Visual Studio LIVE! EXPERT SOLUTIONS FOR ENTERPRISE DEVELOPERS | Las Vegas

Workshop: Full Stack Development

Brian Spendolini (bspendolini@microsoft.com)
Drew Skwiers-Koballa (drskwier@microsoft.com)
Microsoft

Level: Introductory to Intermediate

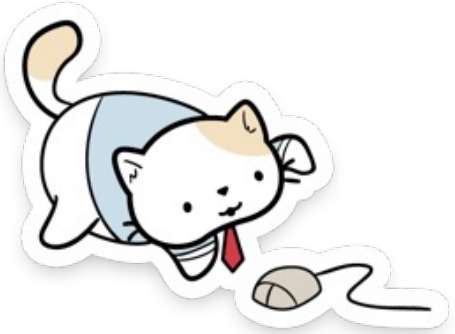
Session Survey

- 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 Goals

- Learn the foundations of Fullstack/Jamstack
  - Architecture and technologies
- Learn how to use Azure to build a Fullstack/Jamstack solution
  - And collaborate in building one
  - Get a solid understanding of all the moving pieces
- Understand alternatives
  - Discuss about alternatives
- Feel empowered and happy!

Warning!



- Focus on the slide and the content
- DO NOT try to replicate right now what I'll be showing on during the workshop
  - You will probably fall behind
  - The goal is to learn; the slide deck and the GitHub repo will help you to do it on your own later

There’s a lot to do today

- Fullstack and Jamstack Architecture
- Tools for the job
- A database discussion
- Cross-Cutting concerns
- Deployment and CI/CD
- Advanced Topics
- Alternative Solutions and Implementations

There’s a lot to do today

- Fullstack and Jamstack Architecture
- Tools for the job
  - Static Web Apps, Azure Functions, Vue.Js, Azure SQL Database
- A database discussion
- Cross-Cutting concerns
- Deployment and CI/CD
- Advanced Topics
- Alternative Solutions and Implementations

There’s a lot to do today

- Fullstack and Jamstack Architecture
- Tools for the job
- A database discussion
  - Why Azure SQL?
  - Stored Procs or not? ORM? Micro-ORM?
  - JSON? What if I have “non-relational” data?
- Cross-Cutting concerns
- Deployment and CI/CD
- Advanced Topics
- Alternative Solutions and Implementations

There’s a lot to do today

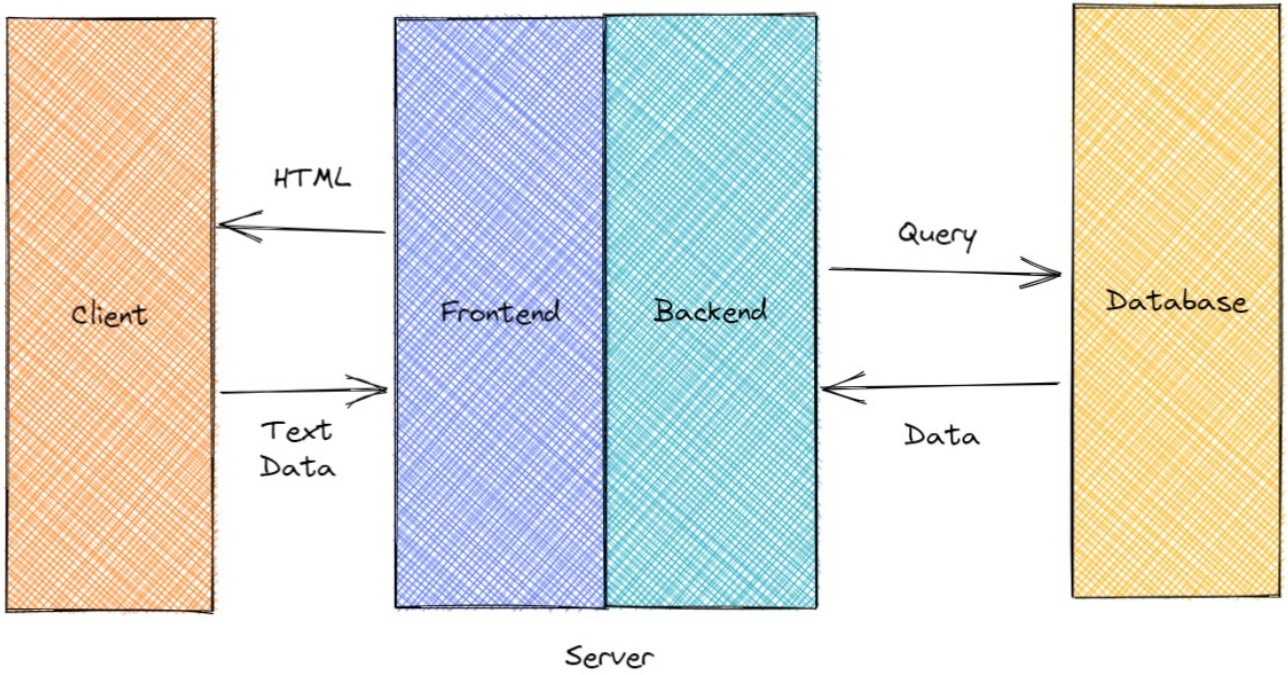
- Fullstack and Jamstack Architecture
- Tools for the job
- A database discussion
- Cross-Cutting concerns
  - Authentication (no passwords please!)
  - Connection Resiliency
- Deployment and CI/CD
- Advanced Topics
- Alternative Solutions and Implementations

There’s a lot to do today

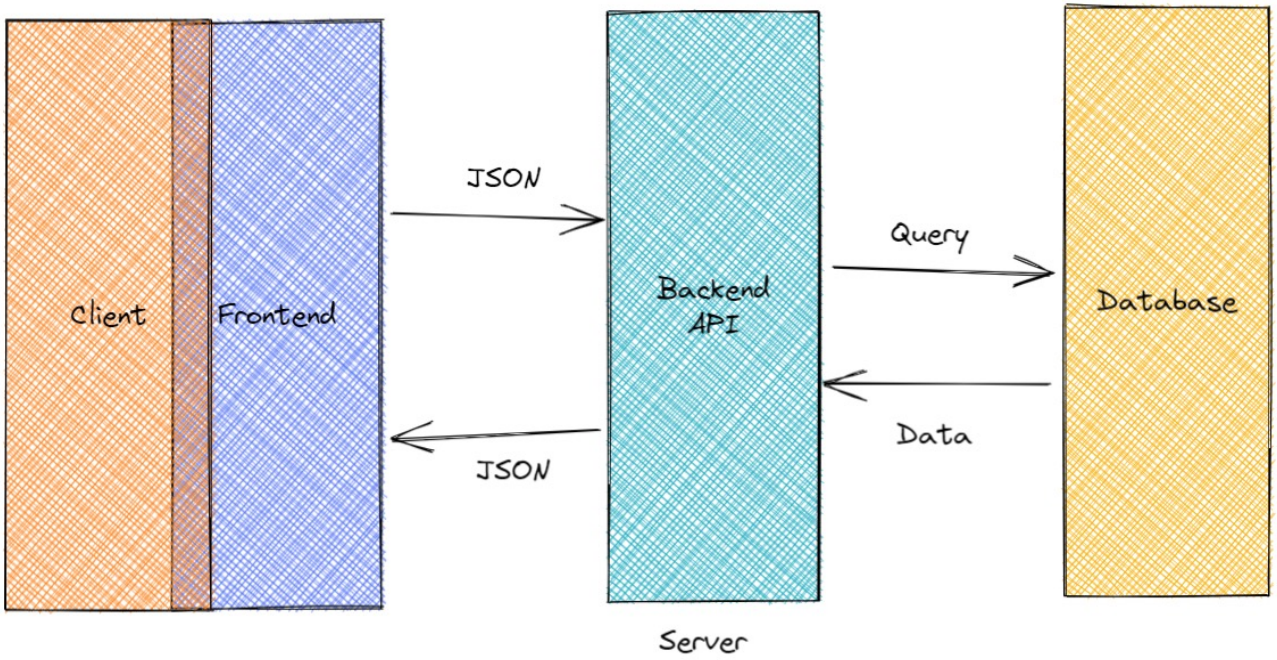
- Fullstack and Jamstack Architecture
- Tools for the job
- A database discussion
- Cross-Cutting concerns
- Deployment and CI/CD
- Advanced Topics
  - Caching
  - Scaling Out
- Alternative Solutions and Implementations

LET'S GET STARTED!

