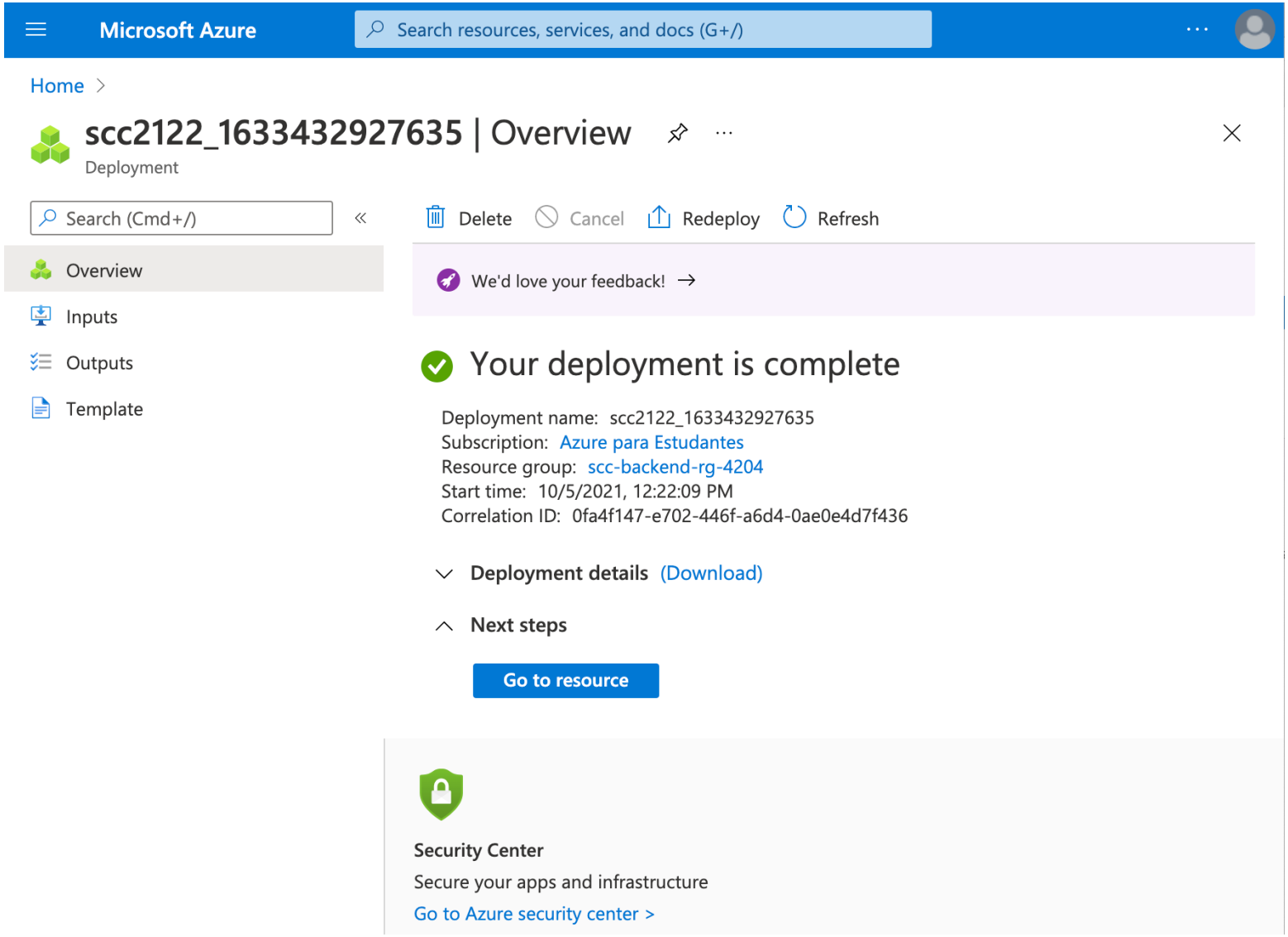
Check lecture 2 for info on these options.

