CLOUD COMPUTING SYSTEMS

Lab 3

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:

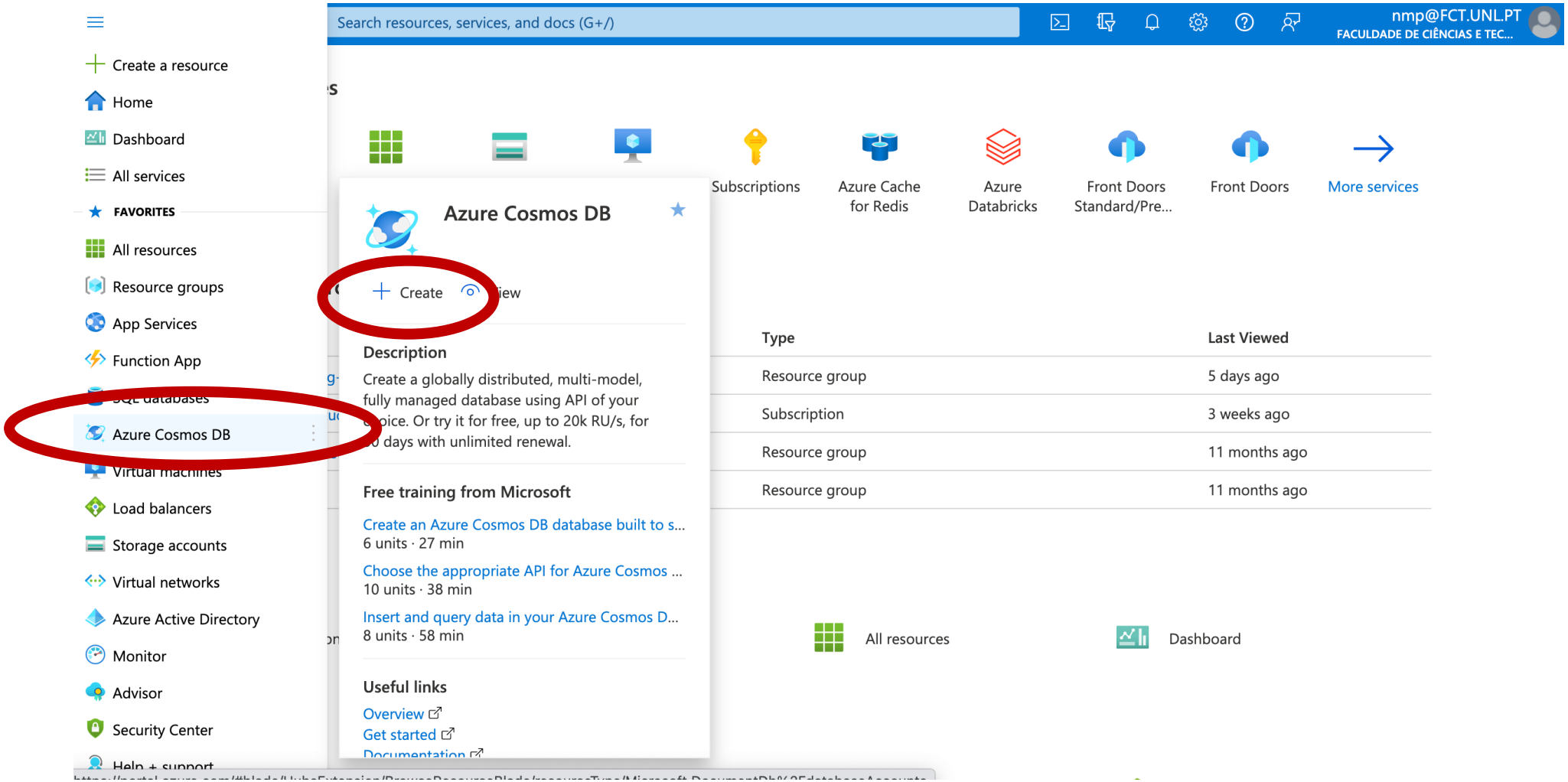
- Create a Cosmos DB account + Cosmos DB database + Cosmos DB Container @ Azure;
- Create the resources for the project by storing data at Azure Cosmos DB