Fullstack

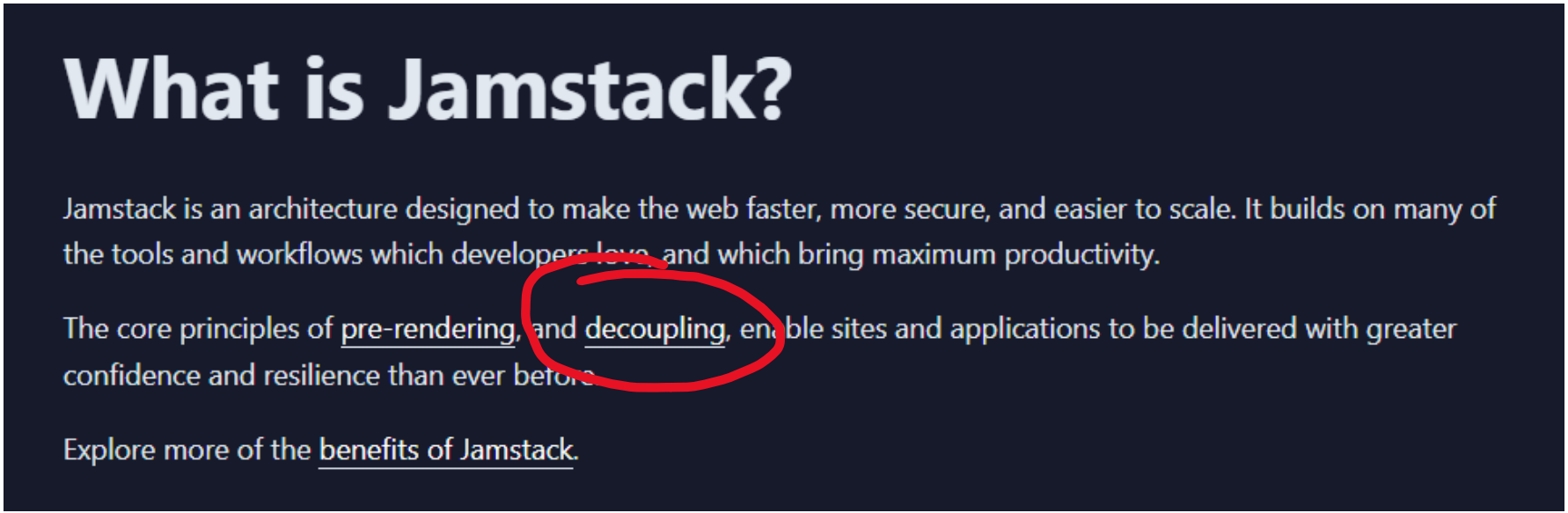


(Fullstack) Jamstack



Jamstack Fullstack

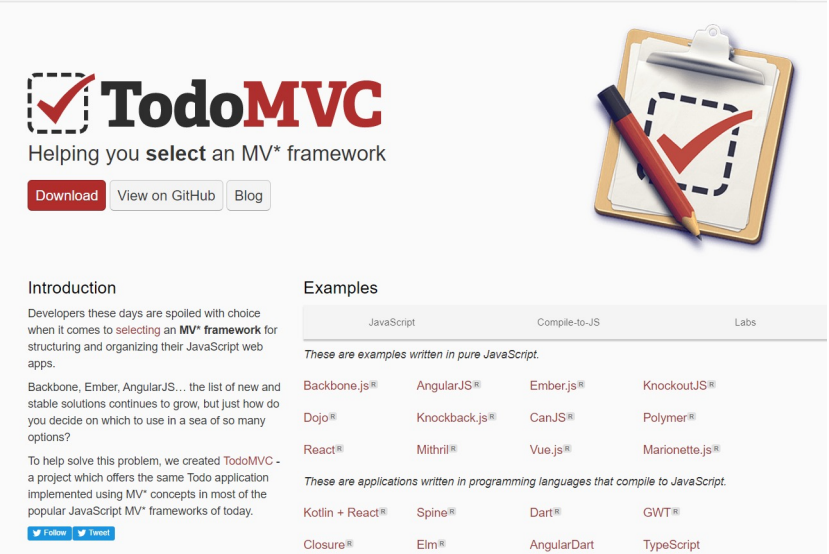
Modern Fullstack architecture:

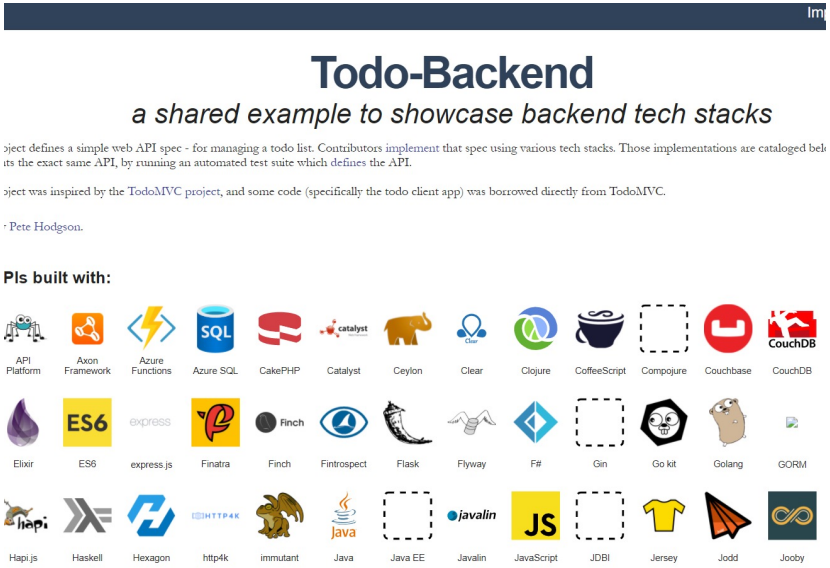


Technologies

- Frontend
  - HTML / CSS
  - JavaScript / TypeScript
- Backend
  - JavaScript / TypeScript, Python, C#, Java…you name it
- Database
  - Azure SQL

The sample application





https://todomvc.com/

https://www.todobackend.com/

30th ANNIVERSARY VSLive! 1993 – 2023

Jamstack on Azure

- Many options
  - For each layer, several different options
- Best way to start:
  - Azure Static Web Apps
  - Azure Functions
  - Azure SQL Database
  - Git & GitHub
- All the mentioned services provide a full offline development experience
- You can also develop completely online – no need to install anything on your machine
  - thanks to GitHub Codespaces or VS Code on the web