CREATE STORAGE ACCOUNT (4)

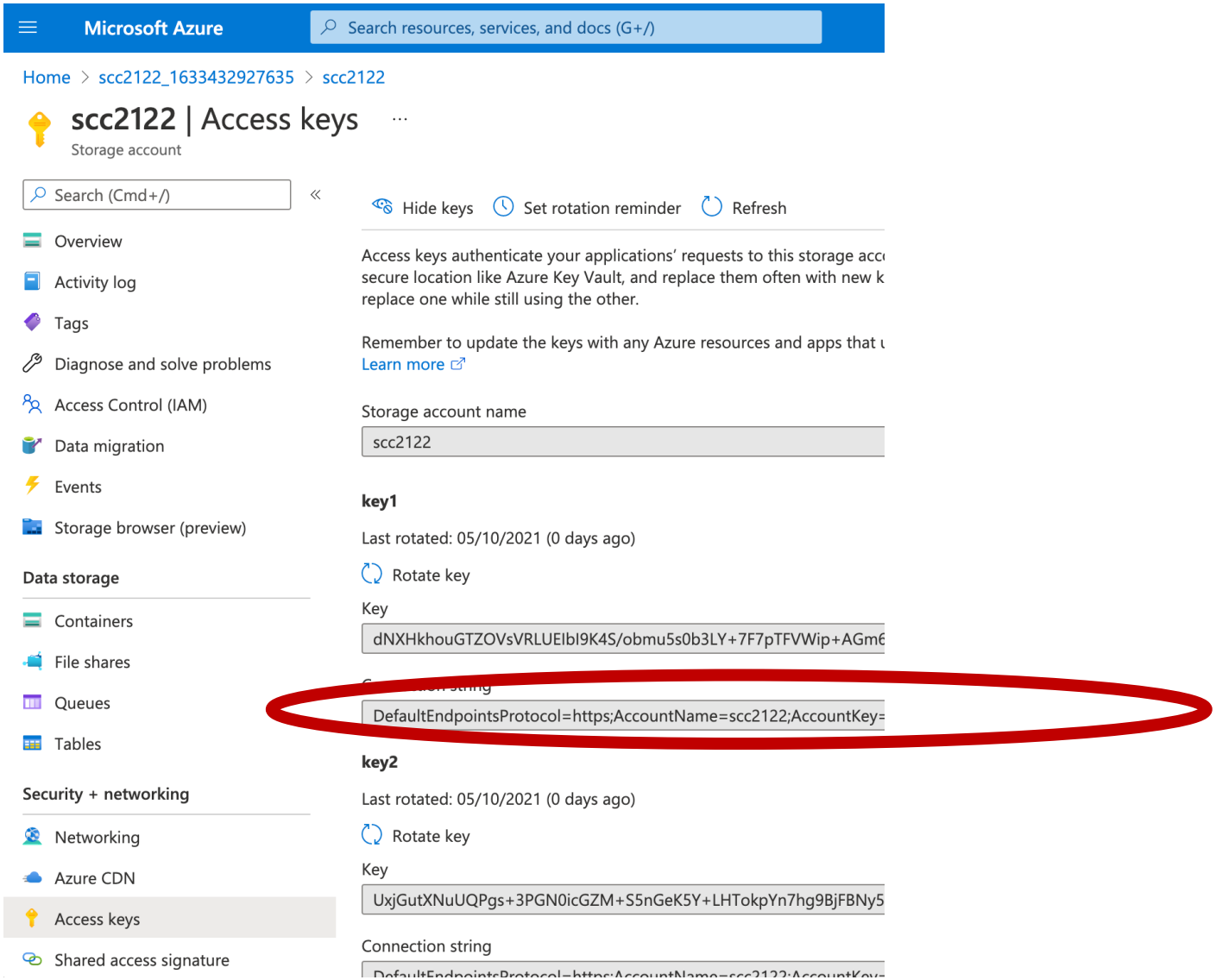


We will not need nothing of this...

