

# VF01 Hands-on Workshop: Build Modern Applications with Azure SQL Database and SQL Server

Your Code Powers the World,  
Our Training Powers You.

# Session Survey

- Your feedback is very important to us
- Please take a moment to complete the session survey found in the mobile app
- Use the QR code or search for “Converge360 Events” in your app store
- Find this session on the Agenda tab
- Click “Session Evaluation”
- Thank you!



# Today's Speakers



**Jerry Nixon**  
**Microsoft**



**Anagha Todabagi**  
**Microsoft**

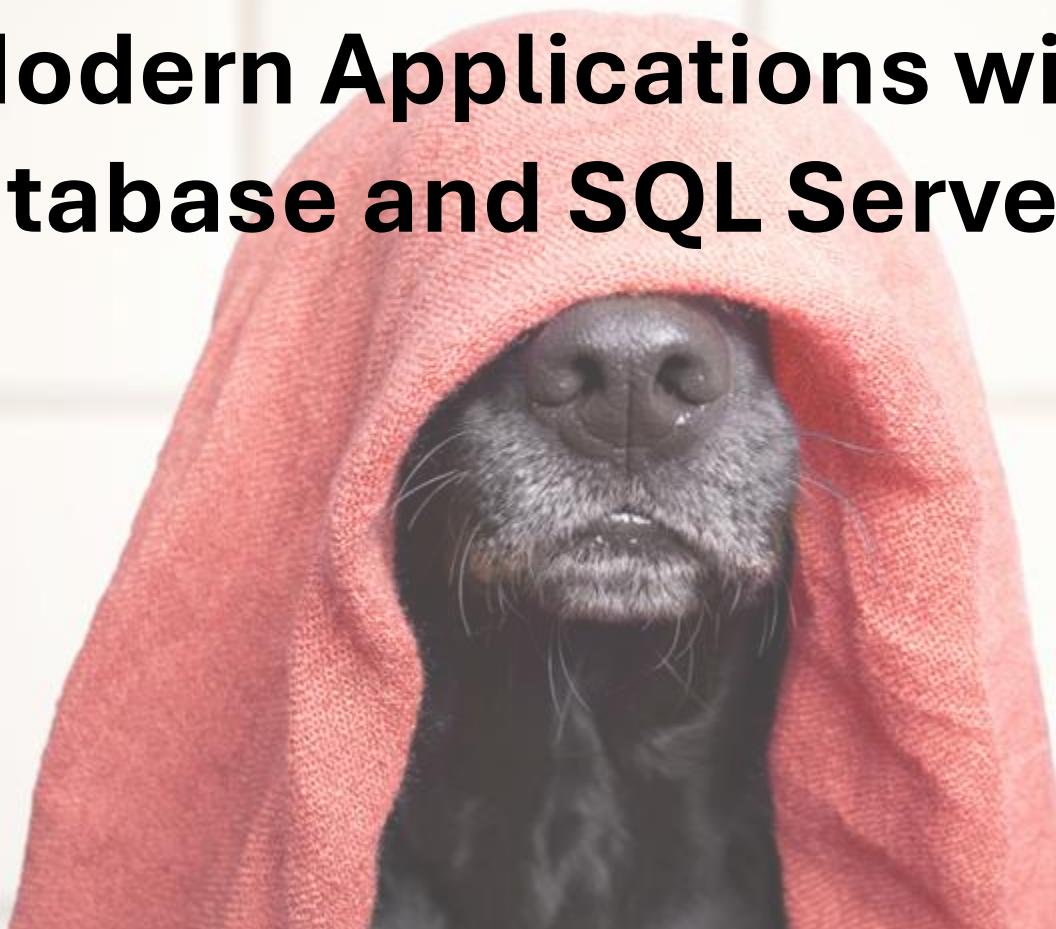


**Brian Spendolini**  
**Microsoft**

# **VF01 Hands-on Workshop:**

# **Build Modern Applications with Azure**

# **SQL Database and SQL Server**



# Agenda

Session Start	8:00am
Morning break	10:15 – 10:30am
Lunch	12:00 – 1:00pm (lunch will be in McKinley at the MS Conference Center)
Afternoon break	3:00 – 3:15pm
Session End	5:00pm

# WORKSHOP OVERVIEW



# Workshop Overview/Tasks

- Setup your environment
- Create database and objects
- Use Data API builder
- Applications with SWA
- Deploy to Azure SQL Database
- Invoke REST endpoints in the database
- Work with Azure OpenAI and friends
- Create a change data stream
- CICD with GitHub Actions

# Getting Started

## Accounts Needed

- GitHub account
- Azure account
  - Free Azure SQL Database

# Getting Started

## Development Environment



# GitHub



docker



Static  
Web Apps  
&  
Data API Builder



sqlcmd



Azure  
Functions

# Getting Started

## Development Environment



**GitHub**



**docker**



sqlcmd



Azure  
Functions



Static  
Web Apps  
&  
Data API Builder



Azure  
Functions



OpenAI

# CHAPTER 1: SETUPS



# Chapter 1: Setups

- Fork the repository
- Create the codespace

<https://aka.ms/forkit>

# **Setups**

## **Repository Contents**

### **.devcontainer**

- Codespace docker config

### **client**

- contains the Todo application for use with Data API builder and SWA

### **app**

- contains the sample JavaScript test application for use with Data API builder and SWA

### **docs**

- Has all the chapters/lessons of the workshop

### **scripts**

- Installs all the needed dev tools

### **labFiles**

- contains files to be used with the workshop in various chapters

# GitHub Codespaces

- A development environment that's hosted in the cloud
- Each codespace is hosted by GitHub in a Docker container
- Browser based environment
- Config files are contained in the repository

[main](#)

1 branch 0 tags

[Go to file](#)[Add file](#) ▾[Code](#) ▾

This branch is up to date with Azure-Samples/azure-sql-db-developers-workshop:main.

[Contribute](#) ▾ [Sync fork](#) ▾ **JetterMcTedder v1.5**

eadcad5 yesterday 167 commits

 .devcontainer	v1.5	last week
 .github	.github/PULL_REQUEST_TEMPLATE.md committed	4 months ago
 app	v1.5	5 days ago
 client	v1.5	5 days ago
 docs	v1.5	yesterday
 labFiles	v1.5	5 days ago
 scripts	v1.5	2 days ago
 CHANGELOG.md	v1.5	2 days ago
 CONTRIBUTING.md	v1.0	4 months ago
 LICENSE.md	LICENSE.md committed	4 months ago
 README.md	Updated README	3 weeks ago
 swa-cli.config.json	v1.5	last week

[README.md](#)

## About

A workshop for developing with the Azure SQL Database and Azure Services

[Readme](#)[MIT license](#)[Activity](#)[0 stars](#)[0 watching](#)[21 forks](#)

## Releases

No releases published

[Create a new release](#)

## Packages

No packages published

[Publish your first package](#)

## Languages



Azure SQL Database Developer Workshop Overview

Workshop Goals

```
graph LR; A((Start up development environment)) --> B((Create and deploy database objects)); B --> C((REST/ GraphQL enable the database)); C --> D((Explore native integration with Azure Services))
```

VS Code Extensions

File Explorer

Code Editor

Terminal Window

EXPLORER

AZURE-SQL-DB-DEVELOPERS-WORKSHOP-FORK [CODESPACES: jubilant system]

PREVIEW README.md

Microsoft

Azure SQL Database Developer Workshop Overview

Workshop Goals

Start up development environment → Create and deploy database objects → REST/ GraphQL enable the database → Explore native integration with Azure Services

Learn how to create a full local development environment  
Deploy a database and create tables with Visual Studio Code  
Utilize a Database Project for deployment into Azure SQL Database via CI/CD flows  
Expose data in your database via REST and GraphQL with Data API Builder  
Native integration with Azure Functions and other Azure Services from within the Azure SQL Database

What you will be building

You are an application developer and have been asked to create a To-do list for your company. This application has the following requirements:

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS AZURE COMMENTS

@JetterMcTeddler →/workspaces/azure-sql-db-developers-workshop-fork (main) \$

No Environment Layout US

# Utilities Check

Checks to run once the post-build scripts are finished:

Functions Core Tools

- `func --version`

Data API builder

- `dab --version`

`sqlcmd`

- `sqlcmd --version`

Static Web Apps

- `swa --version`



**START CHAPTER 1**



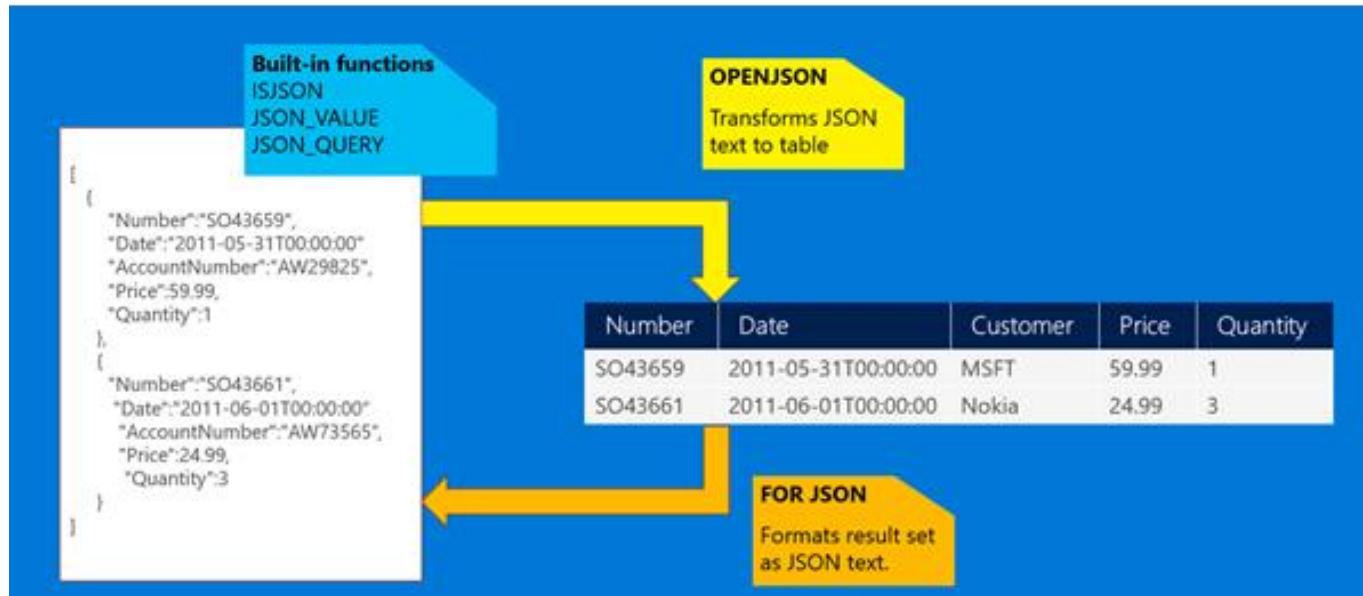
# **CHAPTER 2: CREATE A DATABASE AND OBJECTS**

