# CLOUD COMPUTING SYSTEMS

Lab 5

João Resende, Nuno Preguiça

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

# GOAL

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

- Create Azure Functions in Java and deploy them in Azure Cloud Platform

# DOCUMENTATION

Azure documentation on writing function in Java:

https://docs.microsoft.com/en-us/azure/azure-functions/functions-reference-java

Triggers and bindings:

Timer

https://docs.microsoft.com/en-us/azure/azure-functions/functions-bindings-timer?tabs=java

HTTP

https://docs.microsoft.com/en-us/azure/azure-functions/functions-bindings-http-webhook
https://docs.microsoft.com/en-us/azure/azure-functions/functions-bindings-http-webhook-trigger?tabs=java

Blob storage

https://docs.microsoft.com/en-us/azure/azure-functions/functions-bindings-storage-blob
https://docs.microsoft.com/en-us/azure/azure-functions/functions-bindings-storage-blob-trigger?tabs=java

CosmosDB

https://docs.microsoft.com/en-us/azure/azure-functions/functions-bindings-cosmosdb-v2
https://docs.microsoft.com/en-us/azure/azure-functions/functions-bindings-cosmosdb-v2-trigger?tabs=java

# CREATE A PROJECT FOR THE FUNCTIONS

- **Alternative 1**

Use the example project available in CLIP.

- **Alternative 2**

Run the following command in an empty directory:

```
mvn archetype:generate \
 -DarchetypeGroupId=com.microsoft.azure \
 -DarchetypeArtifactId=azure-functions-archetype
```

# CONFIGURATION OF POM.XML

Some configurations need to be adjusted – functionAppName, functionAppRegion, functionPricingTier, functionResourceGroup.

```
<properties>
    ...
    <functionAppName>sccfunwesteurope4204</functionAppName>
    <functionAppRegion>westeurope</functionAppRegion>
    <functionPricingTier>B1</functionPricingTier>
    <functionResourceGroup>scc2223-rg-westeurope-4204</functionResourceGroup>
</properties>
```

# FUNCTIONS EXAMPLES

HttpTrigger

TimerTrigger

CosmosDBTrigger

BlobTrigger

# FUNCTIONS EXAMPLES

**HttpTrigger**

TimerTrigger

CosmosDBTrigger

BlobTrigger

# HTTPTRIGGER

https://docs.microsoft.com/en-us/azure/azure-functions/functions-bindings-http-webhook-trigger?tabs=java

Defines a function that is called as the result of executing an HTTP request.

Trigger properties:

- **methods** : defines the HTTP methods
  E.g.: `methods = { HttpMethod.GET }`
- **authLevel** : defines the level of authentication necessary to run the function
- **route** : defines the path – URL will be: {url}/api/{route}
  E.g.: `route = "serverless/redis/{key}"`
  URL will be: https://servername/api/serverless/redis/bla
- @BindingName : allow to get part of the route path
  E.g.: `@BindingName("key") String key`

# HTTPTRIGGER (EXAMPLE)

```
@FunctionName("get-redis")
public HttpResponseMessage getRedis( @HttpTrigger(name = "req",
                                     methods = {HttpMethod.GET},
                                     authLevel = AuthorizationLevel.ANONYMOUS,
                                     route = "serverless/redis/{key}")
                    HttpRequestMessage<Optional<String>> request,
                    @BindingName("key") String key,
                    final ExecutionContext context) {

  try (Jedis jedis = RedisCache.getCache().getJedisPool().getResource()) {

     String val = jedis.get(key);

     return request.createResponseBuilder(HttpStatus.OK)
                    .body( "GET key = " + key + "; val = " + val)
                    .build();

  }
}
```

ExecutionContext allows to access information about the invocation, including function name, invocation id, etc. https://docs.microsoft.com/en-us/java/api/com.microsoft.azure.functions.executioncontext?view=azure-java-stable

# FUNCTIONS EXAMPLES

HttpTrigger

**TimerTrigger**

CosmosDBTrigger

BlobTrigger

# TIMERTRIGGER

https://docs.microsoft.com/en-us/azure/azure-functions/functions-bindings-timer?tabs=java

Defines a function that executes periodically.