Technologies needed

- .NET SDK 6.x+
- Node 16+
- Visual Studio Code
  - Azure Function Extensions
- Recommended:
  - Azure Data Studio
  - Windows Subsystem for Linux (WSL) + Ubuntu 20
  - AZ CLI
- All the mentioned tools and languages are free to use

Git & GitHub

- Git: Source Code Control tool
  - To be installed on your machine

https://git-scm.com/doc

- GitHub: a hosted service where you can send your code to be safely stored and accessed
  - Compatible with Git protocol
  - It also provides additional services like “Push Requests”, Continuous Integration / Continuous Delivery, DevOps, Project Management, etc.
  - We’ll be using GitHub Actions

https://docs.github.com/en/get-started/quickstart

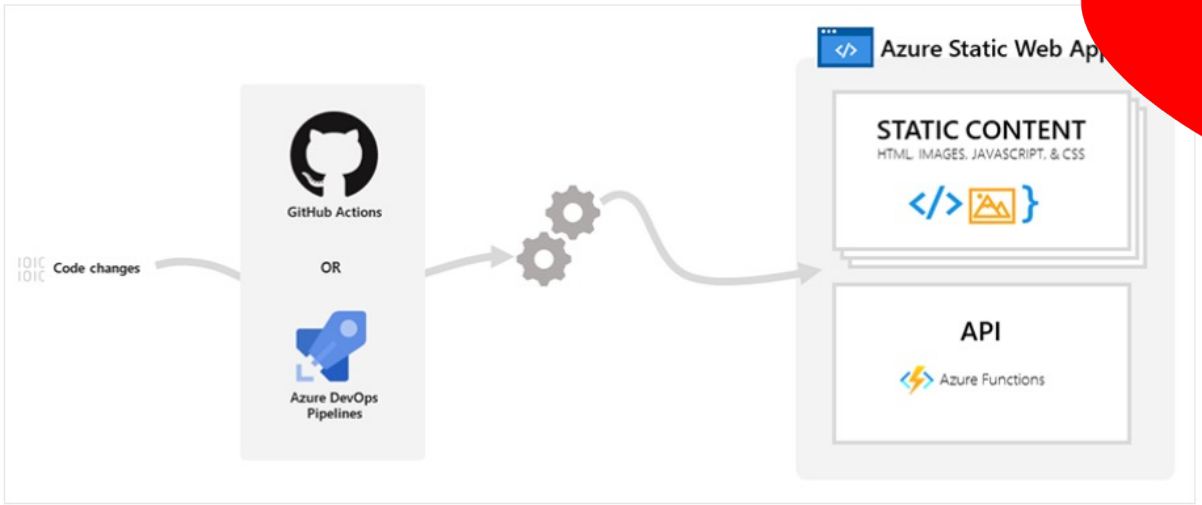
Azure Static Web Apps

What is Azure Static Web Apps?

Article • 10/17/2022 • 2 minutes to read • 9 contributors

Feedback

Azure Static Web Apps is a service that automatically builds and deploys full stack web apps to Azure from a code repository.



Azure Functions

- Serverless solution
  - Just write the function code and deploy it
  - Use your favorite or the best language for the job
  - Support for C#, Python, Java, JavaScript, Go, Rust, etc.
- A function is executed based on events or, better, triggers.
- First 1 Million Request per month is *free* with a Consumption Plan

Meter	Free Grant (Per Month)
Execution Time*	400,000 GB-s
Total Executions*	1 million executions

Azure Static Web Apps CLI

Static Web Apps CLI

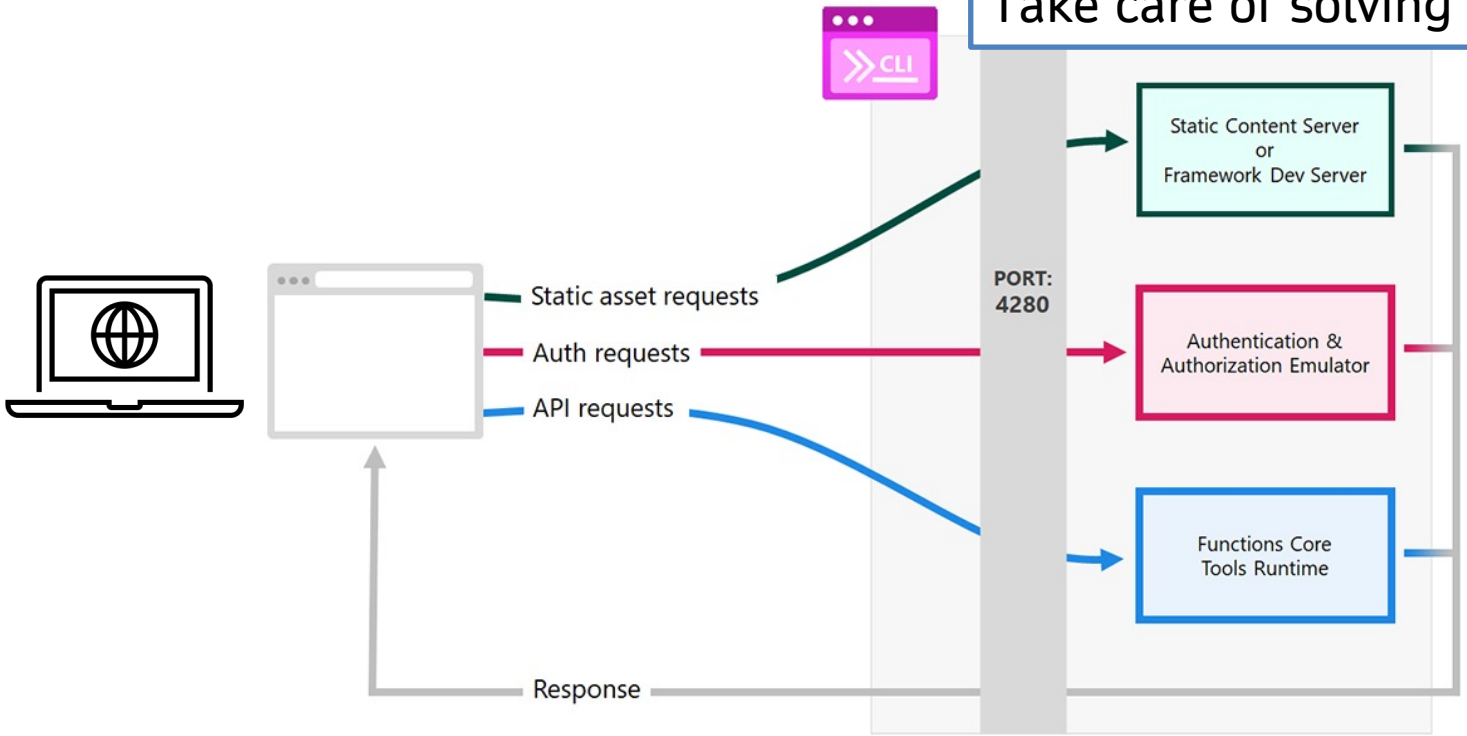
All-in-One Local Development Tool For Azure Static Web Apps

Get Started →

https://azure.github.io/static-web-apps-cli/

Azure Static Web Apps CLI

Take care of solving CORS issues



https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS

Let’s get started!

