



CLOUD DAY 2020

29 OTTOBRE • #CLOUDDAY2020

**SERVERLESS SCALABLE
BACK-END API WITH
HYBRID DATA MODELS**

**Davide Mauri
@mauridb**



Davide Mauri

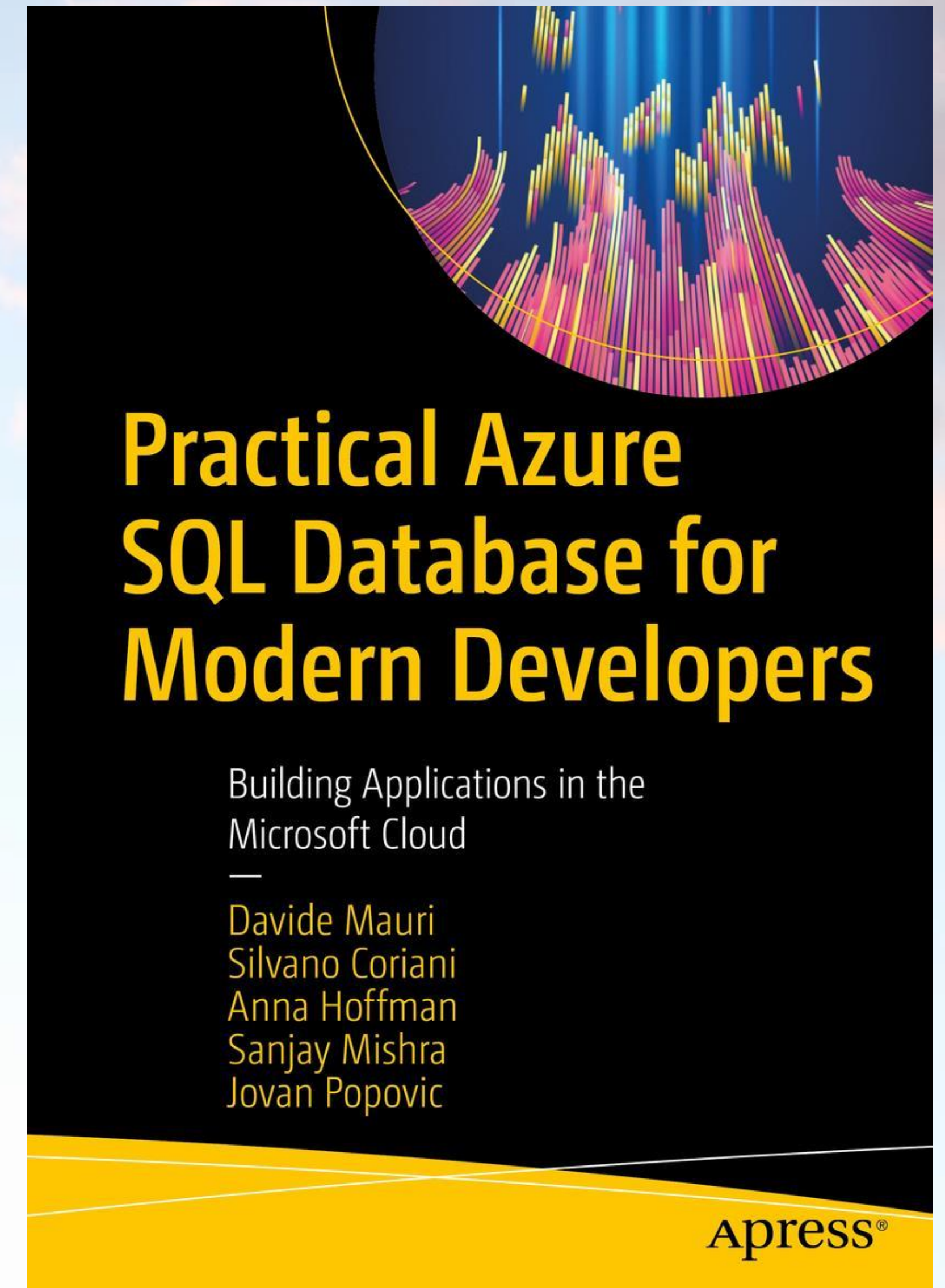
Microsoft Data Platform for 12 years

Azure SQL PM

- Focus on Azure SQL Hyperscale & Developers

<http://davidemauri.it>

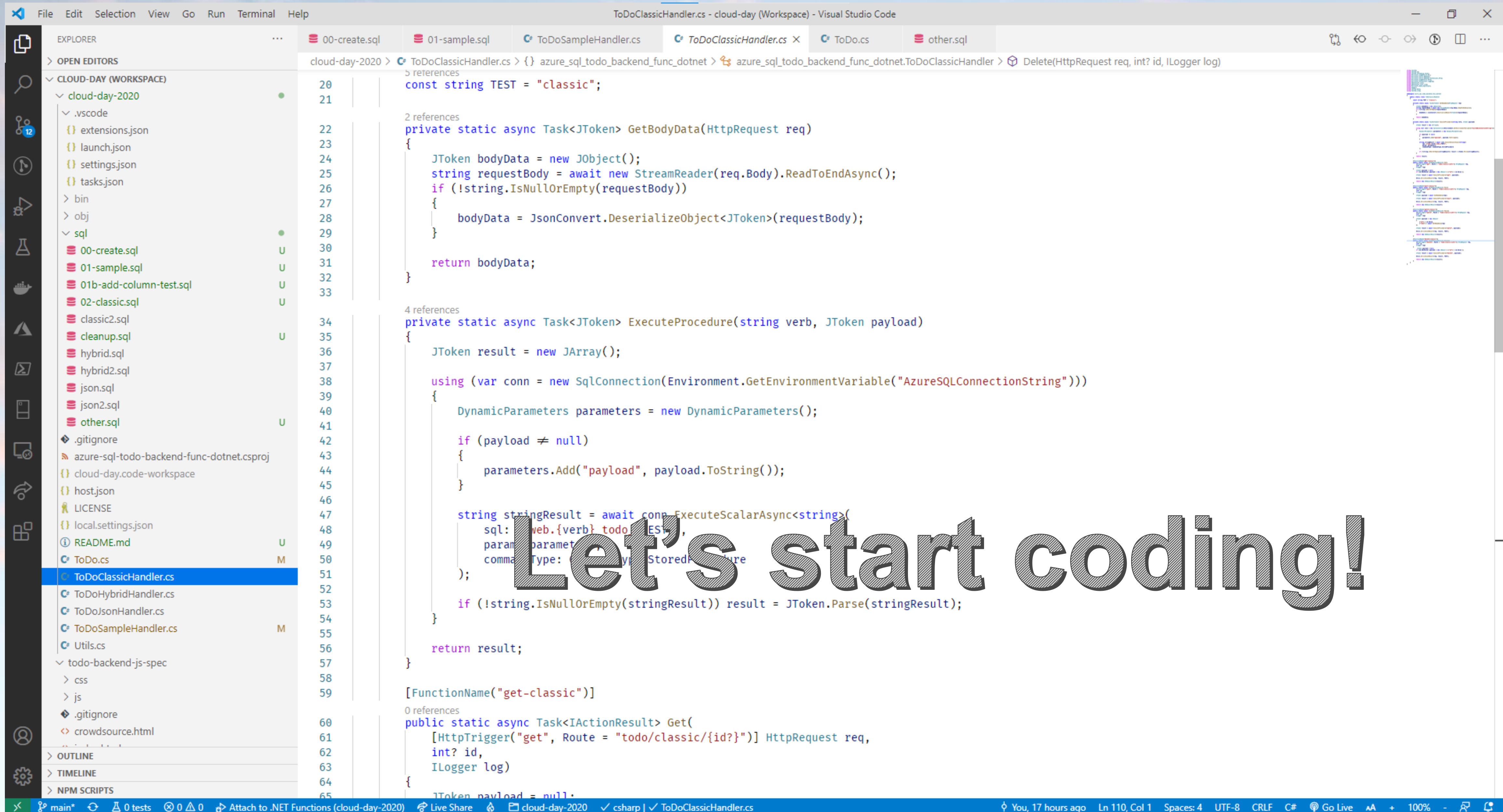
<https://aka.ms/azure-sql-db-dev-samples>



Agenda

Implement TodoMVC Backend API with [Azure Functions](#)

- Allow extensibility with the minimum amount of code
- GET with Dapper, PUT/POST challenges, [JSON](#) as transport
- [JSON as batching technique](#)
- Hybrid Data Models: Fully Relational, Some JSON, More JSON



The screenshot shows the Visual Studio Code interface with the following details:

- Explorer:** Shows a workspace named 'cloud-day-2020' containing a project 'azure_sql_todo_backend_func_dotnet'. The 'sql' folder is expanded, listing files like '00-create.sql', '01-sample.sql', '02-classic.sql', etc.
- Code Editor:** Displays the code for 'ToDoClassicHandler.cs'. The code includes:


```

const string TEST = "classic";

private static async Task<JToken> GetBodyData(HttpRequest req)
{
    JToken bodyData = new JObject();
    string requestBody = await new StreamReader(req.Body).ReadToEndAsync();
    if (!string.IsNullOrEmpty(requestBody))
    {
        bodyData = JsonConvert.DeserializeObject<JToken>(requestBody);
    }

    return bodyData;
}

private static async Task<JToken> ExecuteProcedure(string verb, JToken payload)
{
    JToken result = new JObject();

    using (var conn = new SqlConnection(Environment.GetEnvironmentVariable("AzureSQLConnectionString")))
    {
        DynamicParameters parameters = new DynamicParameters();

        if (payload != null)