Schedule format:

second minutes hours day-of-month month year

E.g.: 30 */5 * * * * means:

Every 5 minutes (*/5)

at second 30

# TIMERTRIGGER (EXAMPLE)

```
@FunctionName("periodic-compute")
public void cosmosFunction( @TimerTrigger(name ="periodicSetTime",
                                          schedule = "*/20 * * * * *")
                String timerInfo,
                ExecutionContext context) {
    try (Jedis jedis = RedisCache.getCache().getJedisPool().getResource()) {
        jedis.set("serverlesstime", timerInfo);
    }
}
```

Check the written value using the HTTP redis get function on:
https://YOUR_SERVER.azurewebsites.net/api/serverless/redis/serverless-time

# FUNCTIONS EXAMPLES

HttpTrigger

TimerTrigger

**CosmosDBTrigger**

BlobTrigger

# CosmosDBTrigger

https://docs.microsoft.com/en-us/azure/azure-functions/functions-bindings-cosmosdb-v2-trigger?tabs=java

Allows to execute a function when there is an insert or update in a collection (but not for deletes).

Trigger properties:

- **databaseName**, **collectionName** – name of database and collection
- **connectionStringSetting** – application property where connection string is defined
  - NOTE: the script created by AzureManagement sets a property with name `AzureCosmosDBConnection`

# CosmosDBTrigger (example)

```
@FunctionName("cosmosDBtest")
public void updateMostRecentUsers(@CosmosDBTrigger(name = "cosmosTest",
                                databaseName = "scc2122dbnmp",
                                collectionName = "users",
                                createLeaseCollectionIfNotExists = true,
                                connectionStringSetting = "AzureCosmosDBConnection")
        String[] users,
        final ExecutionContext context ) {
  try (Jedis jedis = RedisCache.getCachePool().getResource()) {
    for( String u : users) {
      jedis.lpush("serverless::cosmos::users", u);
    }
    jedis.ltrim("serverless::cosmos::users", 0, 9);
  }
}
```

# CosmosDBTrigger (example)

```
@FunctionName("cosmosDBtest")
public void updateMostRecentUsers(@CosmosDBTrigger(name = "cosmosTest",
                                databaseName = "scc2122dbnmp",
                                collectionName = "users",
                                createLeaseCollectionIfNotExists = true,
                                connectionStringSetting = "AzureCosmosDBConnection")
      String[] users,
      final ExecutionContext context ) {
  try (Jedis jedis = RedisCache.getCachePool().getResource()) {
    for( String u : users) {
      jedis.lpush("serverless::cosmos::users", u);
    }
    jedis.ltrim("serverless::cosmos::users", 0, 9);
  }
}
```

Variable with the values inserted/updated.

Check the list of users on:
https://YOUR_SERVER.azurewebsites.net/api/serverless/redis/lrange/serverless::cosmos::users

# CosmosDBTrigger config

Run the following command to set the AzureCosmosDBConnection property

**[note: this command is in the .sh file created by AzureManagement]**

```
az functionapp config appsettings set \
    --name functions name \
    --resource-group functions group \
    --settings
"AzureCosmosDBConnection=AccountEndpoint=<url>;AccountKey=
<key>;"
```

# FUNCTIONS EXAMPLES

TimerTrigger

HttpTrigger

CosmosDBTrigger

**BlobTrigger**

# BLOBTRIGGER

https://docs.microsoft.com/en-us/azure/azure-functions/functions-bindings-storage-blob-trigger?tabs=java

Allows to execute a function when there is an insert or update in collection (but not for deletes).