- Make sure you have Node installed
  - Node also comes with Node Package Manager (NPM)
  - https://nodejs.org/en/

- Install Azure Static Web Apps CLI

```
npm install -g @azure/static-web-apps-cli
```

Azure Functions Core Tools

- Install Azure Function CLI and host

```
npm i -g azure-functions-core-tools --unsafe-perm true
```

- Create our first function

```
func init --worker-runtime dotnet
func new --template HttpTrigger --name SampleAPI
```

Run our first Jamstack app

- Create a super simple HTML page
  - https://aka.ms/jamstack-workshop-html-sample

- Run SWA

```
swa init
swa build
swa start
```

Start to build the Todo app

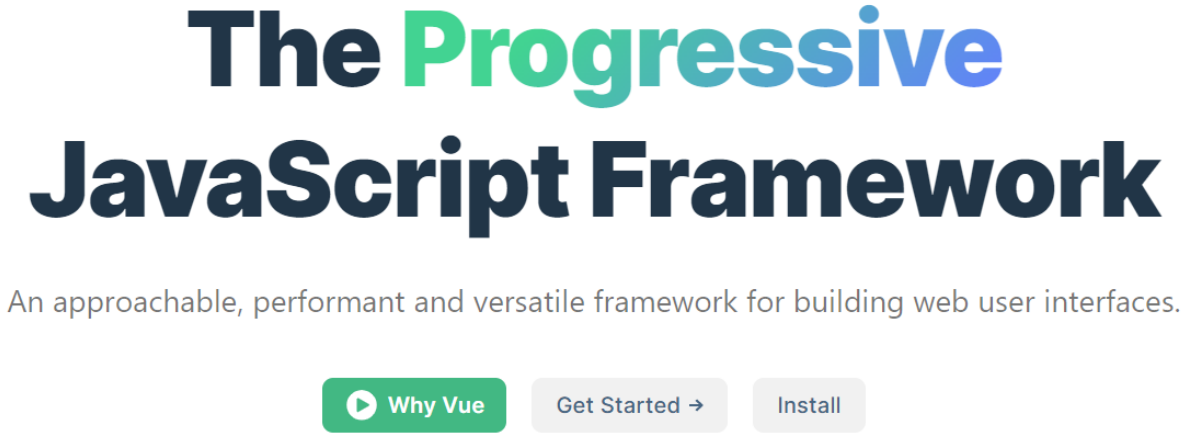
Start to build the Todo app

- Full repository here:
  - https://github.com/azure-samples/azure-sql-db-fullstack-serverless-kickstart

- Several branches
  - Always a work in progress
  - ToDo application is fully working at 3.0
    - Improvements are done in later versions
  - Fork the repo into your own account

One last piece

- How to build the frontend? So many frameworks available
  - We'll use Vue.JS
  - Is one of the simplest yet widely used and powerful



Vue(.js)

- Why Vue?
- From the website:
  - Enhancing static HTML without a build step
  - Embedding as Web Components on any page
  - Single-Page Application (SPA)
  - Fullstack / Server-Side Rendering (SSR)
  - Jamstack / Static Site Generation (SSG)
  - Targeting desktop, mobile, WebGL, and even the terminal
- https://aka.ms/jamstack-workshop-vue-sample

Fork the repo

Create a new fork

A *fork* is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project. View existing forks.

Single sign-on to see more options for organizations within the **Microsoft Open Source** enterprise.

Owner * **Repository name** *

yorek / azure-sql-db-fullstack-serverless-

By default, forks are named the same as their upstream repository. You can customize the name to distinguish it further.

Description (optional)

Fullstack/Jamstack solution with Vue.js, Azure Functions, Azure Static Web apps and Azure SQL.

☐ **Copy the** `main` **branch only**

Contribute back to Azure-Samples/azure-sql-db-fullstack-serverless-kickstart by adding your own branch. Learn more.

You are creating a fork in your personal account.

1.0

DEMO

Azure SQL Database

We want to be ready for anything, so we need:

- Flexibility
- Security
- Performance
- Consistency

Also: a great development experience

Azure SQL is a great **general-purpose**, **battery-included** post-relational database

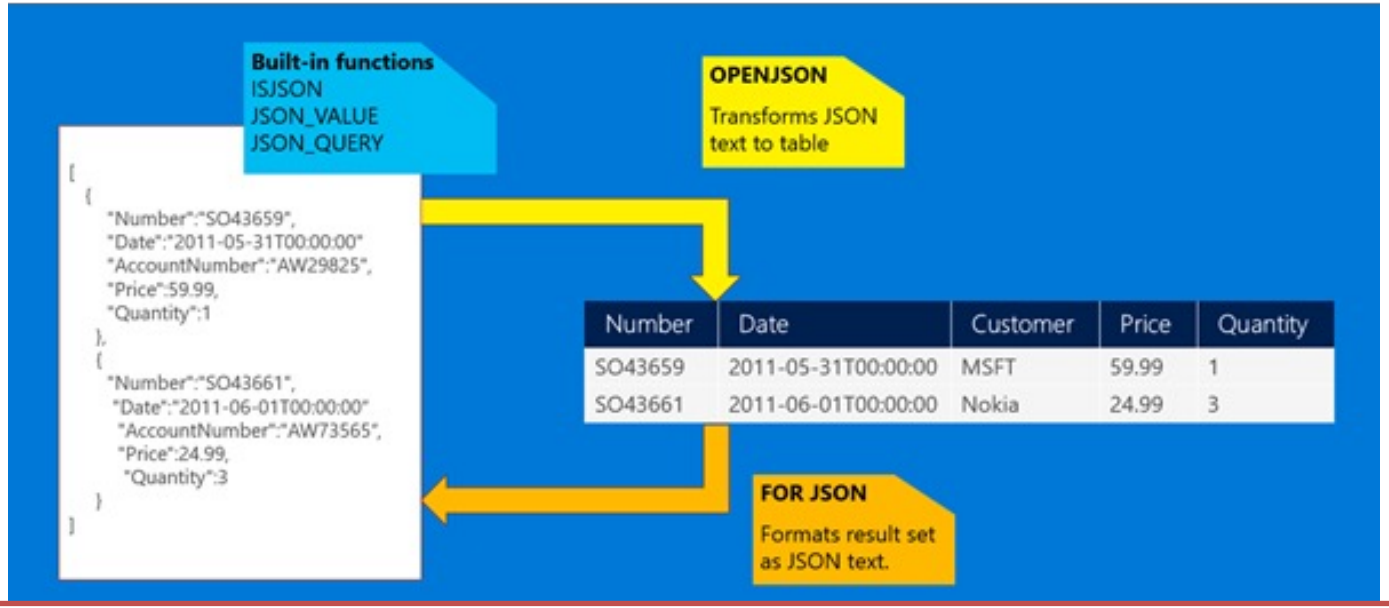
- (cit. from pep-0206)

Azure SQL Database

- Features you want to know/use:
  - Pessimistic (locks) and Optimistic (row-version) concurrency
  - Multi-model support
    - Relational / JSON / Graph / Geospatial / XML
  - In-Memory Lock Free Tables
  - Row-Store and Column-Store
    - on the same table
  - Encryption & Ledger Tables
  - Row Level Security
  - Replicas / High-Availability / Scale-Out / Scale-Up

