**Azure Cosmos DB Emulator with Spring Boot**

This document explains how to set up and integrate the Azure Cosmos DB Emulator with a Spring Boot application, starting from pulling the emulator Docker image to testing your integration with APIs.

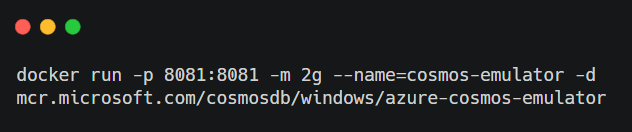
**Step 1: Install the Azure Cosmos DB Emulator**

**Using Docker**

1. Pull the Azure Cosmos DB Emulator Docker image:

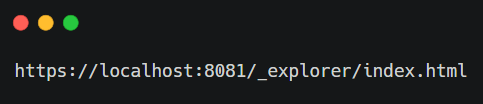


1. Start the emulator container:



This command maps port 8081 on the host to port 8081 in the container. It also allocates at least 2GB of memory, as required by the emulator.

1. Confirm that the emulator is running by accessing the below URL in your browser. If the connection is successful, the emulator is set up.



**Step 2: Download the Emulator Certificate**

1. Open the emulator web interface at https://localhost:8081/\_explorer/index.html.
2. Download the certificate by clicking on the "Download Certificate" link or by visiting https://localhost:8081/emulator.crt in your browser.
3. Save the file to your local machine (e.g., C:\Users\your-username\Downloads\localhost.crt).

**Step 3: Import the Certificate**

1. Import the certificate into the Java KeyStore:
2. keytool -import -trustcacerts -file "C:\Users\your-username\Downloads\localhost.crt" -alias cosmos-emulator -keystore "C:\Program Files\Java\jdk-17\lib\security\cacerts"
   * When prompted, enter the password for the keystore. The default password is typically changeit.
3. Verify the certificate was successfully imported:
4. keytool -list -keystore "C:\Program Files\Java\jdk-17\lib\security\cacerts" -alias cosmos-emulator

Ensure that the Cosmos Emulator certificate appears in the list.

**Step 4: Create a Spring Boot Application**

1. **Generate a Spring Boot application** using [Spring Initializr](https://start.spring.io/):
   * Add the following dependencies:
     + Spring Web
     + Spring Data Cosmos
2. Add the azure-spring-data-cosmos dependency to your pom.xml:
3. <dependency>
4. <groupId>com.azure</groupId>
5. <artifactId>azure-spring-data-cosmos</artifactId>
6. <version>4.0.0</version>
7. </dependency>
8. Add the application.properties to configure the connection to the emulator:
9. azure.cosmos.uri=https://localhost:8081
10. azure.cosmos.key=C2y6yDjf5/R+ob0N8A7Cgv30VRDJIWEHLM+4QDU5DE2nQ9nDuVTqobD4b8mGGyPMbIZnqyMsEcaGQy67XIw/Jw==
11. azure.cosmos.database=my-database
12. azure.cosmos.consistency-level=SESSION
13. azure.cosmos.populate-query-metrics=true
    * The uri and key provided are the default values for the Cosmos Emulator.
    * Replace my-database with your desired database name.

**Step 5: Create a Sample Model and Repository**

**Model:**

Create an entity representing your data:

package com.example.cosmos.model;

import com.azure.spring.data.cosmos.core.mapping.Container;

import org.springframework.data.annotation.Id;

@Container(containerName = "products")

public class Product {

@Id

private String id;

private String name;

private String description;

private double price;

// Getters and setters omitted for brevity

}

**Repository:**

Create a repository for the model:

package com.example.cosmos.repository;

import com.example.cosmos.model.Product;

import com.azure.spring.data.cosmos.repository.CosmosRepository;

import org.springframework.stereotype.Repository;

@Repository

public interface ProductRepository extends CosmosRepository<Product, String> {

}

**Step 6: Test the Integration**

**Controller:**

Create a simple REST controller to test CRUD operations:

package com.example.cosmos.controller;

import com.example.cosmos.model.Product;

import com.example.cosmos.repository.ProductRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController

@RequestMapping("/api/products")

public class ProductController {

@Autowired

private ProductRepository productRepository;

@GetMapping

public List<Product> getAllProducts() {

return productRepository.findAll();

}

@PostMapping

public Product createProduct(@RequestBody Product product) {

return productRepository.save(product);

}

@GetMapping("/{id}")

public Product getProductById(@PathVariable String id) {

return productRepository.findById(id).orElse(null);

}

@PutMapping("/{id}")

public Product updateProduct(@PathVariable String id, @RequestBody Product product) {

product.setId(id);

return productRepository.save(product);

}

@DeleteMapping("/{id}")

public void deleteProduct(@PathVariable String id) {

productRepository.deleteById(id);

}

}

**Postman:**

Use Postman or curl commands to test the CRUD endpoints:

* GET /api/products: Retrieve all products.
* POST /api/products: Add a product.
* GET /api/products/{id}: Get a product by ID.
* PUT /api/products/{id}: Update a product by ID.
* DELETE /api/products/{id}: Delete a product by ID.

**Troubleshooting Tips**

1. **Certificate Issues:**
   * Ensure the certificate is correctly imported into the Java KeyStore.
   * If using WSL or a non-Windows OS, map the certificate path appropriately.
2. **Docker Networking Issues:**
   * Use docker inspect cosmos-emulator to confirm port mappings.
3. **Compatibility Issues:**
   * Ensure the versions of azure-spring-data-cosmos and the Java SDK are compatible.

This completes the setup and integration of Azure Cosmos DB Emulator with Spring Boot. Once everything is set up, you should be able to seamlessly perform CRUD operations against the emulator in a local environment.