GOAL

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

- **Create a Cosmos DB account + Cosmos DB database + Cosmos DB Container @ Azure;**
- Create the resources for the project by storing data at Azure Cosmos DB

Add Cosmos DB account (1)



Add Cosmos DB account (2)

Microsoft Azure | Search resources, services, and docs (G+/) | nmp@FCT.UNL.PT FACULDADE DE CIÊNCIAS E TEC...

Home > Azure Cosmos DB >

Select API option

Which API best suits your workload?

Azure Cosmos DB is a fully managed NoSQL database service for building scalable, high performance applications. Learn more

To start, select the API to create a new account. The API selection cannot be changed after account creation.

Core (SQL) - Recommended

Azure Cosmos DB's core, or native API for working with documents. Supports fast, flexible development with familiar SQL query language and client libraries for .NET, JavaScript, Python, and Java.

Create | Learn more

Azure Cosmos DB API for MongoDB

Fully managed database service for apps written for MongoDB. Recommended if you have existing MongoDB workloads that you plan to migrate to Azure Cosmos DB.

Create | Learn more

Cassandra

Fully managed Cassandra database service for apps written for Apache Cassandra. Recommended if you have existing Cassandra workloads that you plan to migrate to Azure Cosmos DB.

Create | Learn more

Azure Table

Fully managed database service for apps written for Azure Table storage. Recommended if you have existing Azure Table storage workloads that you plan to migrate to Azure Cosmos DB, but do not want to re-write your application to use the SQL API.

Create | Learn more

Gremlin (Graph)

Fully managed graph database service using the Gremlin query language, based on Apache TinkerPop project. Recommended for new workloads that need to store relationships between data.

Create | Learn more

Check lecture 3 for info on these options.

Add Cosmos DB account (3)

Home > Create a resource > Select API option >

Create Azure Cosmos DB Account - Core (SQL) ...

Basics | Global Distribution | Networking | Backup Policy | Encryption | Tags | Review + create

Azure Cosmos DB is a fully managed NoSQL database service for building scalable, high performance applications. Try it for free, for 30 days with unlin
$24/month per database, multiple containers included. Learn more

Project Details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * : Azure para Estudantes

Resource Group * : scc-backend-rg-4204

Create new

Instance Details

Account Name * : scc22234204

Location * : (Europe) West Europe

Capacity mode : (•) Provisioned throughput ( ) Serverless

Learn more about capacity mode

With Azure Cosmos DB free tier, you will get the first 1000 RU/s and 25 GB of storage for free in an account. You can enable free tier on up to one acc
per account.

Apply Free Tier Discount : ( ) Apply (•) Do Not Apply

Limit total account throughput : [x] Limit the total amount of throughput that can be provisioned on this account

Review + create | Previous | Next: Global Distribution

Check lecture 3 for info on these options.

ADD COSMOS DB ACCOUNT (4)

Microsoft Azure | Search resources, services, and docs (G+/) | nmp@FCT.UNL.PT FACULDADE DE CIÊNCIAS E TEC...

Home > Azure Cosmos DB > Select API option >

Create Azure Cosmos DB Account - Core (SQL)

Validation Success

Basics | **Global Distribution** | Networking | Backup Policy | Encryption | Tags | Review + create

Global Distribution

Configure global distribution and regional settings for your account. You can also change these settings after the account is created.

Geo-Redundancy	○ Enable	◉ Disable
Multi-region Writes	○ Enable	◉ Disable
Availability Zones	○ Enable	◉ Disable

Review + create | Previous | Next: Networking

Check lecture 3 for info on these options.

ADD COSMOS DB ACCOUNT (5)

Microsoft Azure | Search resources, services, and docs (G+/) | nmp@FCT.UNL.PT FACULDADE DE CIÊNCIAS E TEC...