# Azure SQL Database

- Features for developers
  - Multi-model support
    - Relational / JSON / Graph / Geospatial / XML
  - Row-Store and Column-Store
    - on the same table
  - Ledger Tables
  - Replicas / High-Availability / Scale-Out / Scale-Up
  - ***Preview – Vector Data Type***

# JSON Support

Azure SQL fully support storage, manipulation and creation of JSON documents



<https://docs.microsoft.com/sql/relational-databases/json/json-data-sql-server>

# JSON Support

**ISJSON:** Tests whether a string contains valid JSON.

**JSON\_PATH\_EXISTS:** Tests whether a specified SQL/JSON path exists in the input JSON string.

**JSON MODIFY:** Updates the value of a property in a JSON string and returns the updated JSON string.

**JSON\_VALUE:** Extracts a scalar value from a JSON string.

**JSON\_QUERY:** Extracts an object or an array from a JSON string.

**JSON\_OBJECT:** Constructs JSON object text from zero or more expressions.

**JSON\_ARRAY:** Constructs JSON array text from zero or more expressions.

**FOR JSON:** Converts SQL Server data types to JSON types.

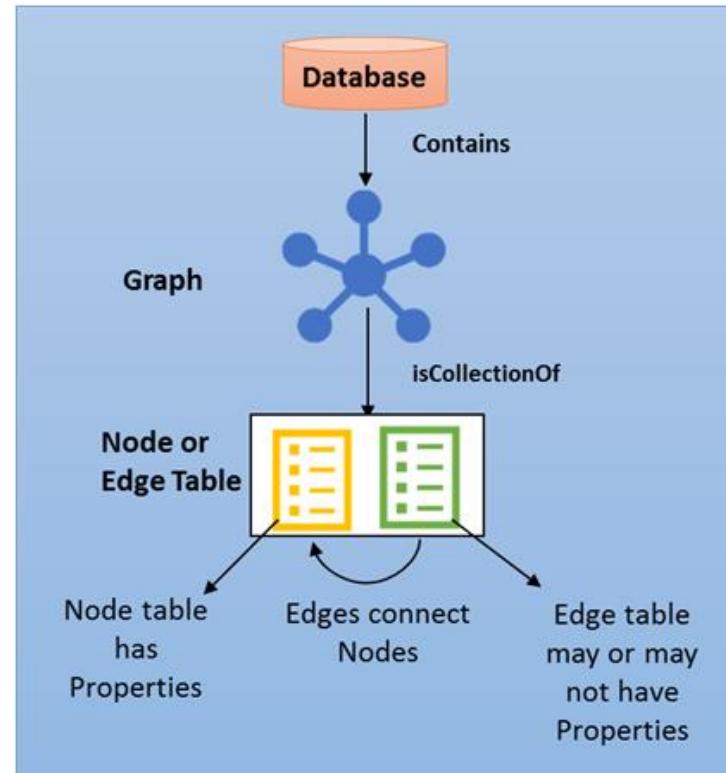
**OPENJSON:** Converts JSON text into a set of rows and columns.

# JSON Support

```
SELECT Name,  
       Surname,  
       JSON_VALUE(jsonCol, '$.info.address.PostCode') AS PostCode,  
       JSON_VALUE(jsonCol, '$.info.address."Address Line 1"')  
             + ' ' + JSON_VALUE(jsonCol, '$.info.address."Address Line 2"')  
             AS Address,  
       JSON_QUERY(jsonCol, '$.info.skills') AS Skills  
FROM People  
WHERE ISJSON(jsonCol) > 0  
      AND JSON_VALUE(jsonCol, '$.info.address.Town') = 'Belgrade'  
      AND STATUS = 'Active'  
ORDER BY JSON_VALUE(jsonCol, '$.info.address.PostCode');
```

# Graph Models

- Create EDGE and NODE objects
  - For simplicity they look like tables
- Use MATCH to find rows that satisfy some relationship condition
  - Syntax is close to Cypher language (supported by Neo4J)
- Getting started:
  - <https://aka.ms/azuresqlgraph>



# Graph Models

```
CREATE TABLE Person (
    ID INTEGER PRIMARY KEY,
    Name VARCHAR(100),
    Age INT)
AS NODE;
```

```
CREATE TABLE Friends (
    StartDate date)
AS EDGE;
```

--Find friends of John

```
SELECT Person2.Name
FROM Person Person1,
     Friends,
     Person Person2
WHERE MATCH(Person1-(Friends)-
>Person2)
AND Person1.Name = 'John';
```

# Graph Models

The screenshot shows the Neo4j Studio interface with a query results table. Annotations are present above the table:

- Node Properties:** Points to the first column of the table, which contains node details.
- Nodes that this edge connects:** Points to the second column, which contains edge details.
- Edge Properties:** Points to the third column, which contains edge properties.

**Code:**

```
SELECT Person2.Name Untitled-2
1  SELECT Person2.Name
2  FROM Person Person1,
3      friendOf,
4      Person Person2
5  WHERE MATCH(Person1-(friendOf)->Person2)
6  AND Person1.Name = 'John';
```

**RESULTS**

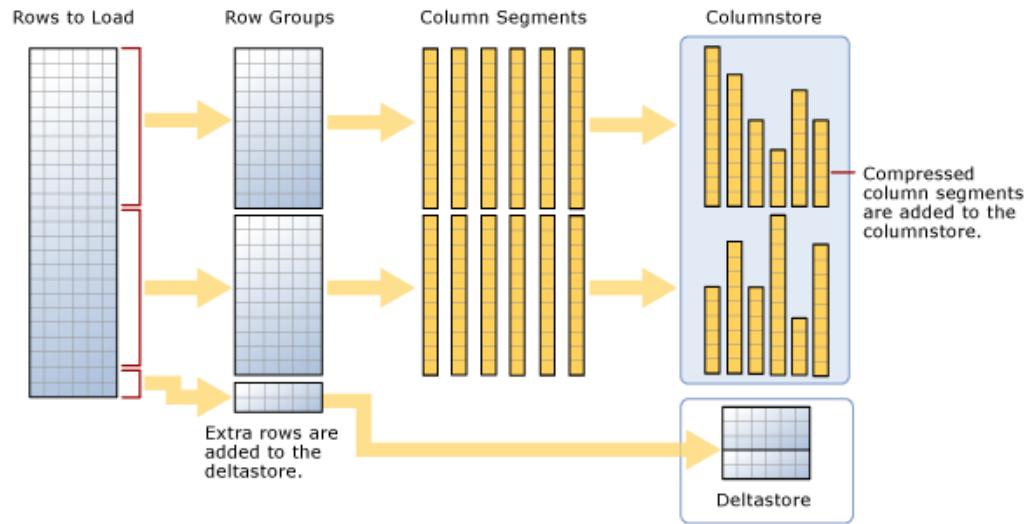
	Name
1	Mary

**Person Node Table**

**Friends Edge Table**

# Columnstore

- Store data by columns instead of by rows
- Great for analytical queries
  - Aggregations / Projections
- Columnstore is updatable
  - available from 2016
  - “Deltastore”



# Reasons for Columnstore Indexes

- Columns store values that commonly have similar values, which result in high compression rates.
- High compression rates improve query performance by using a smaller in-memory footprint
- Queries often select only a few columns from a table, which reduces total I/O from the physical media.

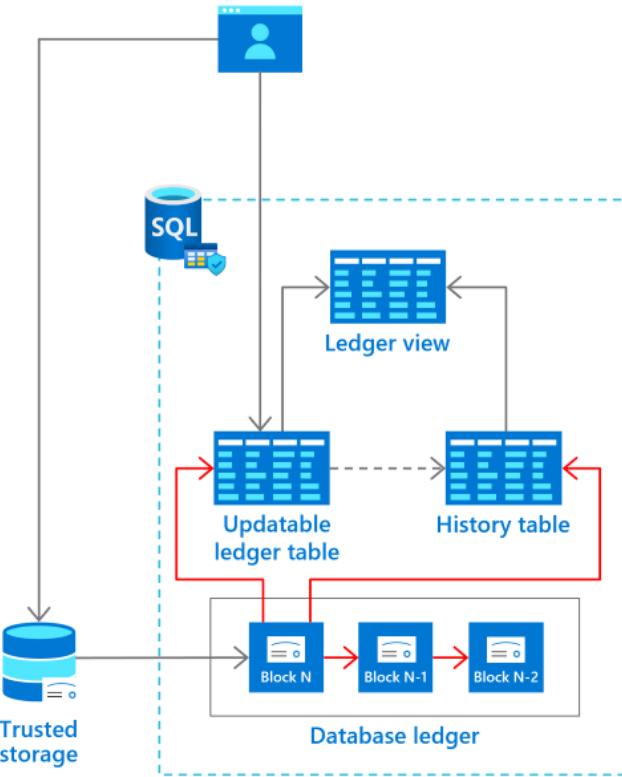
# Ledger Tables

## Updatable ledger tables

Updatable ledger tables are system-versioned tables on which users can perform updates and deletes while also providing tamper-evidence capabilities.

When updates or deletes occur, all earlier versions of a row are preserved in a secondary table, while the latest version of the row remains in the ledger table.

Ideal for systems/tables where auditing and record accuracy are vital.