Trigger properties:

- **path** – includes container name
- **connection** – application property where connection string is defined
  - NOTE: the script created by AzureManagement sets a property with name BlobStoreConnection

# BLOBTRIGGER (EXAMPLE)

```
@FunctionName("blobtest")
public void setLastBlobInfo(@BlobTrigger(name = "blobtest",
                                         dataType = "binary",
                                         path = "images/{name}",
                                         connection = "BlobStoreConnection")
                    byte[] content,
                    @BindingName("name") String blobname,
                    final ExecutionContext context) {
  try (Jedis jedis = RedisCache.getCachePool().getResource()) {
    jedis.set("serverless::blob::name", "Blob name : " +
                        blobname + " ; size = " +
                        (content == null ? "0" : content.length));
  }
}
```

# BLOBTRIGGER (EXAMPLE)

```
@FunctionName("blobtest")
public void setLastBlobInfo(@BlobTrigger(name = "blobtest",
                                dataType = "binary",
                                path = "images/{name}",
                                connection = "BlobStoreConnection")
        byte[] content,
        @BindingName("name") String blobname,
        final ExecutionContext context) {
  try (Jedis jedis = RedisCache.getCachePool().getResource()) {
    jedis.set("serverless::blob::name", "Blob name : " +
                    blobname + " ; size = " +
                    (content == null ? "0" : content.length));
```

Contents of the inserted/updated BLOB

Check the info of the last BLOB on:
https://YOUR_SERVER.azurewebsites.net/api/serverless/redis/serverless::blob::name

# BLOBTRIGGER CONFIG

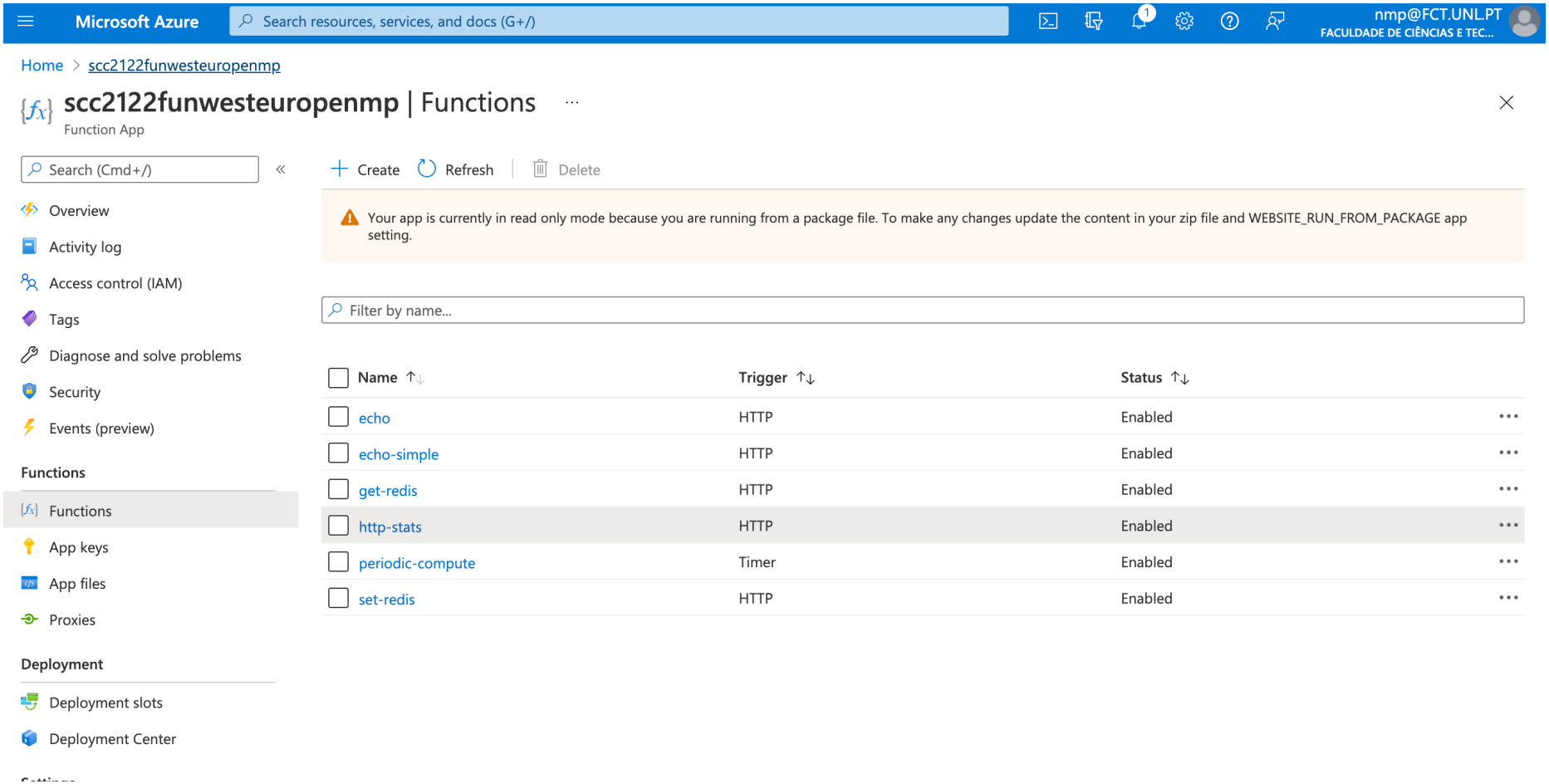
Run the following command to set the BlobStoreConnection property.