Home > Azure Cosmos DB > Select API option >

Create Azure Cosmos DB Account - Core (SQL)

Basics | Global Distribution | Networking | **Backup Policy** | Encryption | Tags | Review + create

Azure Cosmos DB provides two different backup policies. You will not be able to switch between backup policies after the account has been created. Learn more about the differences of the two backup policies and pricing details. Learn more

Backup policy: (•) Periodic ( ) Continuous

Backup interval: 240 Minute(s) — 60-1440

Backup retention: 8 Hours(s) — 8-720

Copies of data retained: 2

Backup storage redundancy *:
- ( ) Geo-redundant backup storage
- ( ) Zone-redundant backup storage
- (•) Locally-redundant backup storage

Review + create | Previous | Next: Encryption

Check lecture 3 for info on these options.

Add Cosmos DB account (5)

The cost of all database operations is normalized and expressed as request units (RU). Azure Cosmos DB offers two database operations models:

- **Provisioned Throughput** is measured in request units per second (RU/s) and billed per hour. It offers single-digit millisecond reads and writes and 99.999-percent availability worldwide, backed by SLAs. It is ideal for large, critical workloads requiring guaranteed low-latency and high-availability. You can choose between two capacity management options: Standard Provisioned Throughput and Autoscale Provisioned Throughput.
- **Serverless** bills for the request units (RU) used for each database operation. Serverless makes it easy to run spiky workloads that don't have sustained traffic. It can handle traffic bursts on demand, without resource planning or management.

Storage is billed as GBs consumed by SSD-backed data and index across all the Azure regions your database is distributed to. Two backup copies are provided free, with additional copies billed as total GBs of data stored.

ADD COSMOS DB ACCOUNT (6)

Microsoft Azure | Search resources, services, and docs (G+/) | nmp@FCT.UNL.PT FACULDADE DE CIÊNCIAS E TEC...

Home >

Microsoft.Azure.CosmosDB-20201013122538 | Overview

Deployment

Search (Cmd+/)

- Overview
- Inputs
- Outputs
- Template

Delete | Cancel | Redeploy | Refresh

We'd love your feedback! →

Your deployment is complete

Deployment name: Microsoft.Azure.CosmosDB-20201013122538
Subscription: Azure para Estudantes
Resource group: scc2021-4204

Start time: 10/13/2020, 12:31:05 PM
Correlation ID: 6b601b58-258d-485c-9d76

Deployment details (Download)

Next steps

Go to resource

This takes time…

ADD DATABASE + CONTAINER (1)

Microsoft Azure | Search resources, services, and docs (G+/)

Home > Microsoft.Azure.CosmosDB-20221003221732 | Overview >

scc22234204
Azure Cosmos DB account

Search

- Overview
- Activity log
- Access control (IAM)
- Tags
- Diagnose and solve problems
- Cost Management
- Quick start
- Notifications
- Data Explorer

Settings

- Features
- Replicate data globally
- Default consistency
- Backup & Restore
- Networking
- CORS
- Dedicated Gateway
- Keys

Add Container | Refresh | Move | Data Explorer | Enable geo-redundancy | Delete Account