# Ledger Tables

```
CREATE TABLE [dbo].[HighScores]
(
    [ScoreID] INT IDENTITY,
    [PlayerID] INT NOT NULL,
    [FirstName] VARCHAR (50) NOT NULL,
```

CommitTime	UserN...	Sco...	Pla...	Firs...	LastNa...	Game	Score	Date	Operation
2023-04-20 23:20:49...	bill	2	2	Bill	Mickelson	Horse Monkey	11099913	2023-04-20	INSERT
2023-04-20 23:20:49...	bill	2	2	Bill	Mickelson	Horse Monkey	11099911	2023-04-20	DELETE
2023-04-20 23:20:35...	sqladmin	1	1	Steve	Wobble	Horse Monkey	11099912	2023-04-20	INSERT
2023-04-20 23:20:35...	sqladmin	3	3	John	Hill	Universe Invaders	218870	2023-04-20	INSERT
2023-04-20 23:20:35...	sqladmin	2	2	Bill	Mickelson	Horse Monkey	11099911	2023-04-20	INSERT

LEDGER = ON );

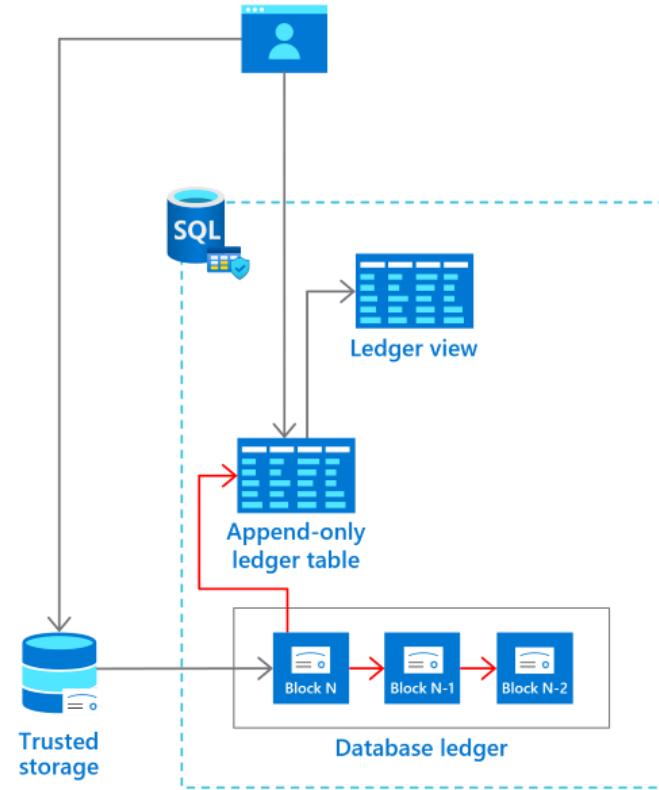
```
INSERT INTO [dbo].[HighScores]
VALUES    (1, 'Steve', 'Wobble', 'Horse Monkey', 11099912, GETDATE()),
          (2, 'Bill', 'Mickelson', 'Horse Monkey', 11099911, GETDATE()),
          (3, 'John', 'Hill', 'Universe Invaders', 218870, GETDATE());
```

# Ledger Tables

## Append-only ledger tables

Append-only ledger tables allow only INSERT operations on your tables, which ensure that privileged users such as database administrators can't alter data through traditional Data Manipulation Language operations.

Ideal for security information event and management systems such as badge in/out or ID readers.



# Ledger Tables

```
UPDATE [dbo].[ArcadeRoomAccess]  
SET Timestamp = GETDATE()  
WHERE AccessID = 4;
```

Msg 37359, Level 16, State 1, Line 1  
Updates are not allowed for the  
append only Ledger table 'dbo.ArcadeRoomAccess'.

# Azure SQL DB Hyperscale

Azure SQL Database Hyperscale provides the flexibility and performance modern applications require:

- Independently scalable compute and storage, without sacrificing performance
- Rapid, auto-scaling storage support up to 100TB
- Low latency, high throughput
- Snapshot-based backups that do not impact query performance
- Available serverless compute and resource-sharing elastic pools

**Now with lower pricing for Azure SQL Database Hyperscale!**

# Regex support in Azure SQL DB

- POSIX standard
- Case sensitivity, character classes, quantifiers, anchors, and capturing groups.
- More complex and powerful search patterns.
- In private preview across all tiers of Azure SQL Database.

REGEXP\_LIKE

REGEXP\_COUNT

REGEXP\_INSTR

REGEXP\_REPLACE

REGEXP\_SUBSTR

Regex in Azure SQL Database

Private Preview Sign-up



# Regex support in Azure SQL DB

-- find all the employees whose email addresses are valid and end with .com

```
SELECT [Name], Email  
FROM Employees  
WHERE REGEXP_LIKE(Email, '^A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.\com$');
```

-- for each employee, count the number of valid email addresses in the email column.

```
SELECT COUNT(*) AS Valid_email_count  
FROM Employees  
WHERE REGEXP_COUNT(Email, '^a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$') > 0
```

-- for each employee, show their name, email, and the position of the @ sign in their email

```
SELECT [Name], Email, REGEXP_INSTR(email, '@') AS Position_of_@  
FROM Employees;
```

-- format the phone numbers in the Employees table to the format (XXX) XXX-XXXX or mask few digits.

```
SELECT Phone_Number, REGEXP_REPLACE(Phone_Number, '(\d{3})-(\d{3})-(\d{4})', '(\1) \2-\3!\1') AS Phone_Format, REGEXP_REPLACE  
(PHONE_NUMBER, '\d{4}$', '****') AS 'Masked'  
FROM Employees;
```

-- for each employee, show the domain of their email address

```
SELECT [Name], Email, REGEXP_SUBSTR(email, '@(.+)$', 1, 1,'c',1) AS Domain  
FROM Employees;
```

# Vector Data Type

A vector is an array of ordered scalar values.

Vector is essentially a point in N-Dimensional space which is represented as an array of K floats/int of certain size. A vector with 3 dimensions would look like

**[0.4840199947357178, 0.09077099710702896, -0.22439000010490417]**

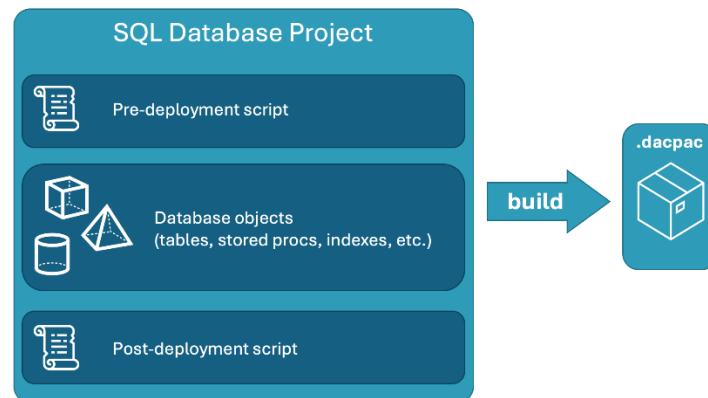
The new type named VECTOR will provide a high-fidelity data type that can store vector data optimized for operations such as similarity search and machine learning applications.

# Chapter 2: Database Tasks

- Create a database SQL project
- Create a SQL 2022 database in the codespace
- Create database objects
- Publish objects to the local SQL 2022 database

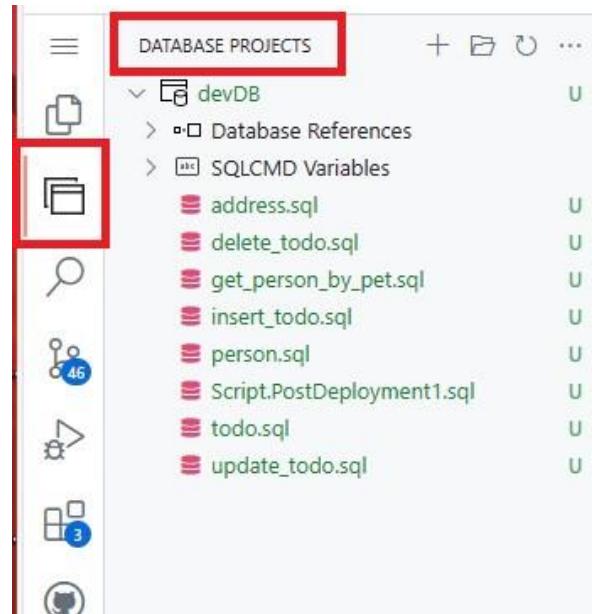
# SQL Database Projects

- An Azure Data Studio and Visual Studio Code extension
- For developing SQL databases and objects in a project-based development environment
- Update/Create project from database
- Compare schemas
- Publish all or just changes
- Participates in CI/CD flows



# SQL Database Projects

- `dotnet new sqlproj -n "devDB" -tp "SqlAzureV12"`



# Create a SQL Database

- SQLCMD CLI
  - Written in Go
- Use the go-mssql driver
- More Azure Active Directory (Entra ID) authentication options
- New commands
  - `sqlcmd create mssql --using https://url.com/backup.bak`

# Create a SQL Database

- `sqlcmd create mssql -u devDB --accept-eula`
- `sqlcmd config connection-strings`

```
● @JetterMcTedder → /workspaces/azure-sql-db-developers-workshop/database (main) $ sqlcmd create mssql -u devDB --accept-eula
Downloading mcr.microsoft.com/mssql/server:latest
Starting mcr.microsoft.com/mssql/server:latest
Created context "mssql" in "/home/vscode/.sqlcmd/sqlconfig", configuring user account...
Disabled "sa" account (and rotated "sa" password). Creating user "vscode"
Creating default database [devDB]
Now ready for client connections on port 1433
```

HINT:

1. Run a query: `sqlcmd query "SELECT @@version"`
2. Start interactive session: `sqlcmd query`
3. View sqlcmd configuration: `sqlcmd config view`
4. See connection strings: `sqlcmd config connection-strings`
5. Remove: `sqlcmd delete`

# Create Database Objects

- Tables
  - ID columns
  - Foreign/Primary Keys
- Table APIs - Stored procedures for
  - Insert
  - Update
  - Delete

## Person

Name	Data Type
person_id	int (IDENTITY)
person_name	nvarchar(200)
person_email	nvarchar(200)
pet_preference	nvarchar(100)

## Address

Name	Data Type
address_id	int (IDENTITY)
person_id	int
address	nvarchar(200)

FK

## Todo

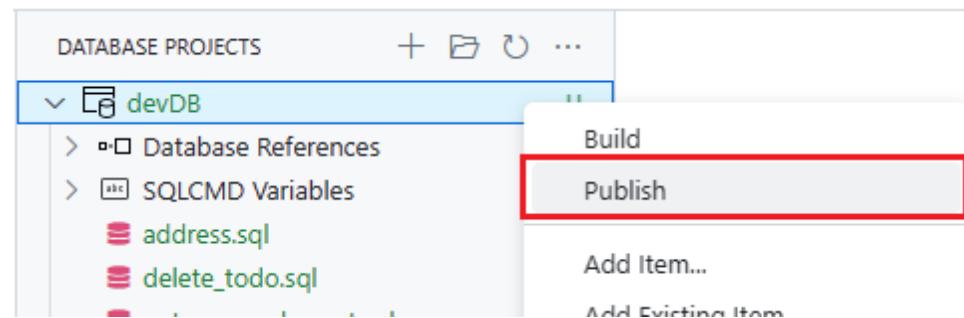
Name	Data Type
id	uniqueidentifier
title	nvarchar(1000)
completed	bit
owner_id	nvarchar(126)
position	int

## Products

Name	Data Type
product_id	int
product_name	nvarchar(1000)
ListPrice	money

# Publish objects to the local database

- Use SQL Database Projects
  - Publish to the local database created with sqlcmd
- Deploy the tables, stored procedures and run a post deploy script
- Verify the deployment