JSON Support

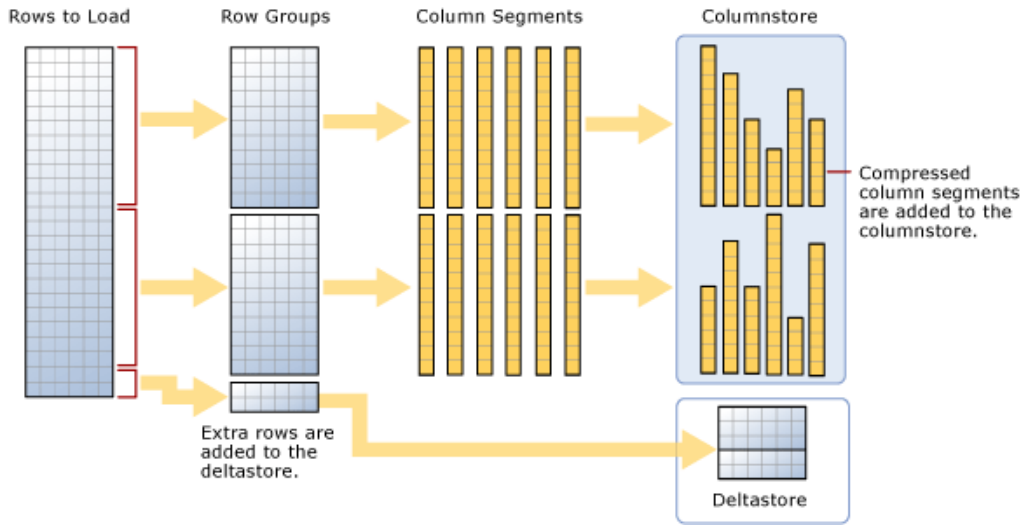
- Azure SQL fully support storage, manipulation and creation of JSON documents



https://docs.microsoft.com/en-us/sql/relational-databases/json/json-data-sql-server

Columnstore

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



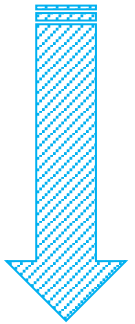
https://docs.microsoft.com/en-us/sql/relational-databases/indexes/columnstore-indexes-overview

Columnstore

TPC-H 10GB, BC_Gen5_2, Query 5

```
1  SELECT
2      n_name,
3      sum(l_extendedprice * (1 - l_discount)) as revenue
4  FROM
5      customer,
6      orders,
7      lineitem,
8      supplier,
9      nation,
10     region
11 WHERE
12     c_custkey = o_custkey
13     AND l_orderkey = o_orderkey
14     AND l_suppkey = s_suppkey
15     AND c_nationkey = s_nationkey
16     AND s_nationkey = n_nationkey
17     AND n_regionkey = r_regionkey
18     AND r_name = 'ASIA'
19     AND o_orderdate >= '1994-01-01'
20     AND o_orderdate < dateadd(year, 1, '1994-01-01')
21 GROUP BY
22     n_name
23 ORDER BY
24     revenue desc;
```

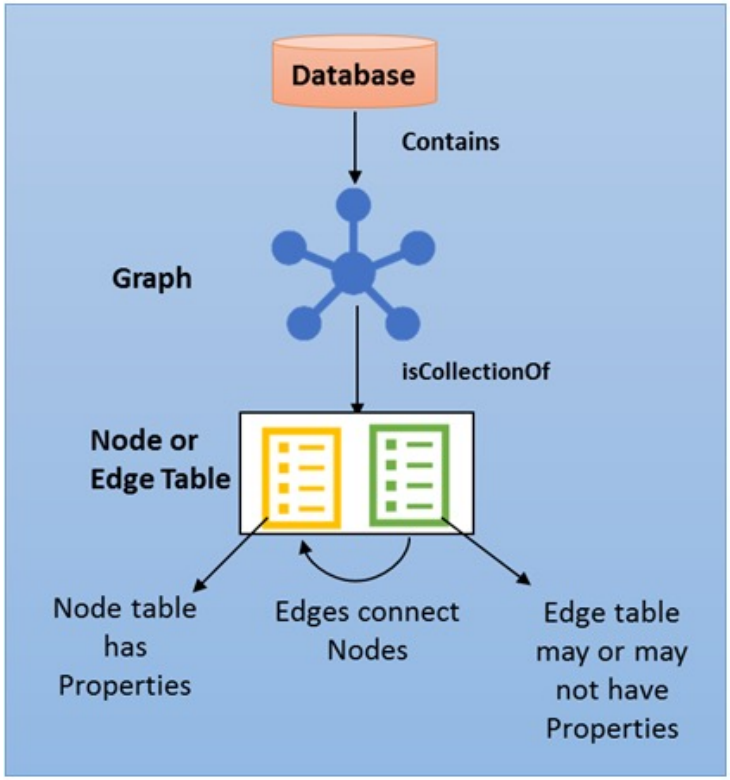
55 seconds



4 seconds

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

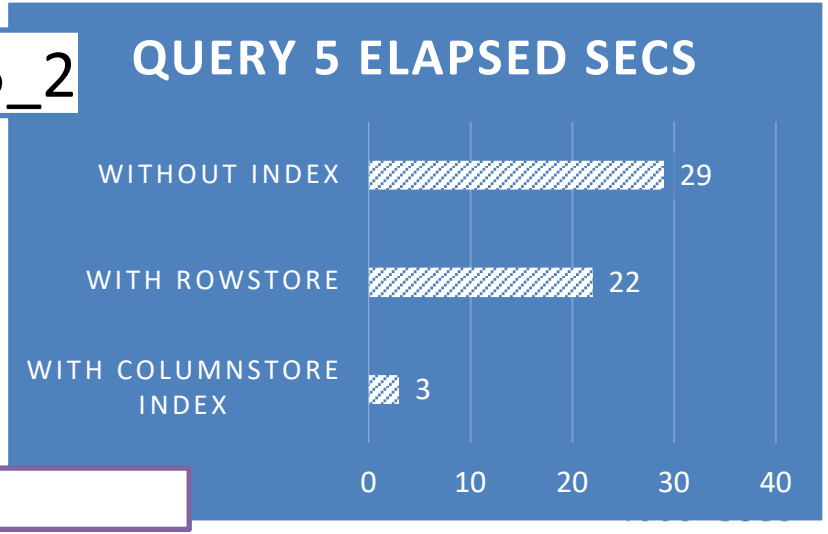


Why Graph in Azure SQL

- Thanks to the Azure SQL engine, Graph support can be used along with existing technologies

- Columnstore + Graph allow the creation of amazing solutions!
- ~ 1 Mil. Unique Songs
- ~ 1 Mil. Unique Users
- ~ 50 Mil. Likes