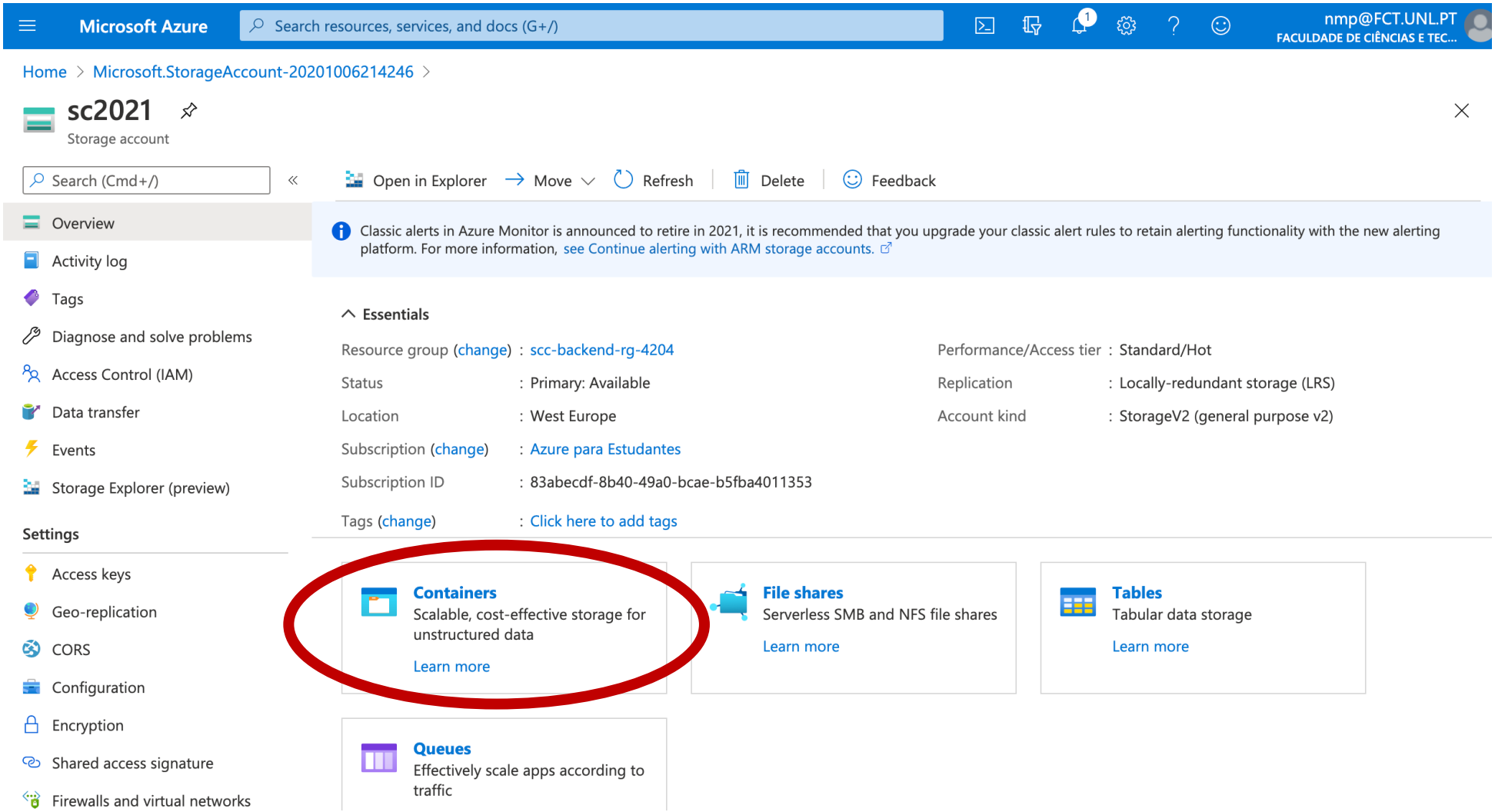
CREATE STORAGE ACCOUNT (5)