# Testing the Table APIs Reminder

You can also test out the stored procedures with the following code:

```
exec dbo.insert_todo @title = 'My Test Todo', @owner_id = '1001001', @order = 1;
select * from dbo.todo;
GO
```

Copy and paste the resulting ID from the result set into the area marked **COPIED ID FROM RESULT SET** and use it in the next stored procedure example:

```
exec dbo.update_todo @id = 'COPIED ID FROM RESULT SET', @title = 'zzzz',
@owner_id = '1001001';
select * from dbo.todo;
GO
```

And again, copy and paste the resulting ID from the result set into the area marked **COPIED ID FROM RESULT SET** and use it in the next stored procedure example:

```
exec dbo.delete_todo @id = 'COPIED ID FROM RESULT SET', @owner_id = '1001001';
select * from dbo.todo;
GO
```

# **START CHAPTER 2**



# **CHAPTER 3:**

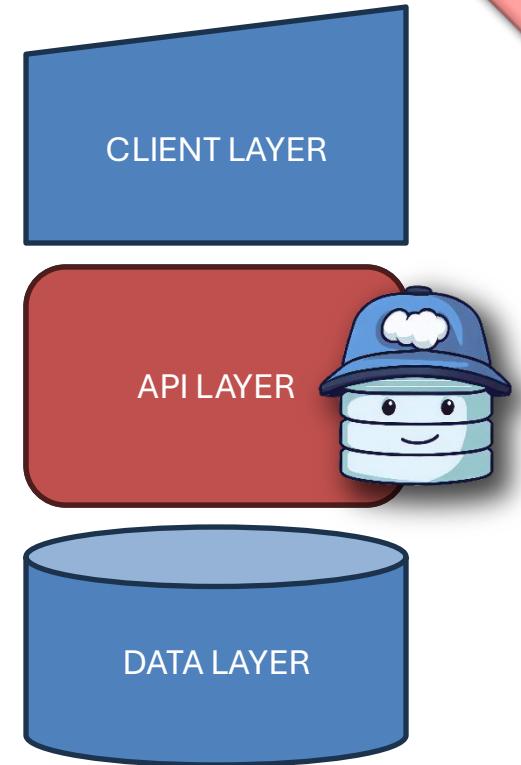
## **DATA API BUILDER**



Now in GA!!!

# Data API builder

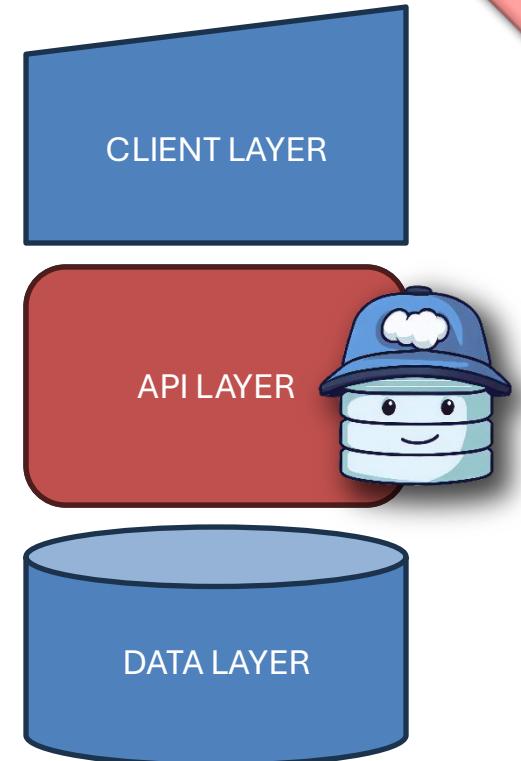
- Open Source and Free (<https://aka.ms/dab>)
- Container-based
- Zero code
  - Configuration-driven
- Supported authentication:
  - EasyAuth, EntralID/AAD, JWT tokens
- Supported endpoints:
  - Standard REST API and Standard GraphQL
- Supported databases:
  - Azure SQL DB & SQL Server
  - Cosmos DB (document)
  - PostgreSQL
  - MySQL



Now in GA!!!

# Data API builder

- Built in developer tools and clients
  - OpenAPI, Swagger, Banana Cake Pop
  - DAB CLI
- Database Objects Supported
  - Tables
  - Views
  - Stored Procedures
- GraphQL Relationships
  - Foreign Keys
  - One-to-Many/Many-to-One/Many-to-Many
- Authorization
  - Database Context
  - Role-based authorization using received claims
  - Item-level security via policy expressions



"Plain"  
GraphQL  
Schema



or  
Relational  
Schema



Configuration  
File



Entity Types only  
**Provided by Developer**

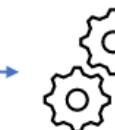
# Data API builder

Runtime



In-Memory  
Schema Abstraction

Runtime



REST  
Methods



REST Endpoint



GraphQL  
Endpoint

Entity Types  
Input/Output Types  
Queries  
Mutations



GraphQL  
Schema

Generated by Data API builder engine

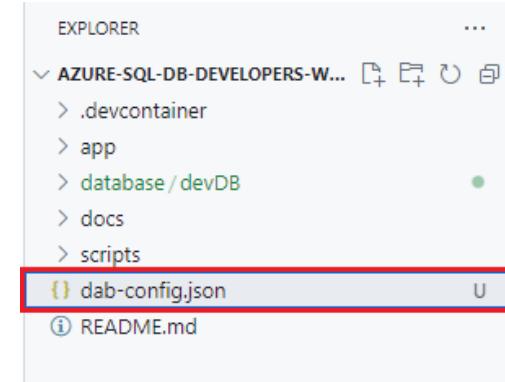
# Chapter 3: Data API Builder

- Setup Data API Builder
- Create REST/GraphQL entities
- Test the endpoints

# Setup Data API Builder

- Create the Data API Builder initialization file

```
dab init --database-type "mssql" --connection-string  
"Server=localhost;Database=devDB;User  
ID=vscode;Password='PASSWORD';TrustServerCertificate=true"  
--host-mode "Development" --rest.path "rest" --set-  
session-context true
```

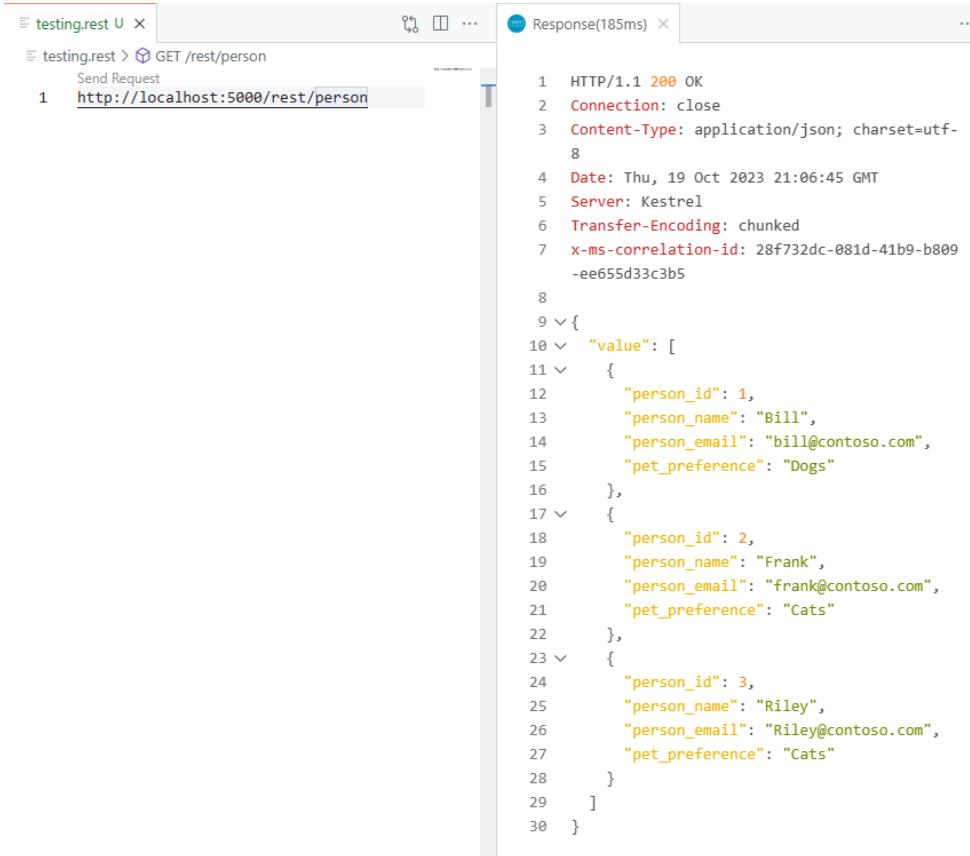


# Create REST/GraphQL entities

Use DAB CLI to add the tables, stored procedures, and relationships to the dab-config.json file

- **dab add person --source dbo.person --permissions "anonymous:\***"
- **dab update person --relationship "address" --cardinality "many" --target.entity "address"**
- **dab add getPersonByPet --source dbo.get\_person\_by\_pet --source.type "stored-procedure" --source.params "pet:" --permissions "anonymous:execute" --rest.methods "get" --graphql.operation "query"**

# Test the endpoints



The screenshot shows a browser window with two tabs. The left tab is titled 'testing.rest U X' and shows the URL 'http://localhost:5000/rest/person'. The right tab is titled 'Response(185ms) X' and displays the following JSON response:

```
1  HTTP/1.1 200 OK
2  Connection: close
3  Content-Type: application/json; charset=utf-
8
4  Date: Thu, 19 Oct 2023 21:06:45 GMT
5  Server: Kestrel
6  Transfer-Encoding: chunked
7  x-ms-correlation-id: 28f732dc-081d-41b9-b809
   -ee655d33c3b5
8
9  [
10 [
11 {
12   "person_id": 1,
13   "person_name": "Bill",
14   "person_email": "bill@contoso.com",
15   "pet_preference": "Dogs"
16 },
17 {
18   "person_id": 2,
19   "person_name": "Frank",
20   "person_email": "frank@contoso.com",
21   "pet_preference": "Cats"
22 },
23 {
24   "person_id": 3,
25   "person_name": "Riley",
26   "person_email": "Riley@contoso.com",
27   "pet_preference": "Cats"
28 }
29 ]
30 }
```

Use codespaces to test your REST Endpoints

# Test the endpoints

The screenshot shows a GraphQL testing interface with the following content:

```
Operation Schema Reference Schema Definition
Builder Request Response
Run D: Run B: Run R:
1 3 Get the first 5 records
2
3
4 people(first: 5) {
5   items {
6     person_id
7     person_name
8     person_email
9   }
10
11 }
12
13 Using an orderBy to sort the results
14
15 Run D: Run B: Run R:
16 people(orderBy: { person_id: DESC } ) {
17   items {
18     person_id
19     person_name
20     person_email
21   }
22 }
23
24
25
26 Get a record by primary key
27
28 Run D: Run B: Run R:
29 person_by_pk(person_id: 1) {
30   person_id
31   person_name
32   person_email
33   pet_preference
34 }
35 }
```

Use codespace or  
Banana Cake Pop to  
test your GraphQL  
endpoints

# START CHAPTER 3

A photograph of a large adult llama and a smaller cria standing in a grassy field. The adult llama is on the left, facing right, with its head turned slightly back towards the cria. The cria is standing next to it, facing left. In the background, there are more llamas and a range of mountains under a clear sky.