**[note: this command is in the .sh file created by AzureManagement]**

```
az functionapp config appsettings set \
    --name functions name \
    --resource-group functions group \
    --settings "BlobStoreConnection=<connection string>"
```

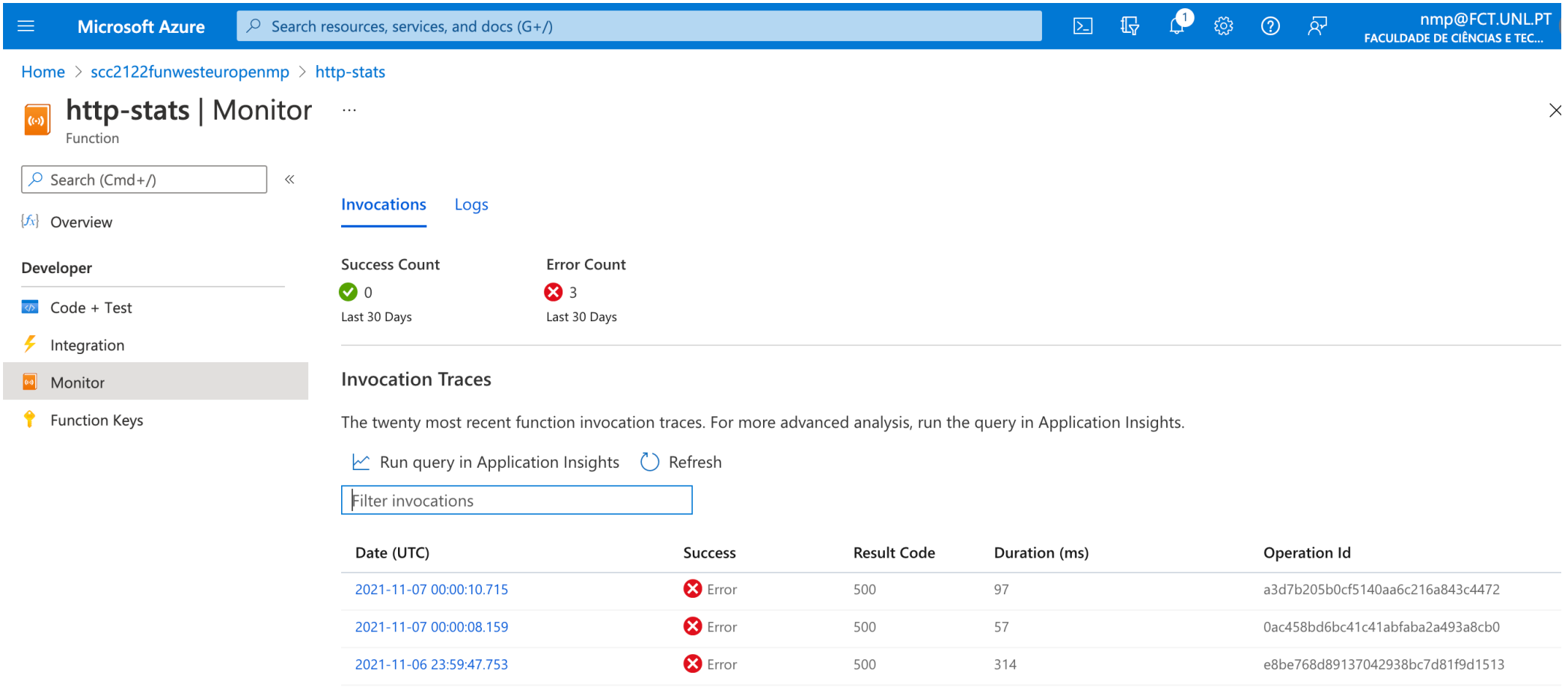
# DEBUG

In the Azure portal, it is possible to access the exceptions launched by an Azure function.



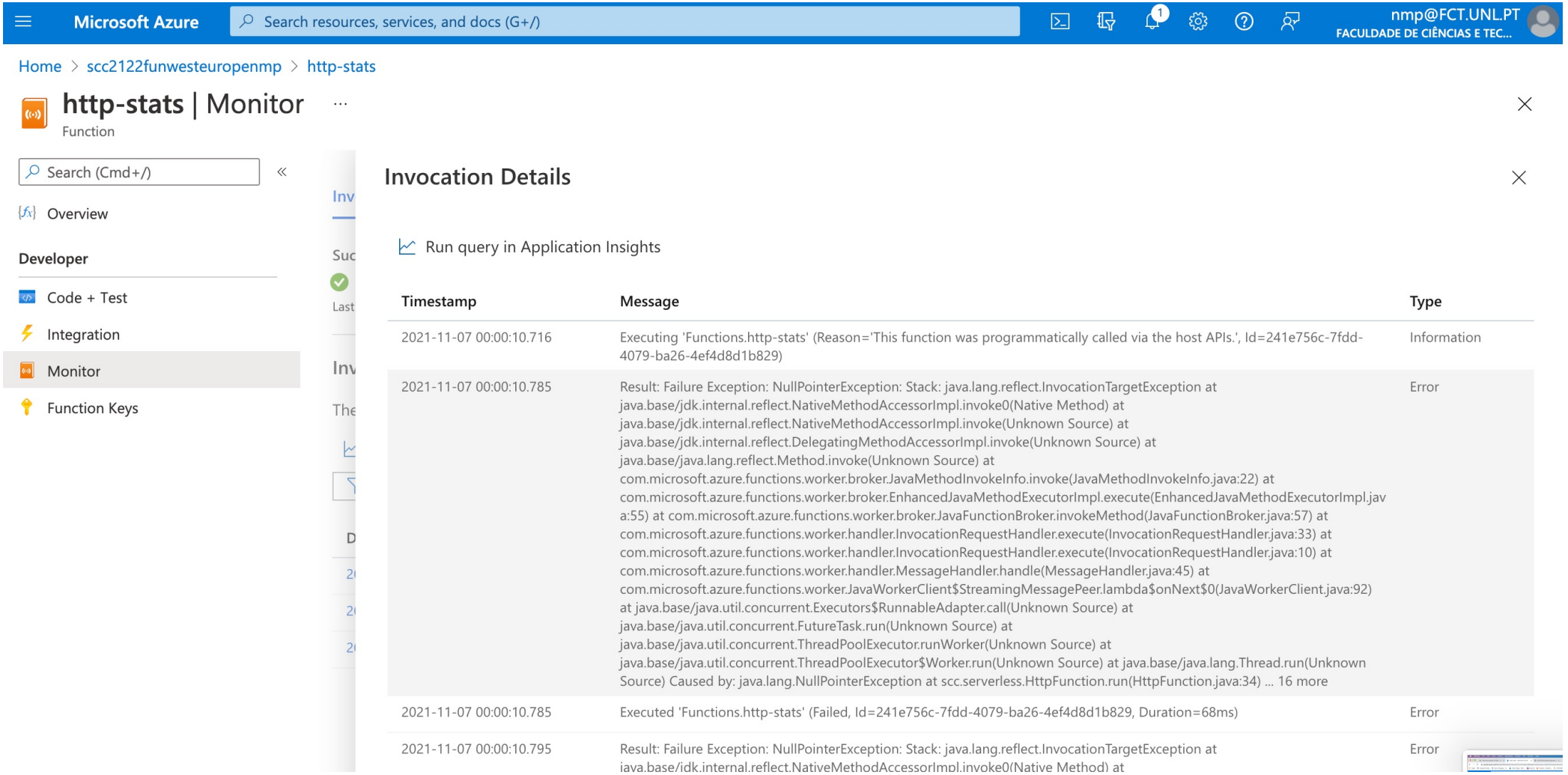
# DEBUG (2)