Add Container

Essentials

Status
Online

Resource group (move)
scc-backend-rg-4204

Subscription (move)
Azure para Estudantes

Subscription ID
83abecdf-8b40-49a0-bcae-b5fba4011353

Total throughput limit
4000 RU/s

Read Locations
West Europe

Write Locations
West Europe

URI
https://scc22234204.documents.azure.com:443/

Free Tier Discount
Opted Out

Capacity mode
Provisioned throughput

See more

Containers

Looks like you don't have any containers yet. Data Explorer

Monitoring

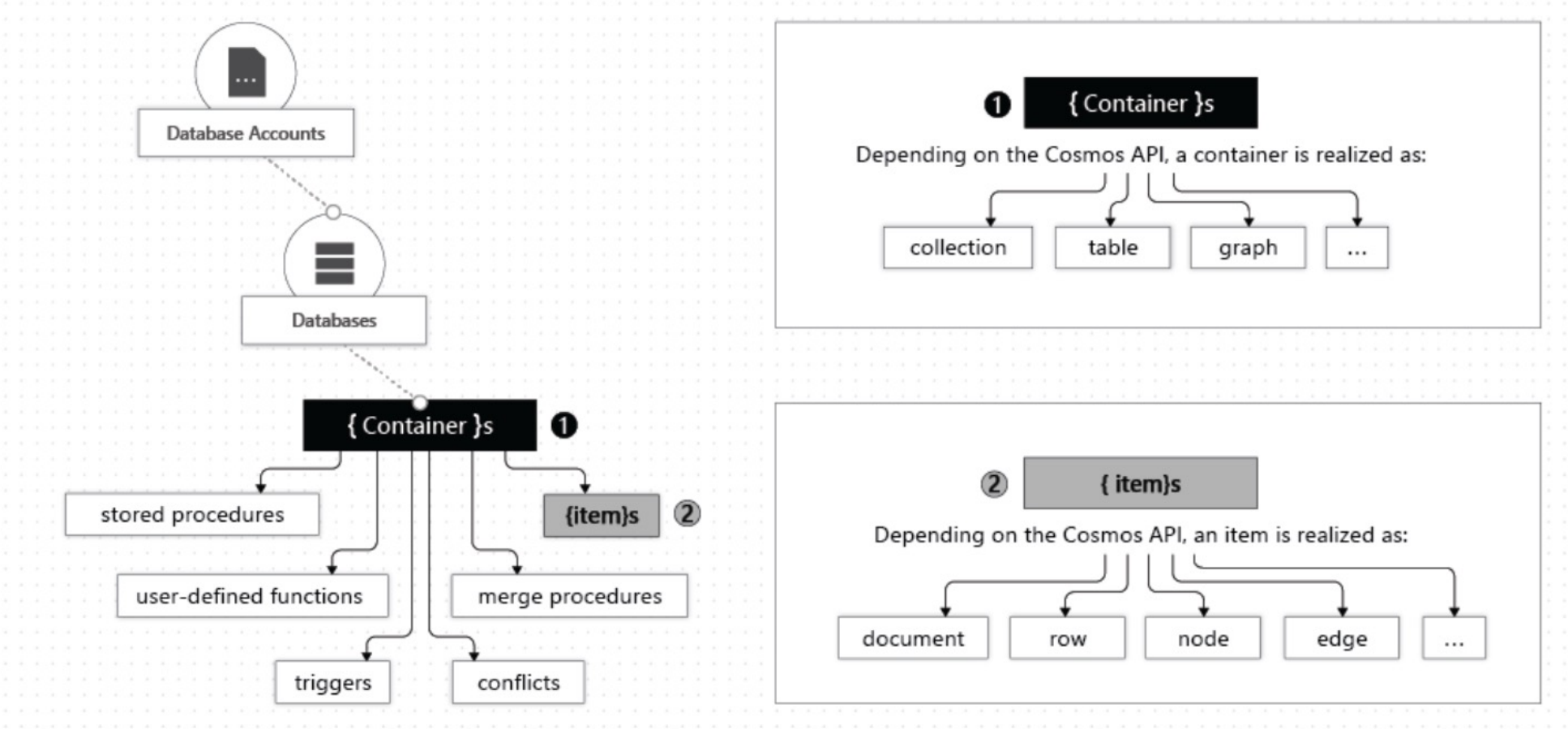
Show data for last 1 hour | 24 hours | 7 days | 30 days