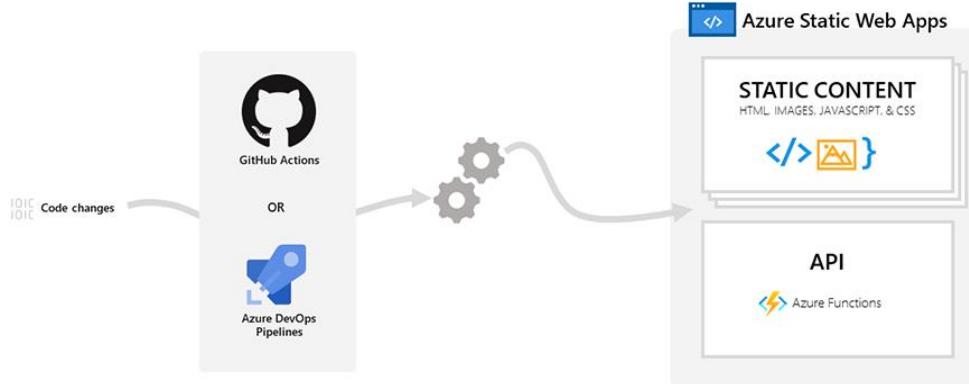
# CHAPTER 4: AZURE STATIC WEB APPS



# Azure Static Web Apps

Automatically build and deploy full stack web apps to Azure from a code repository.

- **Build modern web applications** with JavaScript frameworks and libraries like [Angular](#), [React](#), [Svelte](#), [Vue](#), or using [Blazor](#) to create WebAssembly applications, with an [Azure Functions](#) back-end.
- **Publish static sites** with frameworks like [Gatsby](#), [Hugo](#), [VuePress](#).
- **Deploy web applications** with frameworks like [Next.js](#) and [Nuxt.js](#).



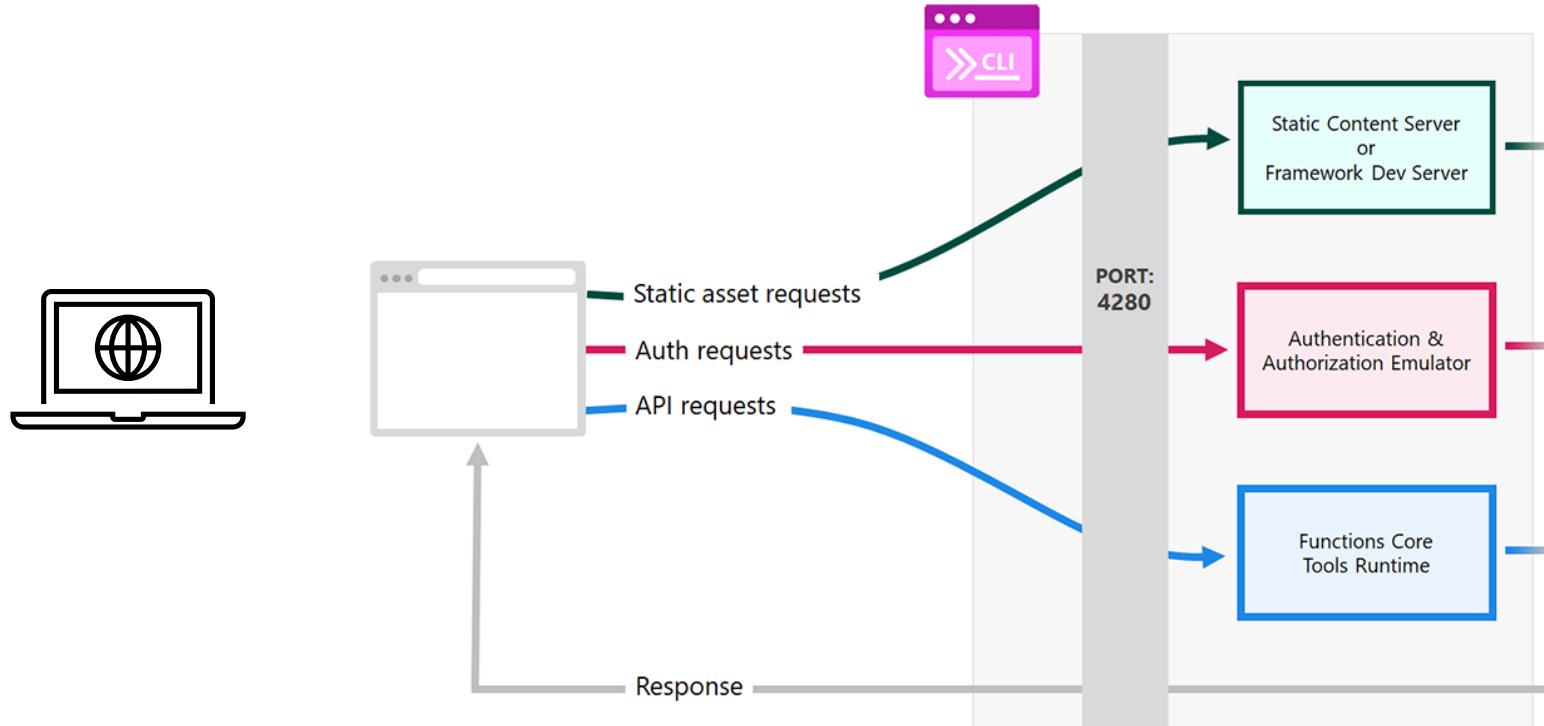
# Key Features

- **Web hosting** for static content like HTML, CSS, JavaScript, and images.
- **Integrated API** support provided by managed Azure Functions, with the option to link an existing function app, web app, container app, or API Management instance using a standard account. If you need your API in a region that doesn't support [managed functions](#), you can [bring your own functions](#) to your app.
- **First-class GitHub and Azure DevOps integration** that allows repository changes to trigger builds and deployments.
- **Globally distributed** static content, putting content closer to your users.
- **Free SSL certificates**, which are automatically renewed.
- **Custom domains** to provide branded customizations to your app.
- **Seamless security model** with a reverse-proxy when calling APIs, which requires no CORS configuration.
- **Authentication provider integrations** with Microsoft Entra ID and GitHub.
- **Customizable authorization role definition** and assignments.
- **Back-end routing rules** enabling full control over the content and routes you serve.
- **Generated staging versions** powered by pull requests enabling preview versions of your site before publishing.
- **CLI support** through the [Azure CLI](#) to create cloud resources, and via the [Azure Static Web Apps CLI](#) for local development.

# Azure Static Web Apps CLI

- Serve static app assets, or proxy to your app dev server
- Serve API requests, or proxy to APIs running in Azure Functions Core Tools
- Emulate authentication and authorization
- Emulate Static Web Apps configuration, including routing and ACL roles
- Deploy your app to Azure Static Web Apps

# Azure Static Web Apps CLI



# Chapter 4: SWA

- Initialize SWA
- Test the web applications
- Use SWA authentication with Data API builder

# But First....

Creating a mapping in Data API Builder

```
> dab update todo --map "position:order"
```

To map a UI element called order to the position column (in dab-config.json)

```
"mappings": {  
    "position": "order"  
}
```

# Initialize SWA and Test

Create the connections file and start with the sample JavaScript application

- 1) cp dab-config.json ./swa-db-connections/staticwebapp.database.config.json
- 2) swa start --app-location ./app --data-api-location ./swa-db-connections

## Sample JavaScript App

Pet Preference:

**Submit**

Person ID	Person Name	Person Email	Pet Preference
1	Bill	bill@contoso.com	Dogs
2	Frank	frank@contoso.com	Cats
3	Riley	Riley@contoso.com	Cats

[Login](#)

# Use SWA authentication with Data API builder

In the permissions section, change the role from **anonymous** to **authenticated**

```
"permissions": [  
  {  
    "role": "authenticated",  
    "actions": [  
      "execute"  
    ]  
  }  
,
```

Username

Username or email address of the user

User's roles

Roles used during authorization. One role per line.  
Note: roles "authenticated" and "anonymous" will be added automatically if not provided.

User's claims

Claims from the identity provider. JSON array of claims. See [documentation](#) for example claims.

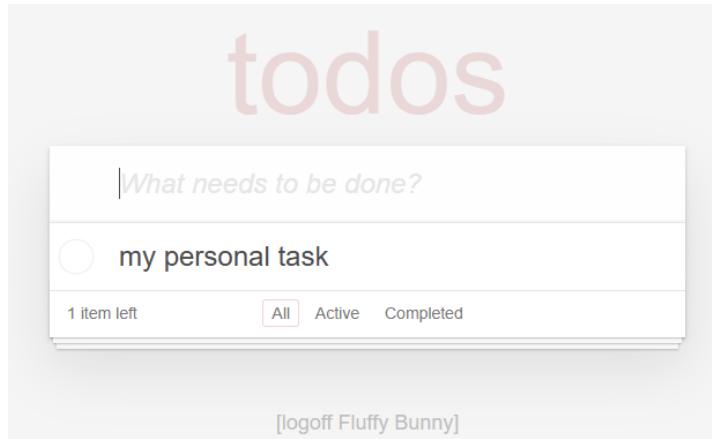
# Working with the ToDo Application

- Change the `swa-cli.config.json`
- Build the app (`swa build`)
- Start swa and test the ToDo application

```
swa start --data-api-location ./swa-db-connections
```

# Working with the ToDo Application

In the permissions section, change replace the permissions section for the Todo table with the following code:



```
"permissions": [
  {
    "role": "anonymous",
    "actions": [
      {
        "action": "*",
        "policy": {
          "database": "@item.owner_id eq 'public'"
        }
      }
    ]
  },
  {
    "role": "authenticated",
    "actions": [
      {
        "action": "*",
        "policy": {
          "database": "@item.owner_id eq @claims.userId"
        }
      }
    ]
  }
],
```



**START CHAPTER 4**

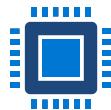
# CHAPTER 5:

# DEPLOY TO AZURE SQL DATABASE



# Azure SQL Database free of charge

Get 100,000 vCore seconds of serverless compute and 32 GB of storage every month!