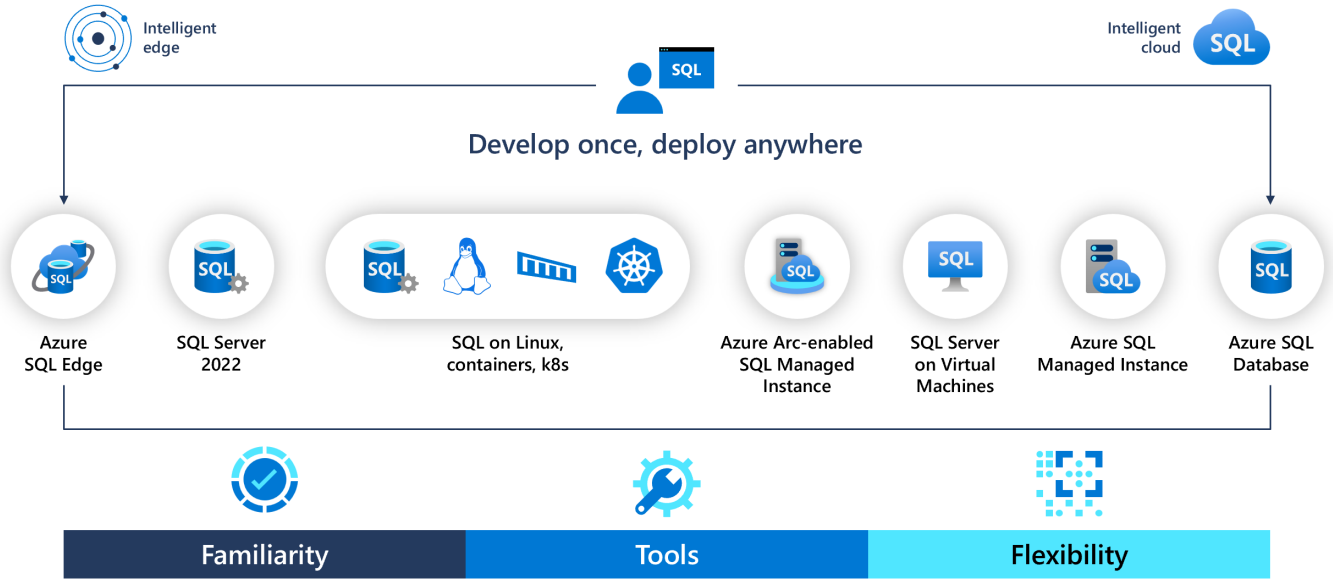
HS_Gen5_2



https://github.com/Azure-Samples/millionsongdataset-sql

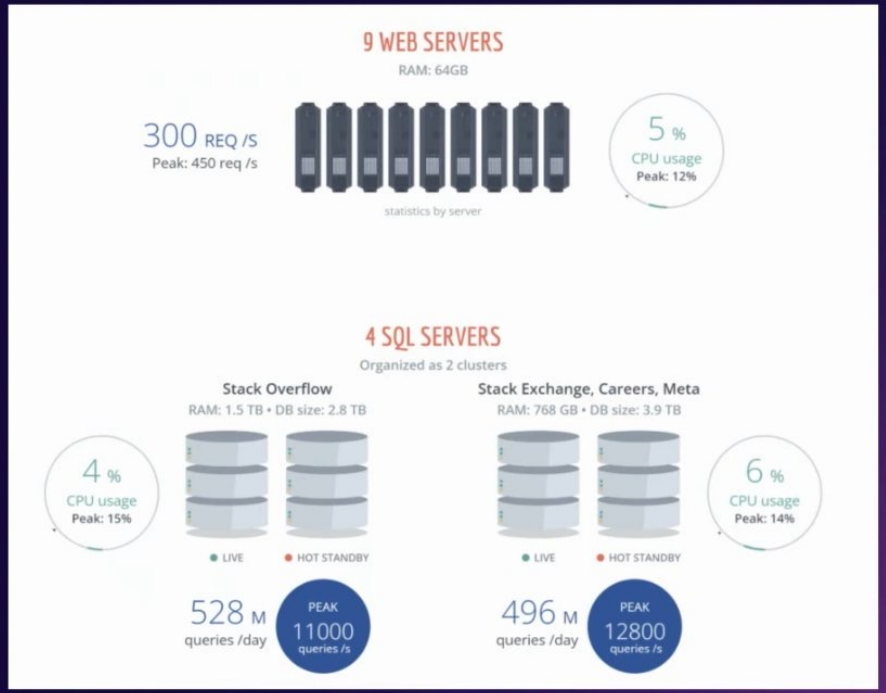
Azure SQL Database

- Has the same engine that SQL Server has
  - Works everywhere from Edge to Cloud
  - The same engine everywhere: On-Prem, IaaS, PaaS



SQL Server engine @ Work

Stack Overflow



.NET

30th ANNIVERSARY VSLive! 1993 – 2023

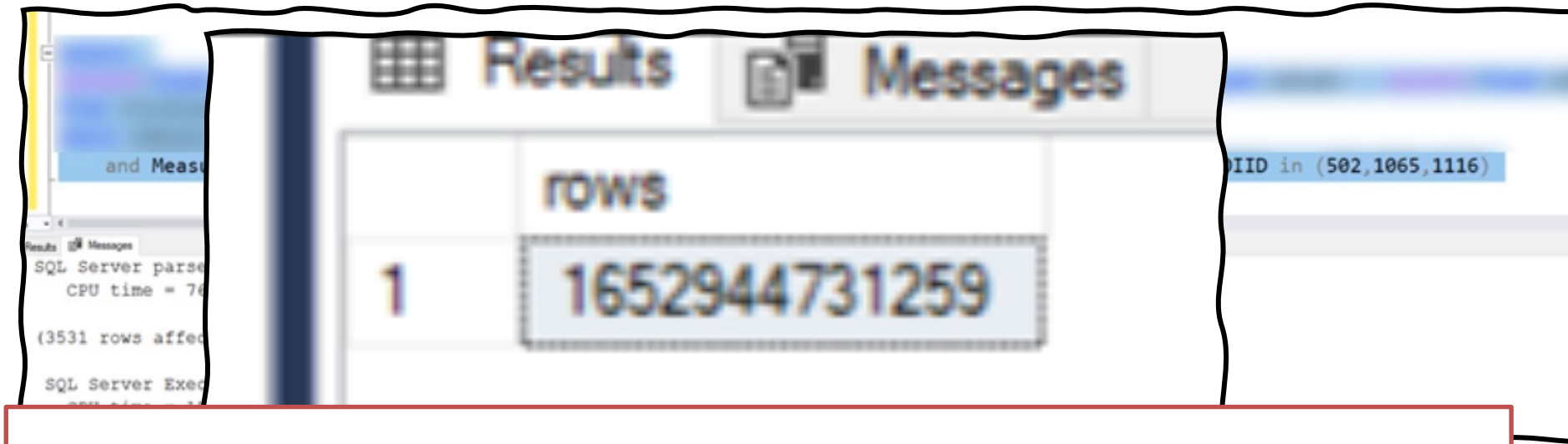
SQL Server engine @ Work

Stack Overflow

Some numbers from October 24th

- **226,451,693** requests to our load balancers
- **85,078,133** of those were page views
- **4,265,481,130,190** bytes sent (4.2 TB)
- **729,214,239 SQL** queries (from HTTP requests alone)
- **297,814,074** Redis hits
- **24ms** average for **66,711,536** question page renders