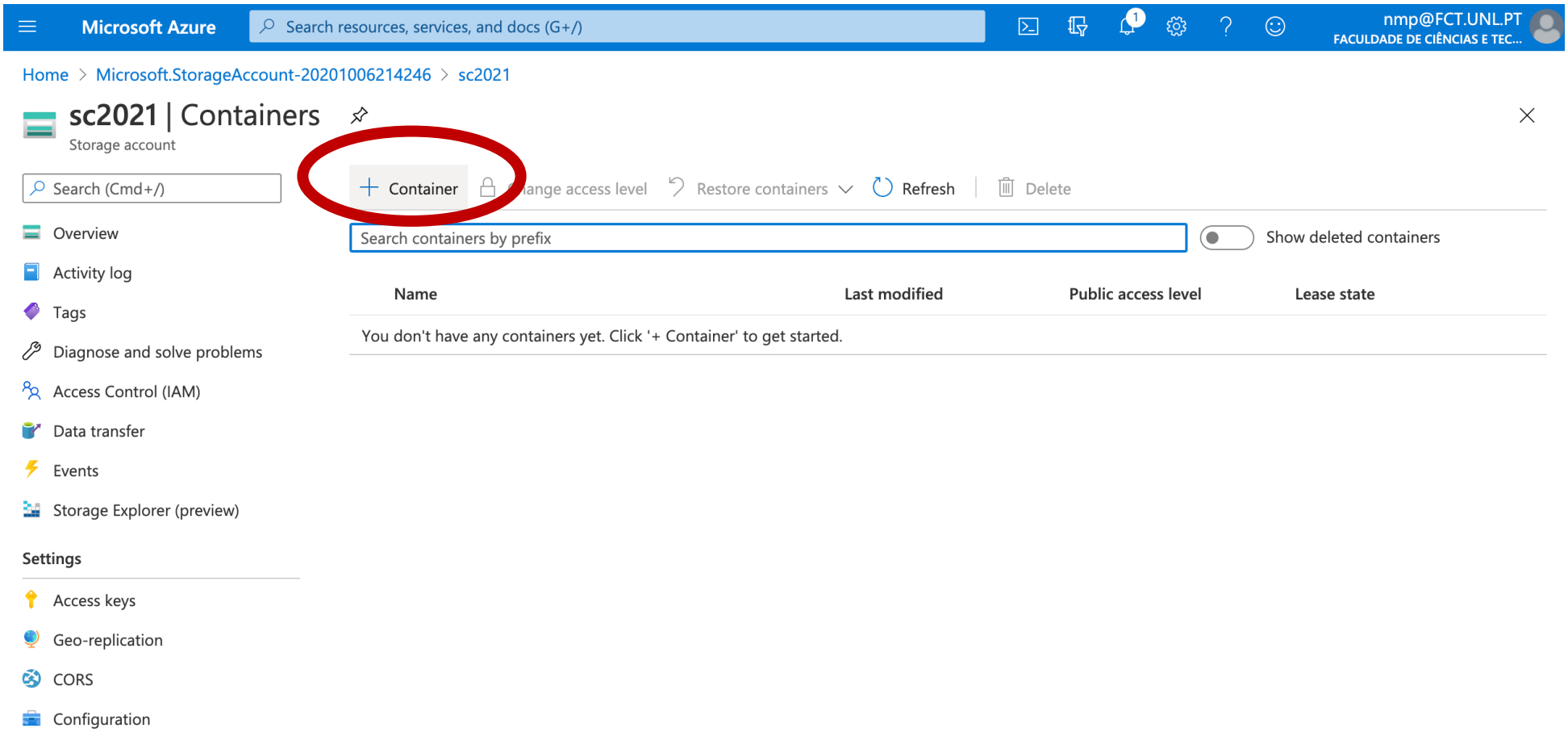
STORAGE ACCOUNT: ACCESS KEYS (FOR CODE)