In the Azure portal, it is possible to access the exceptions launched by an Azure function.



# DEBUG (3)

In the Azure portal, it is possible to access the exceptions launched by an Azure function.



# CODE PROVIDED

The code provided (lab5.zip) is a Maven project with example Azure functions that can be deployed to Azure.

**You will need to change the properties in pom.xml – check slide 5**.

For compiling and deploying, just run:

```
mvn compile package azure-functions:deploy
```

After deploying, **do not forget to set the properties used by the Azure functions**.

**NOTE**: Blob and CosmosDB functions assume certain container/db names – update accordingly for your code.

Functions that access Redis assume that the appropriate keys are set.

# Deploy functions on Azure (cont'd)

```
(base) LazyMBP:lab5 nmp$ mvn azure-functions:deploy
[INFO] Scanning for projects...
[INFO]
[INFO] -------------------< pt.unl.fct.di.scc:scc2223-lab5 >-------------------
[INFO] Building Azure Java Functions 1.0
[INFO] --------------------------------[ jar ]---------
[INFO]
[INFO] --- azure-functions-maven-plugin:1.21.0:deplo
[INFO] Auth type: AZURE_CLI
Default subscription: Azure para Estudantes(83abecdf
Username: nmp@FCT.UNL.PT
[INFO] Subscription: Azure para Estudantes(83abecdf-
[INFO] Reflections took 125 ms to scan 5 urls, producing 25 keys and 751 values
[INFO] Set function worker runtime to java.
[INFO] Starting deployment...
[INFO] Trying to deploy artifact to scc2223funwesteuropenmp...
[INFO] Successfully deployed the artifact to https://scc2223funwesteuropenmp.azurewebsites.net
[INFO] Deployment done, you may access your resource through scc2223funwesteuropenmp.azurewebsites.net
[INFO] Syncing triggers and fetching function information
[INFO] Querying triggers...
[INFO] HTTP Trigger Urls:
[INFO]   echo : https://scc2223funwesteuropenmp.azurewebsites.net/api/serverless/echo/{text}
[INFO]   echo-simple : https://scc2223funwesteuropenmp.azurewebsites.net/api/serverless/echosimple/{text}
[INFO]   get-redis : https://scc2223funwesteuropenmp.azurewebsites.net/api/serverless/redis/{key}
[INFO]   http-info : https://scc2223funwesteuropenmp.azurewebsites.net/api/serverless/info
[INFO]   http-stats : https://scc2223funwesteuropenmp.azurewebsites.net/api/serverless/stats
[INFO]   lrange-redis : https://scc2223funwesteuropenmp.azurewebsites.net/api/serverless/redis/lrange/{key}
[INFO]   set-redis : https://scc2223funwesteuropenmp.azurewebsites.net/api/serverless/redis/{key}
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  01:05 min
[INFO] Finished at: 2022-10-17T16:29:57+01:00
[INFO] ------------------------------------------------------------------------
```

Sometimes deployment fails… when it succeeds you should have something like thisoutput, with the URLs to access functions triggered by HTTP.

# TODO

Use Azure function for implementing features in your project.

Suggestions:

- Use timer functions for closing auctions and clean-up tasks, periodically.
- Use CosmosDB function for maintain list of most recent auctions.