Requests

Total
Http 2xx
Http 400
Http 401

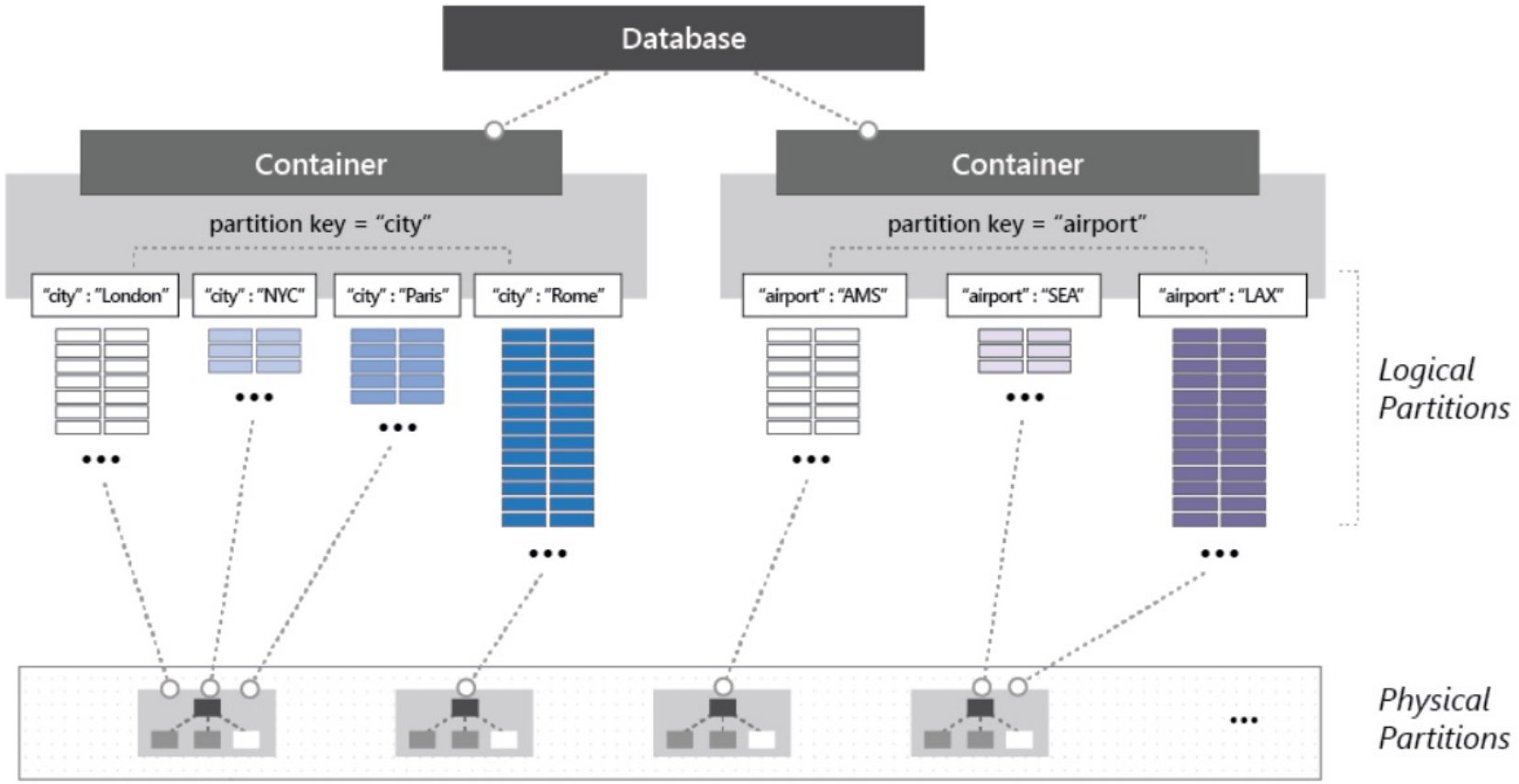
ADD DATABASE + CONTAINER (2)

A database may include a set of containers.
A container can be a collection, table, graph, etc.



ADD DATABASE + CONTAINER (3)

A container is horizontally partitioned across multiple machines according to the partition key.