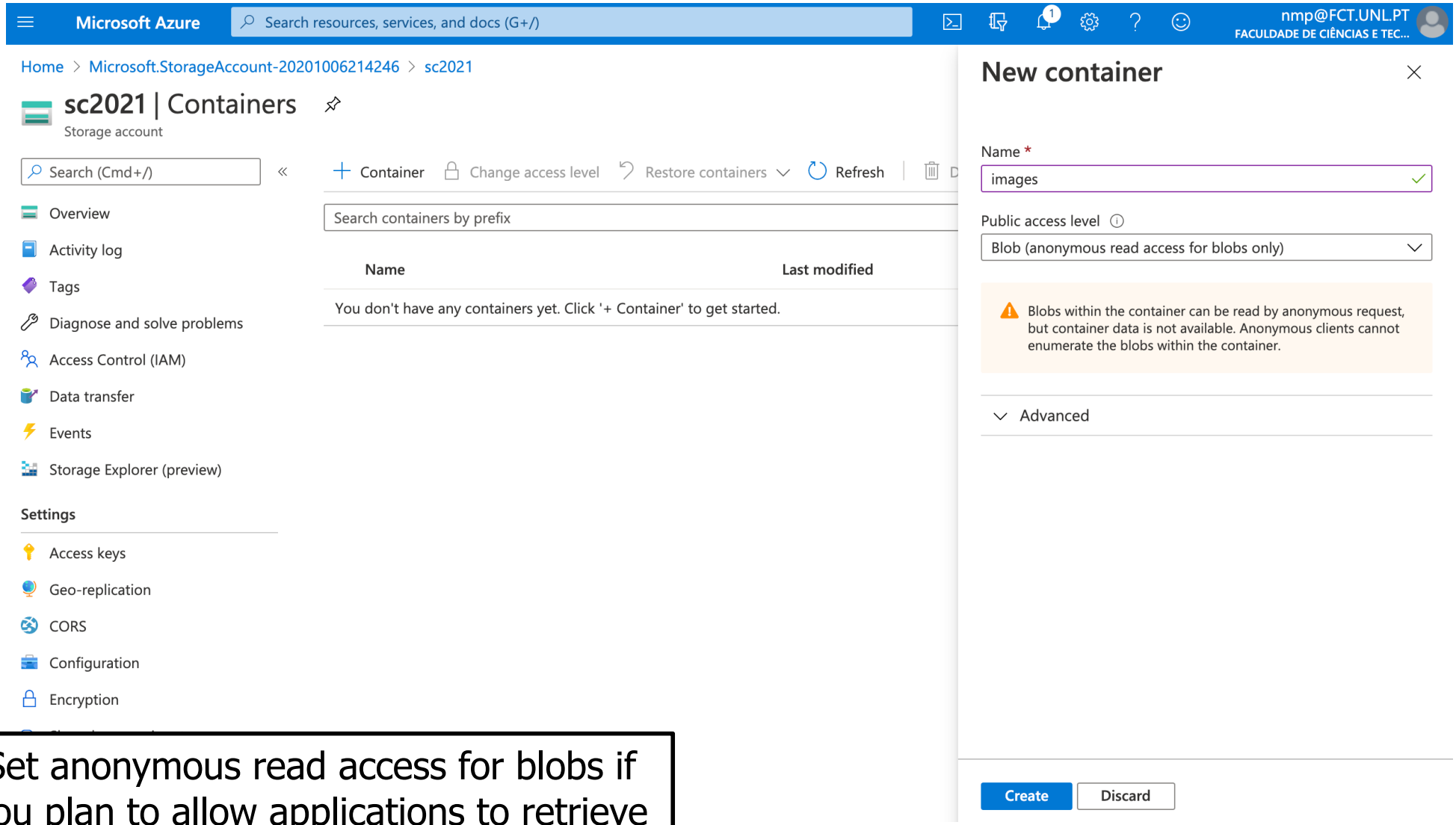
Create Blob Storage Container (1)



CREATE BLOB STORAGE CONTAINER (2)



CREATE BLOB STORAGE CONTAINER (3)