Azure SQL @ Work



1.6 Trillion Rows

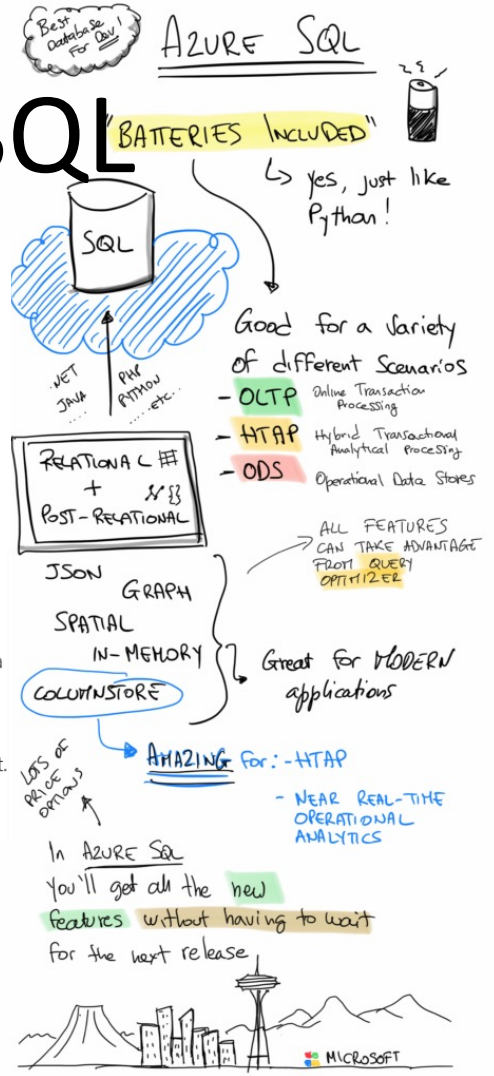
10 reasons for Azure SQL

10 Reasons why Azure SQL is the Best Database for Developers

By Davide Mauri

Published Oct 31 2019 10:30 AM 17K Views

Azure SQL is the relational and post-relational database that I'd like to say has "batteries included". I borrowed that term from Python, one of my preferred languages, and that's also the main reason why it is a great database for developers. A lot of features that you would normally find in different, more specialized, products are nicely integrated into just one. Using different specialized product can bring great advantages but also poses a huge challenge: data must be moved around all those products and the burden of keeping consistency is on developer's shoulders. And it is a big one, one that will increase application complexity by an order or magnitude. And as a developer I just prefer to follow the KISS principle. Azure SQL allows me to focus only on what it's unique to my company and application, delegating all the responsibilities related to data to something specialized on that. That's a big win for everyone.



https://aka.ms/azuresqldbdev10

Using Azure SQL Database

- Four options
  - ORM
  - Micro-ORM
  - Query Builders
  - Direct
  - Direct*

ORM

- Object Relational Mapping tools
  - For .NET: Entity Framework (EF Core)
    - Just released version 7.0, super fast and super rich!
  - Python: SQLAlchemy
  - Typescript: Prisma
  - Javascript: Sequelize

- Abstract the database access almost completely
  - You define and manipulate objects (POCO = Plain Old Class Object)

ORM – EF Core

```
public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }
    public int Rating { get; set; }
    public List<Post> Posts { get; set; }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public int BlogId { get; set; }
    public Blog Blog { get; set; }
}
```

```
using (var db = new BloggingContext())
{
    var blogs = db.Blogs
        .Where(b => b.Rating > 3)
        .OrderBy(b => b.Url)
        .ToList();
}
```

```
using (var db = new BloggingContext())
{
    var blog = new Blog { Url = "http://sample.com" };
    db.Blogs.Add(blog);
    db.SaveChanges();
}
```

```
dotnet add package Microsoft.EntityFrameworkCore.SqlServer
```

ORM

- Good SQL/Database knowledge *recommended*, but not needed
  - For example: how to check the performance how underlying generated query
  - Also, usually possible to also send “raw” SQL
- (EF Core especially) provides a lot of out-of-the-box best-practices and optimizations
  - Make sure to learn your ORM of choice very well
- Provides a way to incrementally update/migrate the database

Micro ORM

- Do not provide the full set of features of an ORM, but removes the need for plumbing code
  - You still have full control (no database abstraction)
  - Mapping to POCO object is done automatically for you
  - Good SQL knowledge is *required*
- The most known is Dapper for .NET

```
dotnet add package Dapper
```

- Almost no additional knowledge required to use it correctly
  - 6 Articles series on Dapper

Micro ORM

```
public class User
{
  public int Id { get; set; }
  public string FirstName { get; set; }
  public string LastName { get; set; }
}
```

```
// conn is a SqlConnection
var queryResult = conn.Query<User>("SELECT [Id], [FirstName],[LastName] FROM dbo.[Users]");
```

```
int affectedRows = conn.Execute("UPDATE dbo.[Users] SET [FirstName] = 'John' WHERE [Id] = 3");
```

Query Builder

- Helps you to build a query using your backend programming language (eg: C#)
  - No need to define models upfront
  - Automatically maps results to your POCO objects
- .NET: SQLKata, Javascript: Knex
- Something in-between ORM and Micro-ORM
  - Provides some sort of database abstraction (you don't write SQL)
  - Still very lightweight (it "just" write SQL)

Query Builder

```
IEnumerable<Post> posts = await db.Query("Posts")
    .Where("Likes", ">", 10)
    .WhereIn("Lang", new [] {"en", "fr"})
    .WhereNotNull("AuthorId")
    .OrderByDesc("Date")
    .Select("Id", "Title")
    .GetAsync<Post>();
```

```
SELECT [Id], [Title] FROM [Posts] WHERE
  [Likes] > @p1 AND
  [Lang] IN ( @p2, @p3 ) AND
  [AuthorId] IS NOT NULL
ORDER BY [Date] DESC
```

```
var query = new Query("Books").AsInsert(new {
    Title = "Toyota Kata",
    CreatedAt = new DateTime(2009, 8, 4),
    Author = "Mike Rother"
});
```

```
INSERT INTO [Books] ([Title], [CreatedAt], [Author]) VALUES ('Toyota Kata', '2009-08-04 00:00:00',
'Mike Rother')
```

Direct

- Use native language libraries. For example, with .NET Microsoft.Data.SqlClient
  - System.Data.SqlClient is the *old* version
- You have total control over how queries are sent to the database and how the result is manipulated
  - SQL knowledge is *required*
- But you also have total responsibility to use it correctly
  - It might be a lot of work
  - Not the funniest things to do, after all is “plumbing” code.
  - Returned result is a result set (DataTable or Enumerable)

Direct

```
dotnet add package Microsoft.Data.SqlClient
```

```
using (SqlConnection conn = new SqlConnection(connectionString))
{
    conn.Open();

    var cmd = new SqlCommand("select id, todo, completed from dbo.todos", conn);
    var r = cmd.ExecuteReader();
    while(r.Read()) {
        Console.WriteLine("{0}\t{1}\t{2}",
            r.GetInt32(0),
            r.GetString(1),
            r.GetByte(2));
    }
}
```

Direct*

- Azure SQL bindings for Azure Functions

  T08 Further, Faster, with Azure Functions and Azure SQL Integrations
  03/21/2023 9:30am - 10:45am
  https://aka.ms/sqlbindings

- Data API builder

  W22 Data API Builder: Instant REST and GraphQL endpoints to your Azure Databases
  03/22/2023 4:00pm - 5:15pm
  https://aka.ms/dab