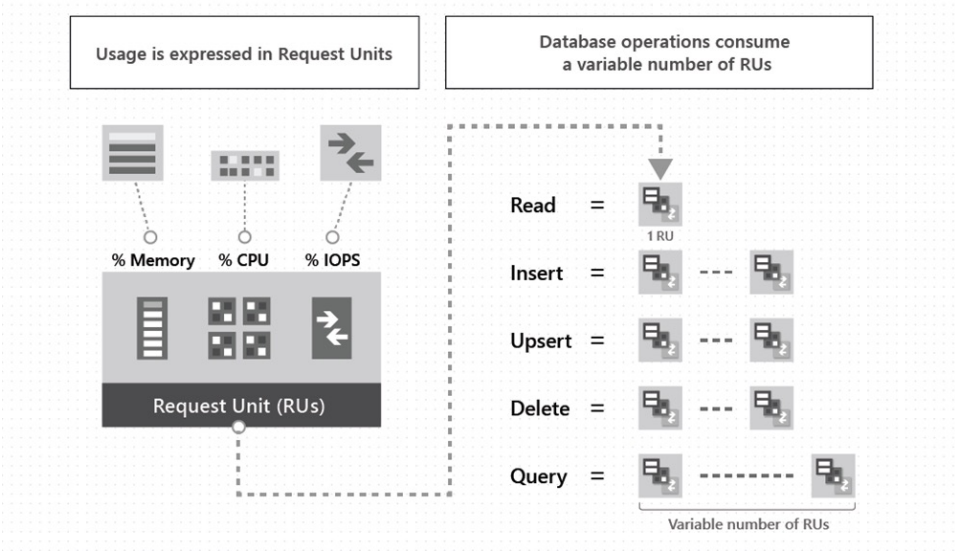
ADD DATABASE + CONTAINER (4)

In Cosmos DB, the throughput is provisioned (i.e., Cosmos DB reserve the necessary resources to achieve the provisioned throughput) at the database or container level.

Better use at the database level: first 400RU/s are free.

Minimum: 400 RU/s.

Read 1Kb document cost 1 RU.



Info on units:
https://docs.microsoft.com/en-us/azure/cosmos-db/request-units

ADD DATABASE + CONTAINER (5)

Unlike what is common in databases, by default, CosmosDB indexes every property of every item and enforces range indexes for any string or number all containers.

It is possible to override the default policy by:

1. Overriding the indexing mode: support for consistent, lazy (where indices are updated in background), none;
2. Specifying that some properties do not need to be indexed, using the exclude path.

ADD DATABASE + CONTAINER (6)

The partition key allows to control which objects will be co-located – e.g. if you want to co-locate all messages for a given channel, set the channel id as the partition key for messages.

New Container

* Database id

Create new Use existing

scc2223db

Share throughput across containers

* Database throughput (400 - unlimited RU/s)

Autoscale Manual

Estimate your required RU/s with capacity calculator.

400 *

Estimated cost (USD): **$0.032 hourly / $0.77 daily / $23.36 monthly** (1 region, 400RU/s, $0.00008/RU)

* Container id

users

* Partition key

For small workloads, the item ID is a suitable choice for the partition key.

/id

* Co

Us

* Partition key

For small workloads, the item ID is a suitable choice for the partition key.

/id

Unique keys

Add unique key

Analytical store

On Off

Azure Synapse Link is required for creating an analytical store container. Enable Synapse Link for this Cosmos DB account. Learn more

Enable

Advanced

OK

Cosmos DB account: access keys (for code)

Microsoft Azure | Search resources, services, and docs (G+/)

Home > Microsoft.Azure.CosmosDB-20211016233538 > scc21224204

scc21224204 | Keys

Azure Cosmos DB account

Search (Cmd+/)

- Tags
- Diagnose and solve problems
- Quick start
- Notifications
- Data Explorer