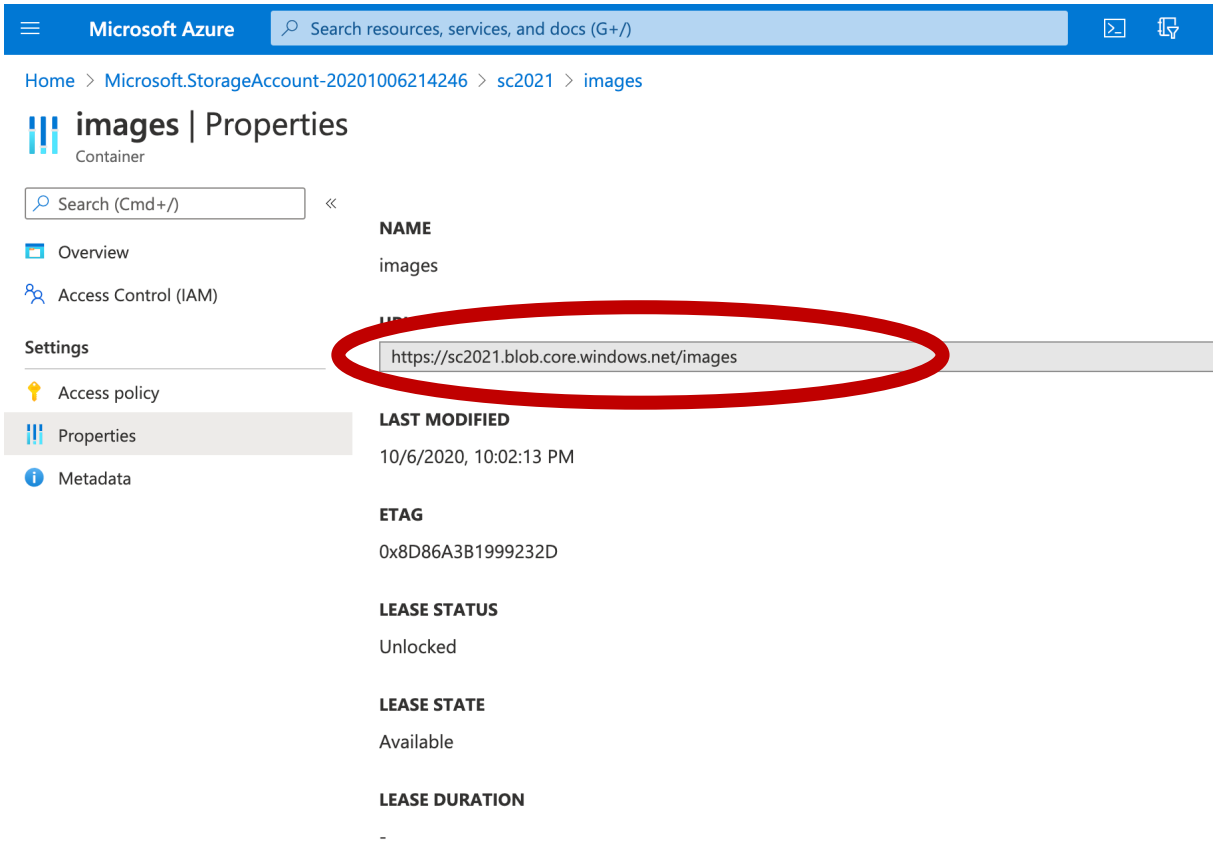


Set anonymous read access for blobs if you plan to allow applications to retrieve blobs directly from Blob Storage.

CONTAINER BASE URL

The base URL allows direct access to the blob by clients.

Try uploading a file (in Azure interface) and downloading it in the browser using this URL.



GOAL

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

- Create a StorageAccount + Blob Container @ Azure;
- **Complete the MediaResource, by storing data at Azure Blob Storage**
- Testing web apps

ACCESSING AZURE (BLOB) STORAGE

We will use the library provided by Microsoft.

Java Docs available at:

https://javadoc.io/doc/com.microsoft.azure/azure-storage-blob/latest/index.html

Example available at:

https://docs.microsoft.com/en-us/azure/storage/blobs/storage-quickstart-blobs-java?tabs=powershell

Maven dependencies

```
<dependency>
        <groupId>com.azure</groupId>
        <artifactId>azure-storage-blob</artifactId>
        <version>12.19.1</version>
</dependency>
```

Step 1: create client to blob storage

```
// Get connection string in the storage access keys page
String storageConnectionString = ...

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

STEP 2: UPLOAD BYTE ARRAY

```
// Get client to blob
BlobClient blob = containerClient.getBlobClient( filename);


// Upload contents from BinaryData (check documentation for
other alternatives)
// This will throw an exception if blob exists !!
blob.upload(data);
```

STEP 3: DOWNLOAD TO BYTE ARRAY

```
// Get client to blob
BlobClient blob = containerClient.getBlobClient(
filename);


// Download contents to BinaryData (check documentation
for other alternatives)
BinaryData data = blob.downloadContent();


byte[] arr = data.toBytes();
```

SOME NOTES

- Is it possible to create containers, etc. from the code?

Yes. Check documentation.

We will see that later.

CODE PROVIDED

The code provided (lab2.zip) is a Maven project with a single class that uploads a file to the Blob store (directly).

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-lab2-1.0-jar-with-dependencies.jar
scc.utils.UploadToStorage cats.jpeg
```

TO DO

Complete the media resource of lab 1, by storing data on a BLOB storage.

- Test your server with some simple tests.
  E.g. use the client provided to upload files; access the contents using your browser (or Postman).

Use the example scripts to benchmark your server.