## Azure SQL Database with serverless compute

Flexible compute automatically scales to meet demand.

## No time limits

Apply this free offer for the life of your subscription.

## Need more? No problem.

Stick with the default auto-pause option or continue usage for additional charges.

### What's included:



One Azure SQL Database with serverless compute per Azure Subscription with 100,000 vCore seconds every month.



32 GB data storage +  
32 GB backup storage.

**Learn More:** [aka.ms/sqlfreeoffer](https://aka.ms/sqlfreeoffer)



Want to try Azure SQL Database for free? Create a free serverless database with the first 100,000 vCore seconds, 32GB of data, and 32GB of backup storage free per month for the lifetime of the subscription. [Learn more](#)

**Apply offer (Preview)**

# Chapter 5: Deploy to Azure

- Create a Free Azure SQL Database
- Connect to the Azure SQL Database
- Publish objects to the Azure SQL Database

# Create a Free Azure SQL Database

- Login to the Azure Portal
- Create a new SQL Database
- Apply the promotion
- Create!

The screenshot shows the Azure portal interface for creating a new SQL database. At the top, there's a blue 'SQL' button. Below it, a 'Cost summary' section displays the following details:

General Purpose (GP_S_Gen5_2)	
Cost per GB (in USD)	0.00
Max storage selected (in GB)	x 41.6
First 32 GB storage free	
First 100,000 vCore seconds free	
Overage billing <sup>1</sup>	Disabled
ESTIMATED STORAGE COST / MONTH	0.00 usd
COMPUTE COST / VCORE SECOND <sup>2</sup>	0.000000 usd

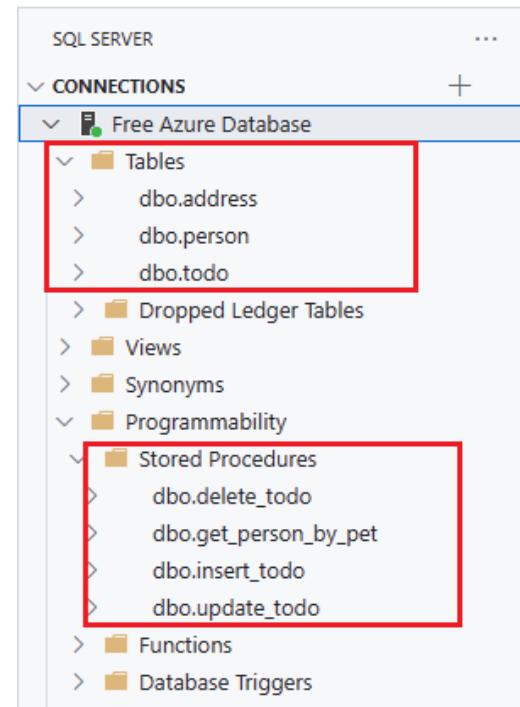
At the bottom, two footnotes provide additional information:

<sup>1</sup> There will be no charges for usage within the free limits. The database will be paused automatically when the free limits are reached.

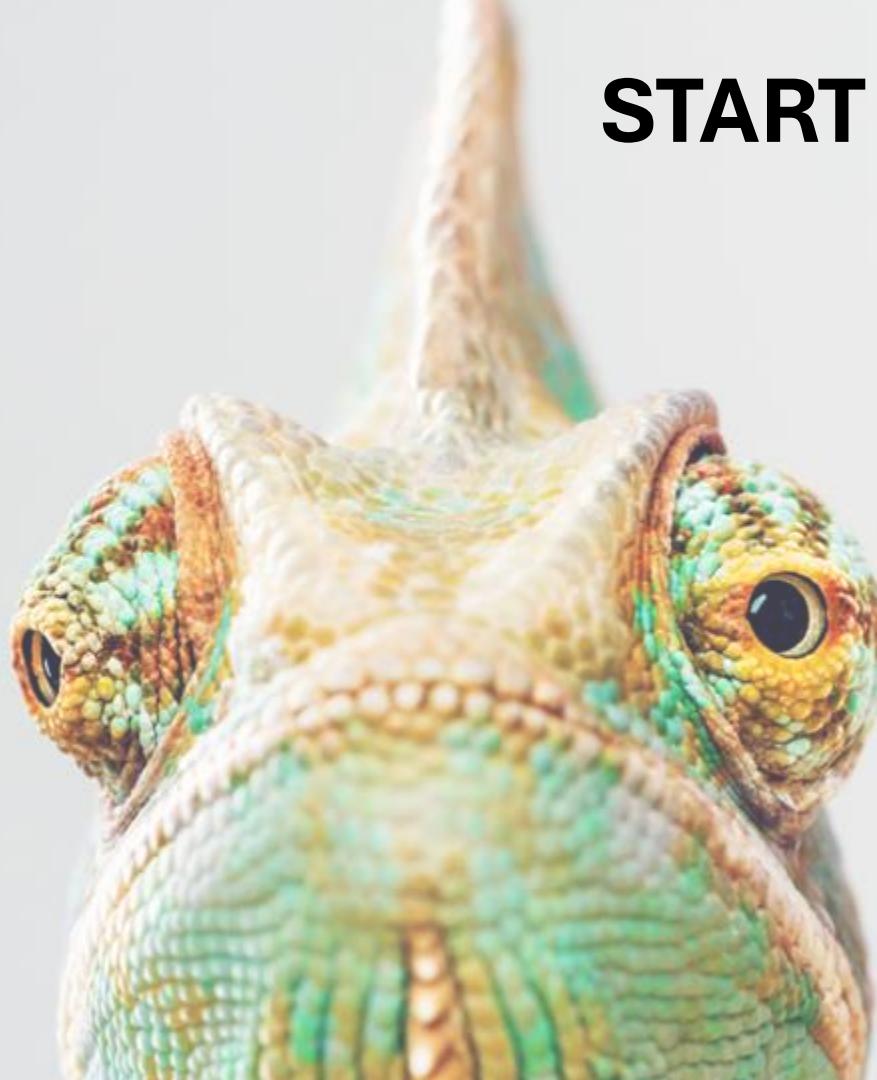
<sup>2</sup> Serverless databases are billed in vCore seconds based on a combination of CPU and memory utilization. [Learn more about serverless billing](#)

# Connect and Publish to the Azure SQL Database

- Create a connection in codespace
- Deploy the database project to the free Azure SQL Database
- Verify the deployment and post-install script



# START CHAPTER 5



# CHAPTER 6: JSON IN THE DATABASE



# JSON Features

- Native JSON data type
- T-SQL value and array functions
- T-SQL convert to and from JSON functions
- T-SQL construct JSON functions

# JSON Functions

**ISJSON**: Tests whether a string contains valid JSON.

**JSON\_PATH\_EXISTS**: Tests whether a specified SQL/JSON path exists in the input JSON string.

**JSON MODIFY**: Updates the value of a property in a JSON string and returns the updated JSON string.

ANSI SQL JSON functions

**JSON\_VALUE**: Extracts a scalar value from a JSON string.

**JSON\_QUERY**: Extracts an object or an array from a JSON string.

ANSI SQL JSON constructors

**JSON\_OBJECT**: Constructs JSON object text from zero or more expressions.

**JSON\_ARRAY**: Constructs JSON array text from zero or more expressions.

# JSON Functions

The available operators are:

**FOR JSON**: Converts SQL Server data types to JSON types.

**OPENJSON**: Converts JSON text into a set of rows and columns.

**New** in the Azure SQL Database:

**JSON\_OBJECTAGG**: Construct a JSON object value from relational data

**JSON\_ARRAYAGG**: Construct a JSON array value from relational data

# Chapter 6: JSON in the Database

- Querying JSON Data
  - JSON data type
- Working with JSON Data
  - JSON\_ARRAYAGG
  - JSON\_OBJECTAGG





**START CHAPTER 6**

# **CHAPTER 7: INVOKE REST ENDPOINTS**



# Services Integration

I really want to use other Azure products and services from within my Azure SQL Database without having to write complex code, functions, or scheduled events. Is this even possible?

# External REST Endpoint Invocation

Using a single stored procedure, call REST/GraphQL service endpoints in Azure directly from within an Azure SQL Database.



# Use Cases

- Activate workflows
- Data enrichment
- Cache invalidation / update
- Augment business/process logic
- Update websites
- Integrate with event-based architectures
- Data streams

# Security Included

## In the Database

- Requires EXECUTE ANY EXTERNAL ENDPOINT database permission.

## Endpoint Authentication

- Header
- Query String/URL
- Managed Identity

# Easy to use

One simple and easy call to a REST endpoint directly from the Database

```
EXEC @returnValue = sp_invoke_external_rest_endpoint
[ @url = ] N'url'
[ , [ @payload = ] N'request_payload' ]
[ , [ @headers = ] N'http_headers_as_json_array' ]
[ , [ @method = ] 'GET' | 'POST' | 'PUT' | 'PATCH' | 'DELETE' | 'HEAD' ]
[ , [ @timeout = ] seconds ]
[ , [ @credential = ] credential ]
[ , @response OUTPUT ]
```

# Chapter 7: Invoke REST endpoints

- Invoke an Azure Function endpoint
- Invoke an OpenAI endpoint
- Add translation logic to the todo application with OpenAI

# Invoke an Azure Function endpoint

- Use a function to augment/enhance data
- Get the conversion rate for currency X and see what the price is
- Call endpoint and extract rate from JSON into a sql variable

```
set @priceConversion = (select JSON_VALUE(@response,'$.result.priceConversion') AS priceConversion);
```

```
select product_id, product_name, ListPrice,  
cast(round(ListPrice*@priceConversion,2,1) as money) AS convertedPriceInYen from dbo.products
```

# **START CHAPTER 7**



# CHAPTER 8: AI LANGUAGE



# Azure AI Language

Azure AI Language is a cloud-based service that provides Natural Language Processing (NLP) features for understanding and analyzing text.

Use this service to help build intelligent applications using the web-based Language Studio, REST APIs, and client libraries.