Settings

- Features
- Replicate data globally
- Default consistency
- Backup & Restore
- Firewall and virtual networks
- Private Endpoint Connections
- CORS
- Dedicated Gateway
- Keys
- Advisor Recommendations
- Add Azure Cognitive Search

Read-write Keys | Read-only Keys

URI

https://scc21224204.documents.azure.com:443/

PRIMARY KEY

5B782TMdtkz70AAUOx9LAtwyz0mBDdJnk9OY4tdZx273bkRe37Y7B9Ib8gmN06iBPj7ia5h3uC9a

SECONDARY KEY

BQq3WHfUhlPFdOenq4wrvykmEtsqUG5UJckjMxRrKZ8EmoYnsAp3V9g5Gt9dKSZTSOj5Hf4jnnv

PRIMARY CONNECTION STRING

AccountEndpoint=https://scc21224204.documents.azure.com:443/;AccountKey=5B782TMdtkz

SECONDARY CONNECTION STRING

AccountEndpoint=https://scc21224204.documents.azure.com:443/;AccountKey=BQq3WHfUhl

GOAL

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

- Create a Cosmos DB account + Cosmos DB database + Cosmos DB Container @ Azure;
- **Create the resources for the project by storing data at Azure Cosmos DB**

ACCESSING COSMOS DB: USEFUL LINKS

We will use the library provided by Microsoft.

Java Docs available at:

https://azuresdkdocs.blob.core.windows.net/$web/java/azure-cosmos/latest/index.html

Overview on how to use at:

https://docs.microsoft.com/en-us/azure/cosmos-db/create-sql-api-java

Cosmos DB SQL cheat sheet

https://go.microsoft.com/fwlink/?LinkId=623215

MAVEN DEPENDENCIES

```
<dependency>
        <groupId>com.azure</groupId>
        <artifactId>azure-cosmos</artifactId>
        <version>4.37.0</version>
</dependency>
<dependency>
        <groupId>org.apache.commons</groupId>
        <artifactId>commons-lang3</artifactId>
        <version>3.12.0</version>
</dependency>
<dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-simple</artifactId>
        <version>2.0.3</version>
</dependency>
<dependency>
        <groupId>org.apache.logging.log4j</groupId>
        <artifactId>log4j-core</artifactId>
        <version>2.19.0</version>
</dependency>
```

Maven dependencies (cont.)

```
<dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-core</artifactId>
        <version>2.13.4</version>
</dependency>
<dependency>
        <groupId>org.jboss.resteasy</groupId>
        <artifactId>resteasy-jackson2-provider</artifactId>
        <version>6.2.0.Final</version>
</dependency>
```

STEP 1: CREATE CLIENT TO COSMOS DB (1)

```
private static final String CONNECTION_URL =
"https://sc22234204.documents.azure.com:443/";

private static final String DB_KEY = ...

private static final String DB_NAME = "scc4204db";
```

STEP 1: CREATE CLIENT TO COSMOS DB (2)

```
CosmosClient client = new CosmosClientBuilder()
        .endpoint(CONNECTION_URL)
        .key(DB_KEY)
        .directMode()// comment this if not to use direct mode
        .consistencyLevel(ConsistencyLevel.SESSION)
        .connectionSharingAcrossClientsEnabled(true)
        .contentResponseOnWriteEnabled(true)
        .buildClient();
```

On write, return the object written.

STEP 2: GET REFERENCE TO CONTAINER

```
db = client.getDatabase(DB_NAME);
users = db.getContainer("Users");
```

STEP 3: USERDAO CLASS

CosmosDB adds a set of fields to documents – add them to your class to access them easily.
_rid: unique id of item
_ts: timestamp of the last update to the item

```
public class UserDAO {
        private String _rid;
        private String _ts;
        private String id;
        private String name;
        private String photoId;
        private String[] channelIds;

        public UserDAO() {
        }
        ...
        public String get_rid() {
                return _rid;
        }
        public void set_rid(String _rid) {
                this._rid = _rid;
        }
        // Other setters/getters
        ...
        public User toUser() {
                return new User( ... );
        }
}
```