- Code is on CLIP (lab2-test.zip)

GOAL

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

- Create a StorageAccount + Blob Container @ Azure;
- Complete the MediaResource, by storing data at Azure Blob Storage
- **Testing web apps**

ARTILLERY

https://artillery.io/docs/guides/overview/welcome.html

Specific docs for HTTP/REST testing.

https://artillery.io/docs/guides/guides/http-reference.html

TEST-IMAGES.YML (1)

```
config:
  target: 'https://scc-backend-4204.azurewebsites.net/rest'
  plugins:
    metrics-by-endpoint: {}
  processor: "./test-utils.js"
  variables:
     metricsProcessEndpoint : "myProcessEndpoint"
  phases:
  - name: "Warm up"
    duration: 10
    arrivalRate: 1
    rampTo: 5
  - name: "Main test"
    duration: 20
    arrivalRate: 5
```

Defines the base URL

Specifies the file with auxiliary functions

Specifies the phases of the tests:
Warm up, for 10 sec, with a new client per second, up to 5 clients.
Main Test, for 20 seconds, with 5 new clients per second.

TEST-IMAGES.YML (2)

Scenarios specify the sequence of operations to be executed.

```
scenarios:
  - name: 'Upload image'
    weight: 1
    flow:
      - post:
          url: "/media"
          headers:
            Content-Type: application/octet-stream
            Accept: application/json
          beforeRequest: "uploadImageBody"
          afterResponse: "processUploadReply"
```

Weight: relative weight of this scenario in the tests

url, header: to be used in the HTTP request
beforeRequest: function called before the request is performed.
afterResponse: function called after the response is received.

TEST-UTILS.JS (1)

Adds an image to the request body.

```
/** Sets the body to an image, when using images. */
function uploadImageBody(requestParams, context, ee, next) {
        requestParams.body = images.sample()
        return next()
}
```

Store the result id in a list of image ids.

```
/** Process reply of the download of an image.
 * Update the next image to read. */
function processUploadReply(requestParams, response, context, ee, next) {
        if( typeof response.body !== 'undefined' && response.body.length > 0) {
                imagesIds.push(response.body)
        }
    return next()
}
```

TEST-IMAGES.YML (3)

```
- name: 'Download image'
  weight: 5
  flow:
    - function: "selectImageToDownload"
    - get:
        url: "/media/{{ imageId }}"
        headers:
          Accept: application/octet-stream
        ifTrue: "imageId"
```

function: allows to call a function at any point in the test. In this case, before executing the HTTP request.

{{ imageId }}: this will be replaced by the value of variable imageId.

ifTrue: allows to make a call conditionally to the value of a variable.

TEST-UTILS.JS (2)

```
/** Select an image to download. */
function selectImageToDownload(context, events, done) {
        if( imagesIds.length > 0) {
                context.vars.imageId = imagesIds.sample()
        } else {
                delete context.vars.imageId
        }
        return done()
}
```

If we already have image ids in the list of image ids, select one; otherwise delete the variable.

INSTALLATION

- Install nodejs
- Install the following packages

npm install -g artillery

npm install -g faker

npm install -g node-fetch -save

npm install -g https://github.com/preguica/artillery-plugin-metrics-by-endpoint.git

RUNNING

- Run with:

```
artillery run test-images.yml
```

Do not forget to replace the base URL.

- Debug options
- http -> print HTTP request
  http:request / http:response -> prints request/response info;
  http* ->all information
- Mac/Linux

```
DEBUG=http artillery run test-images.yml
```

- Windows

```
SET DEBUG=http
artillery run test-images.yml
```

OUTPUT OF ARTILLERY

lab2-test — -bash — 119×35

```
All virtual users finished
Summary report @ 09:54:23(+0100) 2021-10-12
  Scenarios launched:  330
  Scenarios completed: 279
  Requests completed:  281
  Mean response/sec: 7.19
  Response time (msec):
    min: 57
    max: 2906
    median: 81
    p95: 1395
    p99: 2324.7
  Scenario counts:
    Download image: 281 (85.152%)
    Upload image: 49 (14.848%)
  Codes:
    200: 281
  Errors:
    ETIMEDOUT: 51
  POST:/rest/media:
    min: 183
    max: 9601
    median: 219
    p95: 9383.9
    p99: 9601
  GET:/rest/media/:
    min: 183
    max: 9969
    median: 230
    p95: 8121.8
    p99: 9660.6
  code 200 on POST:/rest/media: 41
  code 200 on GET:/rest/media/: 235
```

Information on launched scenarios.