# AI Language Use Cases

What do you want to do?	Azure Language AI Solution
Detect and/or redact sensitive information such as PII and PHI.	PII detection
Extract categories of information without creating a custom model.	The preconfigured NER feature
Extract main topics and important phrases.	Key phrase extraction
Determine the sentiment and opinions expressed in text.	Sentiment analysis and opinion mining
Summarize long chunks of text or conversations.	Summarization
Disambiguate entities and get links to Wikipedia.	Entity linking

# AI Language Use Cases

What do you want to do?	Azure Language AI Solution
Extract medical information from clinical/medical documents, without building a model.	Text analytics for health
Build a conversational application that responds to user inputs.	Question answering
Detect the language that a text was written in.	Language detection

# Chapter 8: AI Language

- PII and Redaction
- Answer Questions
- Document summarization
- Sentiment analysis
- Language detection
- Named Entity Recognition (NER)
- Entity Linking

# Use External REST Endpoint Invocation

- Using EREI, send requests to AI Language from within the database
- Endpoints can be used with local relational data for various use cases
- REST calls will be using the Query Editor Worksheet in Codespace



# **START CHAPTER 8**

# CHAPTER 9: CONTENT SAFETY



# Azure AI Content Safety

Azure AI Content Safety detects harmful user-generated and AI-generated content in applications and services.

Azure AI Content Safety includes text and image APIs that allow you to detect material that is harmful.

# Content Safety Use Cases

The following are a few scenarios in which a software developer or team would require a content moderation service:

- Online marketplaces that moderate product catalogs and other user-generated content.
- Gaming companies that moderate user-generated game artifacts and chat rooms.
- Social messaging platforms that moderate images and text added by their users.
- Enterprise media companies that implement centralized moderation for their content.
- K-12 education solution providers filtering out content that is inappropriate for students and educators.

# Chapter 9: Content Safety

- Moderate text content
- Prompt Shields
- Protected material detection

# Use External REST Endpoint Invocation

- Using EREI, send requests to AI Content Safety from within the database
- Endpoints can be used with local relational data for various use cases
- REST calls will be using the Query Editor Worksheet in Codespace

# START CHAPTER 9



# CHAPTER 10: USING AZURE OPENAI



# Azure OpenAI

Azure OpenAI Service provides REST API access to OpenAI's powerful language models including GPT-4o, GPT-4 Turbo with Vision, GPT-4, and Embeddings model series.

Users can access the service through REST APIs, Python SDK, or our web-based interface in the Azure OpenAI Studio.



# Azure OpenAI Models

Models	Description
GPT-4o (mini)& GPT-4 Turbo	The latest most capable Azure OpenAI models with multimodal versions, which can accept both text and images as input.
GPT-4	A set of models that improve on GPT-3.5 and can understand and generate natural language and code.
GPT-3.5	A set of models that improve on GPT-3 and can understand and generate natural language and code.
Embeddings	A set of models that can convert text into numerical vector form to facilitate text similarity.
DALL-E	A series of models that can generate original images from natural language.
Whisper	A series of models in preview that can transcribe and translate speech to text.
Text to speech (Preview)	A series of models in preview that can synthesize text to speech.

# Chapter 10: Azure OpenAI

- Create Embeddings
- Images with DALL-E 3
- OpenAI GPT-4o image description
- Using Azure OpenAI Service and Relational Data

# Create Embeddings

Using ERL, send text to the embeddings REST endpoint

```
@message = N'{"input": "Its me Hi Im the problem, its me At teatime Everybody  
agrees Ill stare directly at the sun but never in the mirror It must be  
exhausting always rooting for the anti-hero"}';
```

```
"result": {  
  "object": "list",  
  "data": [  
    {  
      "object": "embedding",  
      "index": 0,  
      "embedding": [  
        -0.0023090523,  
        -0.04028898,  
        -0.0047508087,  
        ...  
      ]  
    }  
  ]  
};
```

# Create an image with DALL-E

Using text from the database (adventure works product description), create a image by calling the DALL-E 3 REST endpoint.

```
@message = N'{ "prompt": "This bike is ridden by race winners. Developed with the Adventure Works Cycles professional race team, it has an extremely light heat-treated aluminum frame, and steering that allows precision control.", "size": "1024x1024", "n": 1, "quality": "hd", "style": "vivid" }';
```

# Invoke an OpenAI endpoint

Use AI to get generated content from data in the database

```
set @adcopy =
(select 'You are an experienced marketing expert named Don Chase
Katz. Generate 200 letters of ad copy to '
+ person_name +
' to convince them to love ' +
case when pet_preference = 'Cats' then 'Dogs'
      when pet_preference = 'Dogs' then 'Cats'
end
from person
where person_id = 1);
```

# Point SWA at the Azure SQL Database

Change the connection string in the  
**staticwebapp.database.config.json** file to  
point to the Azure SQL Database

```
{ staticwebapp.database.config.json 3, U X
swa-db-connections > { staticwebapp.database.config.json > { data-source
1  {
2    "$schema": "https://github.com/Azure/data-api-builder/releases/download/v0.8.52/dab.draft.schema.json",
3    "data-source": {
4      "database-type": "mssql",
5      "connection-string": "Server=freedbsqlserver.database.windows.net;Database=freeDB;User ID=sqladmin;Password=PASSWORD;TrustServerCertificate=true",
6      "options": {
7        "set-session-context": false
8      }
9    },

```

# Add translations with OpenAI

Alter the insert\_todo stored procedure to call OpenAI to translate the text of a ToDo before an insert is done

```
declare @payload nvarchar(max) =  
N'{"messages": [{"role": "system", "content": "Translate  
\'' +(@title)+ '\'" into german, only respond with the  
translation"}]}'
```

# NOTE

- You may run into some traffic issues using the DALL-E REST endpoint.
- If the request does not go through, send it again.

A close-up photograph of a black dog's head and shoulders against a solid pink background. The dog has a wrinkled forehead, large brown eyes, and a small white patch on its chest. It is looking slightly upwards and to the right.

**START CHAPTER 10**

# CHAPTER 11: AZURE SQL BINDINGS



Sup?

# Azure Functions Core Tools

- Develop Azure Functions locally using Core Tools
- Deploy your code project to Azure
  - Function, App Container,
- Work with app settings locally  
(local.settings.json)

# Change Detection & Change Stream

How can I leverage the change tracking features in Azure SQL Database (or SQL Server/MI/VM) to expose data changes via an outgoing change stream without the need for custom code and scheduler functions?

# Azure SQL bindings for Azure Functions

- Azure SQL
  - encompasses all Azure SQL products (DB, MI, VM)  
+ SQL Server with cloud connectivity
- Azure Functions
  - serverless runtime for standalone use or integrated with Azure Static Web Apps
- Popular Languages
  - C# (in proc and out of proc), JavaScript, Python, Java, and PowerShell

# Three Azure SQL bindings for Azure Functions

Input binding



object in function



SQL query results



Azure SQL Input bindings take a SQL query or stored procedure to run and returns the output to the function.

---

Output binding



object in function



SQL table



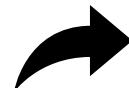
Azure SQL Output bindings take a list of rows and upserts them to the user table.

---

SQL trigger



SQL data change



starts function



The Azure SQL trigger uses SQL change tracking functionality to monitor a SQL table for changes

# Azure SQL bindings for Azure Functions

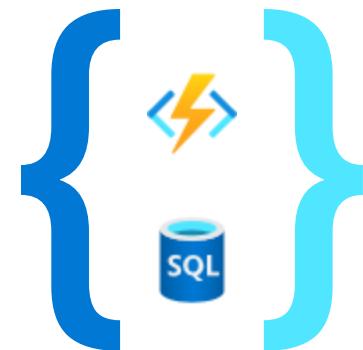
## Use Cases

- **Data enrichment**

- Determine if a value is an outlier or not using Azure Cognitive services/OpenAI
- Perform reverse geocoding using Azure Functions
- Call a REST/GraphQL service from a function with data from Azure SQL

- **Start complex processing**

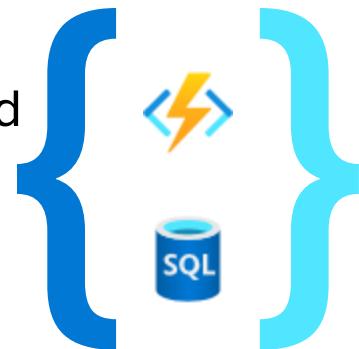
- Call a Function to kick off some complex process and return the data



# Azure SQL bindings for Azure Functions

## Use Cases

- **Update websites**
  - Broadcast a SignalR message
- **Integrate with event-based architectures**
  - Send data to Event Hubs for further integration options
- **Create a change data stream**
  - Send data to Stream Analytics for further investigation/fraud detection



# SQL bindings + Azure SQL compatibility

- The bindings use the [OPENJSON](#) statement which requires a **database compatibility level of 130 or higher (2016 or higher)**.
- Databases on **SQL Server, Azure SQL Database, or Azure SQL Managed Instance** which meet the compatibility level requirement above are supported.

# Connecting to the database

- SQL bindings connect to the target database by using a **Connection String configured in the app settings**. This will require a login be created that the function will use to access the server.
- For local testing and development using a SQL (username/password) or Azure Active Directory Login is typically the easiest, but for deployed function apps it is **recommended to use Azure Active Directory Managed Authentication**.

# SQL trigger app settings

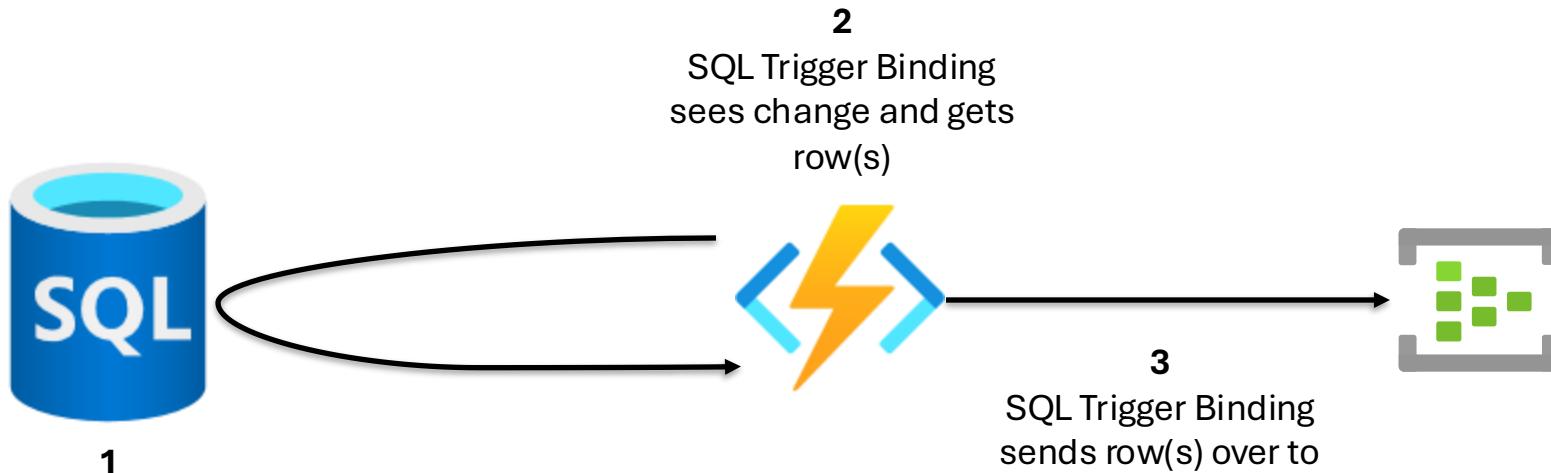
- **Sql\_Trigger\_BatchSize**: controls the number of changes processed at once before being sent to the triggered function.
- **Sql\_Trigger\_PollingIntervalMs**: controls the delay in milliseconds between processing each batch of changes.
- **Sql\_Trigger\_MaxChangesPerWorker**: controls the upper limit on the number of pending changes in the user table that are allowed per application-worker. If the count of changes exceeds this limit, it may result in a scale out. (only applies for Azure Function Apps with runtime driven scaling enabled)

# Change stream scenario

Requirement is to capture changes from a table and put them onto an Azure Event Hub for down stream subscribers (microservices).