STEP 3: USER CLASS

Companion User class for transferring to clients – it does not include CosmosDB fields.

```
public class User {
        private String id;
        private String name;
        private String photoId;
        private String[] channelIds;

        public User() {
        }
        ...

}
```

STEP 4: WRITE ITEM

```
CosmosItemResponse<UserDAO> res = users.createItem(u);
if( res.getStatusCode() < 300)
        return res.getItem();
else
        throw new NotFoundException();
```

STEP 5: READ ITEMS

```
CosmosPagedIterable<UserDAO> res = users.queryItems(
        "SELECT * FROM Users WHERE Users.id=\"" + id +
        "\"", new CosmosQueryRequestOptions(), UserDAO.class);
for(UserDAO u : res) {
        System.out.println(u);
}
```

CODE PROVIDED

The code provided (lab3.zip) is a Maven project with a single class that creates a User and access user directly on CosmosDB.

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/scc2122-lab3-1.0-jar-with-dependencies.jar scc.utils.TestUsers
```

SOME NOTES

- Is it possible to access a Cosmos DB from a program running in my machine?

Yes. The example in CLIP does that for users.

NOTE: this is particularly useful for getting the code correct.

- Is it possible to use Cosmos DB with other data models?

Yes. Check documentation.

CosmosDB SQL

Getting started:

https://docs.microsoft.com/en-us/azure/cosmos-db/sql/sql-query-getting-started

Check the cheat sheet:

https://go.microsoft.com/fwlink/?LinkId=623215

Pagination

Some endpoints will require pagination – e.g. messages in a channel. This can be performed using OFFSET and LIMIT keywords

SELECT * FROM Users ORDER BY Users.id OFFSET 20 LIMIT 10

- LIMIT defines the number of results to return;
- OFFSET defines the number of results to skip.

TODO

Extend your backend to support the necessary endpoints for storing users.

NOTE: do not forget to delete the Cosmos DB resources

TRAB-TEST-V1.ZIP

First version of scripts for testing project 1.

Single script to create users.

CREATE-USERS.YML

```yaml
scenarios:
  - name: 'Create users'
    weight: 1
    flow:
      - loop:                           # let's create 50 users - loop ... count
        - post:                         # First: post image for the user
            url: "/media"
            headers:
              Content-Type: application/octet-stream
              Accept: application/json
            beforeRequest: "uploadImageBody"
            capture:
              regexp: "(.+)"
              as: "imageId"             # capture the reply as image id to be used in user
creation
        - function: "genNewUser"        # Generate the needed information for the user
        - post:
            url: "/user"
            headers:
              Content-Type: application/json
              Accept: application/json
            json:
              id: "{{ id }}"
              name: "{{ name }}"
              pwd: "{{ pwd }}"
              photoId: "{{ imageId }}"
              channelIds: []
            afterResponse: "genNewUserReply"    # capture result and store in file
        count: 50
```

Capture allows to get a result and store it in a variable – imageId in this case.

JSON document represent using **json:**
Variable used generated in method genNewUser

TEST-UTILS.JS

```
/**
 * Generate data for a new user using Faker
 */
function genNewUser(context, events, done) {
        const first = `${Faker.name.firstName()}`
        const last = `${Faker.name.lastName()}`
        context.vars.id = first + "." + last
        context.vars.name = first + " " + last
        context.vars.pwd = `${Faker.internet.password()}`
        return done()
}
```

Faker library generates realistic data for testing.

```
/**
 * Process reply for of new users to store the id on t
 */
function genNewUserReply(requestParams, response, context, ee, next) {
        if( response.statusCode >= 200 && response.statusCode < 300 &&
response.body.length > 0)  {
                let u = JSON.parse( response.body)
                users.push(u)
                fs.writeFileSync('users.data', JSON.stringify(users));
        }
        return next()
}
```

We are storing the users created for later use.