        {
            parameters.Add("payload", payload.ToString());
        }

        string stringResult = await conn.ExecuteScalarAsync<string>(
            sql: "web.{verb} todo {verb} ES",
            param: parameters,
            commandType: CommandType.StoredProcedure);

        if (!string.IsNullOrEmpty(stringResult)) result = JToken.Parse(stringResult);
    }

    return result;
}

[FunctionName("get-classic")]
public static async Task<IActionResult> Get(
    [HttpTrigger("get", Route = "todo/classic/{id?}")] HttpRequest req,
    int? id,
    ILogger log)
{
    JToken payload = null;

```
- Watermark:** A large, stylized, semi-transparent text overlay reads "Let's start coding!".
- Status Bar:** Shows the current file is 'ToDoClassicHandler.cs' in the 'csharp' language, with a cursor at line 110, column 1.

JSON in your Azure SQL Database? Let's benchmark some options!



Silvano

June 10th, 2020

Introduction

Storing and retrieving data from JSON fragments is a common need in many application scenarios, like IoT solutions or microservice-based architectures. You can persist these fragments can be in a variety of data stores, from blob or file shares, to relational and non-relational databases, and there's a long standing debate in the industry on what's the database technology that fits "better" for this task.

Azure SQL Database offers several [options](#) for parsing, transforming and querying JSON data, and this article doesn't pretend to provide a definitive answer to that debate, but rather to explore these options for common scenarios like data loading and retrieving, and benchmarking results to provide a clear indication of how Azure SQL Database will perform manipulating JSON data.

Kudos



managed/designs