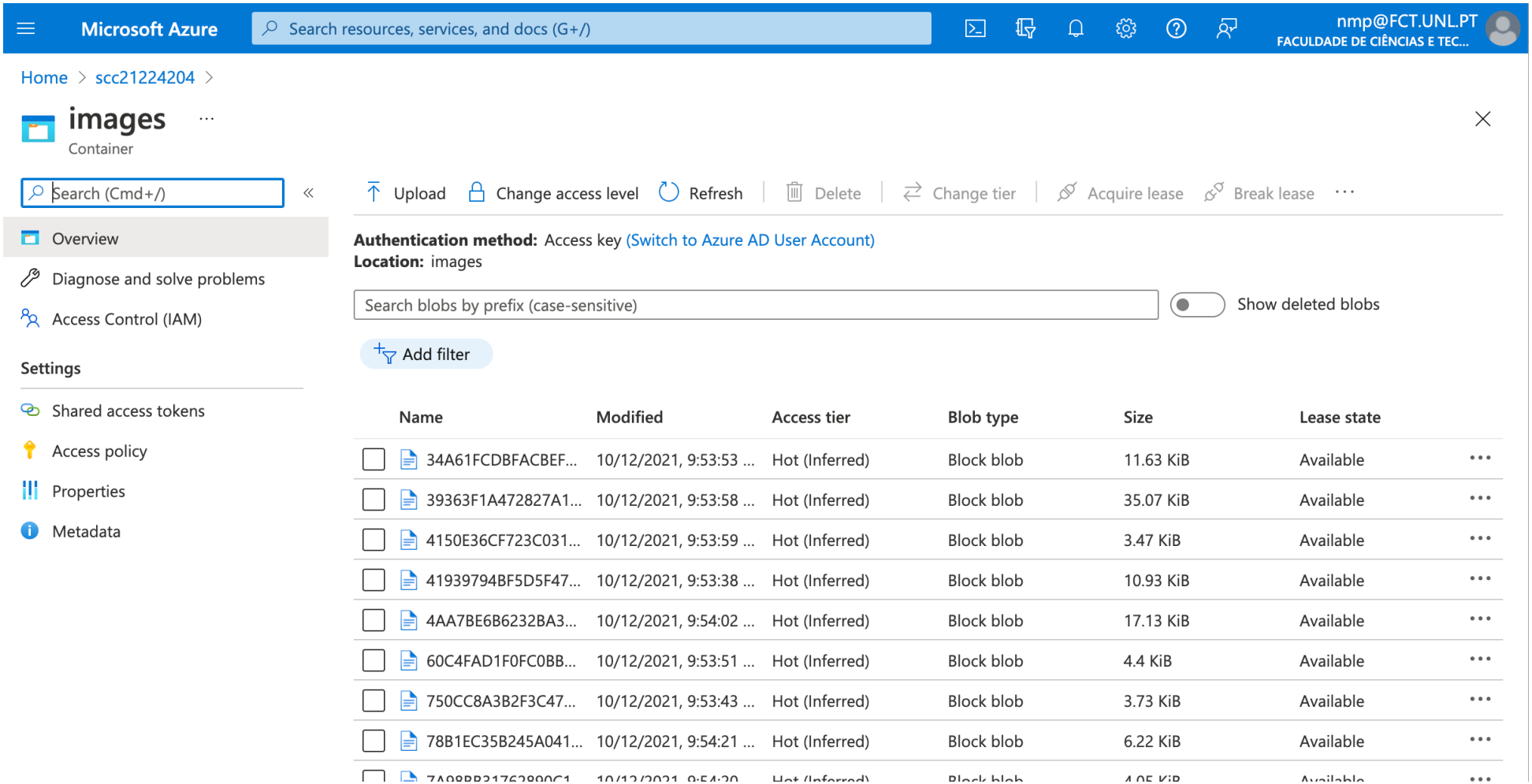
Information on individual REST requests.

GETs of all images are shown under /rest/media

Remove the plugin metrics-by-endpoint from YML to have information on individual requests.

CONTENTS OF BLOB CONTAINER

You can check that the BLOB container has the images POSTed using Azure portal.



TO DO

Complete the media resource.

- Test your server with some simple tests. E.g. using postman or a similar tool to submit REST requests.

Use the example scripts to benchmark your server.

- Code is on CLIP (lab2.zip)