Also, I don't want to write a lot of code.

# Change stream architecture



```
1  
Update on Products table  
  UPDATE products  
  SET product_price = 3.99  
  WHERE product_id = 2;
```

# Chapter 7: Azure SQL Bindings

- Create a function project
- Add a SQL trigger binding
- Run the function and test on the Azure SQL Database

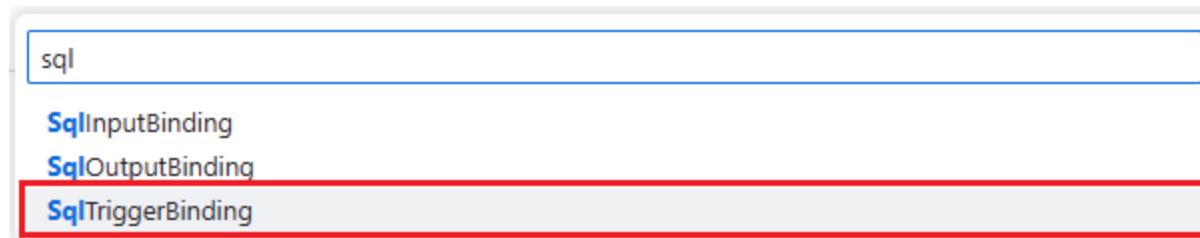
# Create a function project

- Create a .NET function using core tools

```
func init triggerBinding --worker-runtime dotnet
```

# Add a SQL trigger binding

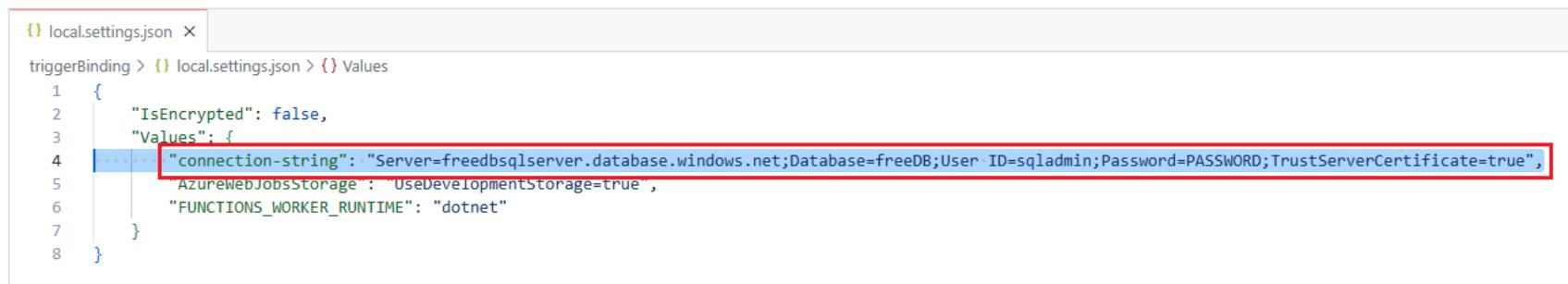
- Use the Create Function wizard to create the SQL Trigger Binding
  - can also be used to create input and output bindings



# Run and Test the Function

- Use core tools to run locally
- Connect to a local or cloud SQL Database

```
func host start
```



```
1  {
2      "IsEncrypted": false,
3      "Values": {
4          "connection-string": "Server=freedbsqlserver.database.windows.net;Database=freeDB;User ID=sqldadmin;Password=PASSWORD;TrustServerCertificate=true",
5          "AzureWebJobsStorage": "UseDevelopmentStorage=true",
6          "FUNCTIONS_WORKER_RUNTIME": "dotnet"
7      }
8  }
```

A large flock of Barn Swallows is perched on several horizontal wires against a clear blue sky. The birds are arranged in four distinct rows, with the highest row having the most birds. They are facing towards the left of the frame. The background is a uniform light blue.

**START CHAPTER 11**

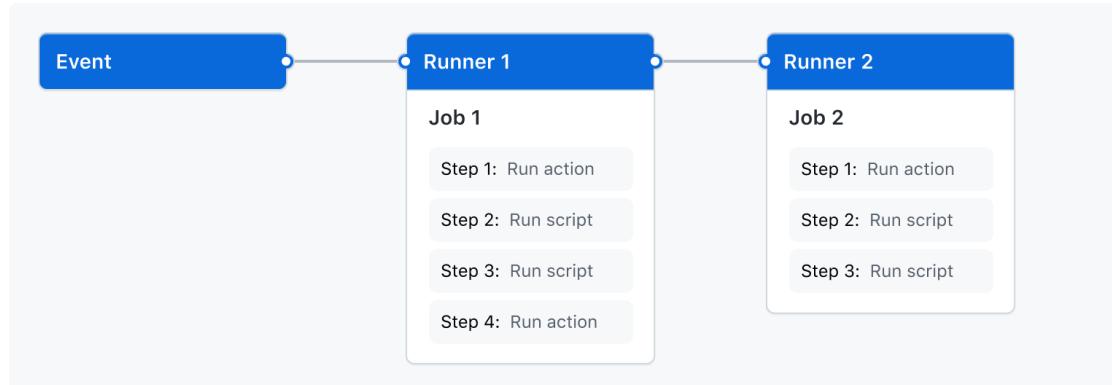


# CHAPTER 12:

# CI/CD WITH GITHUB ACTIONS

# GitHub Actions

- Are a continuous integration and continuous delivery (CI/CD) platform
- Allows you to run workflows based on repository events
- Test or deploy code with workflows

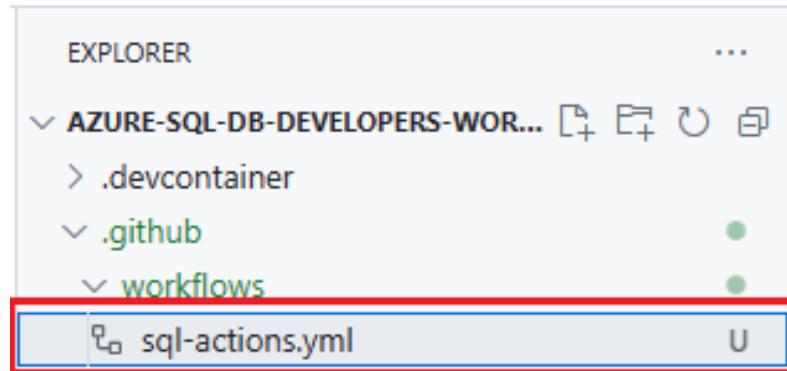


# Chapter 8: CI/CD with GitHub Actions

- Create a workflow folder and actions file
- Use sql-actions for DB creation and testing
- Push changes to start the action

# Create a workflow folder

- GitHub actions are in the .github/workflows folder
- Can contain multiple action workflows



# Use sql-actions

- Sql-actions are SQL specific activities that can be run in a workflow
- Publish (.dacpac) and run scripts on database
- Integrated with GitHub Secrets

# Push to Start

- Push code back to your fork to start the workflow
- Workflow will:
  - Create a SQL 2022 DB in docker
  - Create a test DB and set server parameters
  - Deploy the database project to the test database
  - Log all steps for review
  - Automatically deletes environment when done

## sql-actions.yml

```
# .github/workflows/sql-actions.yml  
  
name: SQL Server container in deployment  
test pipeline  
on: [push]
```

Name of the workflow

Runs on a git push

# sql-actions.yml

```
jobs:  
  build-and-deploy:  
    # Containers must run in Linux based operating  
    systems  
    runs-on: ubuntu-latest  
  
    # service/sidecar container for azure-sql-2022  
    services:  
      mssql:  
        image: mcr.microsoft.com/mssql/server:2022-  
latest  
        env:  
          ACCEPT_EULA: 1  
          SA_PASSWORD: XXXXXXXX  
        ports:  
          - 1433:1433
```

Groups together all the jobs that run in the workflow.

OS the container is going to run in

A docker container that will run SQL 2022

# sql-actions.yml

```
steps:
  - name: 'Checkout GitHub Action'
    uses: actions/checkout@v4

  - name: 'wait for sql to be ready'
    run: |
      set +o pipefail +e
      for i in {1..60};
      do
        sqlcmd -S localhost -U sa -P XXXXXXXX -d master -Q "select getdate()"
        if [ $? -eq 0 ]
        then
          echo "sql server ready"
          break
        else
          echo "not ready yet..."
          sleep 1
        fi
      done
      set -o pipefail -e
```

Check out the repository code

Wait for the database to be up and running

## sql-actions.yml

```
- name: 'Create and setup database'
  uses: azure/sql-action@v2
  with:
    connection-string: "Server=localhost;Initial Catalog=master;User ID=sa;Password=XXXXXX;Encrypt=False;TrustServerCertificate=False;" # the local connection string
    path: './labFiles/setupDatabase.sql' # the sql script to create db and configure for clr

- name: 'Deploy Database Project'
  uses: azure/sql-action@v2.2
  with:
    connection-string: "Server=localhost;Initial Catalog=testingDB;User ID=sa;Password=XXXXXX;Encrypt=False;TrustServerCertificate=False;" # the local connection string
    path: './database/devDB/devDB.sqlproj' # the SQLproj file
    action: 'Publish'
```

Once the DB is ready, run the following script.  
(creates testDB and run env config options)

Deploy the database project we created to the testDB

# **START CHAPTER 12**



# THANK YOU!



# Session Survey

- Your feedback is very important to us
- Please take a moment to complete the session survey found in the mobile app
- Use the QR code or search for “Converge360 Events” in your app store
- Find this session on the Agenda tab
- Click “Session Evaluation”
- Thank you!