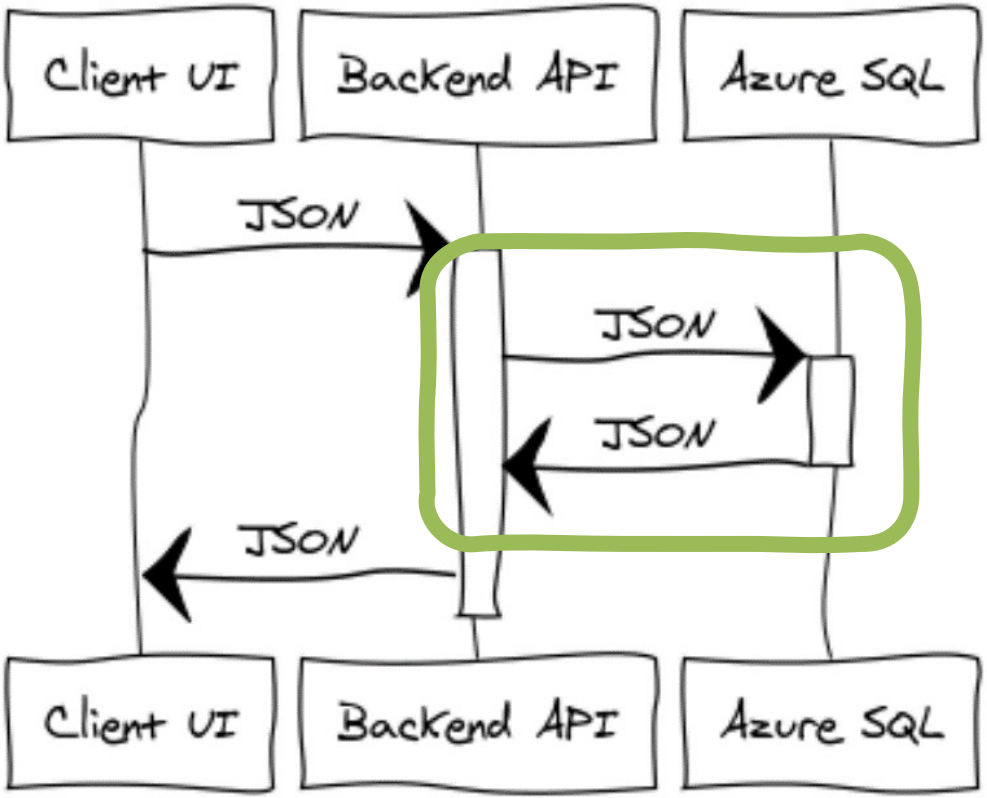
Using Azure SQL Database

- Four options
  - ORM
  - Micro-ORM
  - Query Builders
  - Direct
  - Direct*

Database/Code/Model First?

- "Database First"
  - Create the database and then you map it to your code Dapper
- "Code First"
  - You create the code and then generate/map the database EF Core
- "Model First"
  - You create an abstract model and then generate the code and the database Prisma

JSON and Databases



JSON and Databases

- Modern relational databases all supports JSON very well
- Aside from storing JSON data, JSON is an interesting transport protocol
  - Can be very easily serialized and deserialized in *any* language
  - No need to deal with columns and rows anymore
- JSON also helps to make a database more flexible
  - You decide where you want to enforce the schema validation (in app, in database or both)

Stored Procedures?

- Stored Procedures allows you to present a set of API for developers to use
  - Abstraction from the underlying database schema
- Even better separation between backend and database
  - Encapsulation of *data access and manipulation* logic
- Provide you total control over transaction management, error handling, etc, all on the database side
  - Generally, provide performance improvements
  - Less network usage/roundtrips, improved plan caching

Stored Procedures?

- Good SQL knowledge required

- Could be harder to move from one database system to another one
  - But it is easier to take advantage of unique database features, if those are not supported by your ORM, while keeping the backend code clean and lean

Secure database access

- Do not use the administrator account for your app
  - Too powerful, open doors to devastating injection attacks
- Create a dedicated user for your app, with the minimum permission needed
- Spend some time to become confident with the security model of the chosen database system

Secure database access

- Azure SQL
  - GRANT/DENY/REVOKE model on INSERT/UPDATE/DELETE/SELECT/EXECUTE
  - Database Roles and Database Schemas to make management easier
  - “Ownership chain” helps a lot to simplify management

- Connect to Azure SQL in 6 easy steps

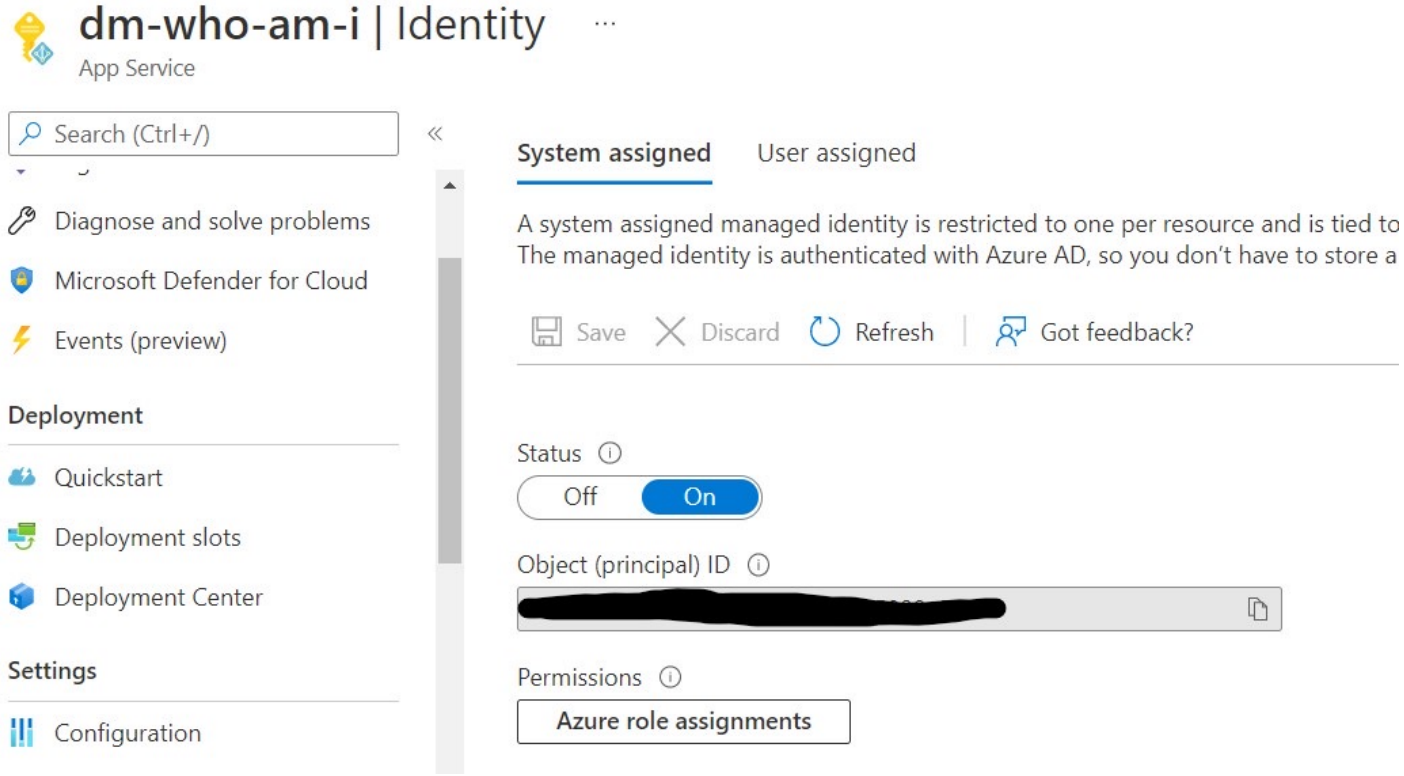
2.0

DEMO

No More Passwords

- To make your solution even more secure, try to use a passwordless connection string
- All you need is to use Managed Identities and the Azure.Identity library
- Then, use MSI – Managed Service Identity – to remove passwords from connection strings

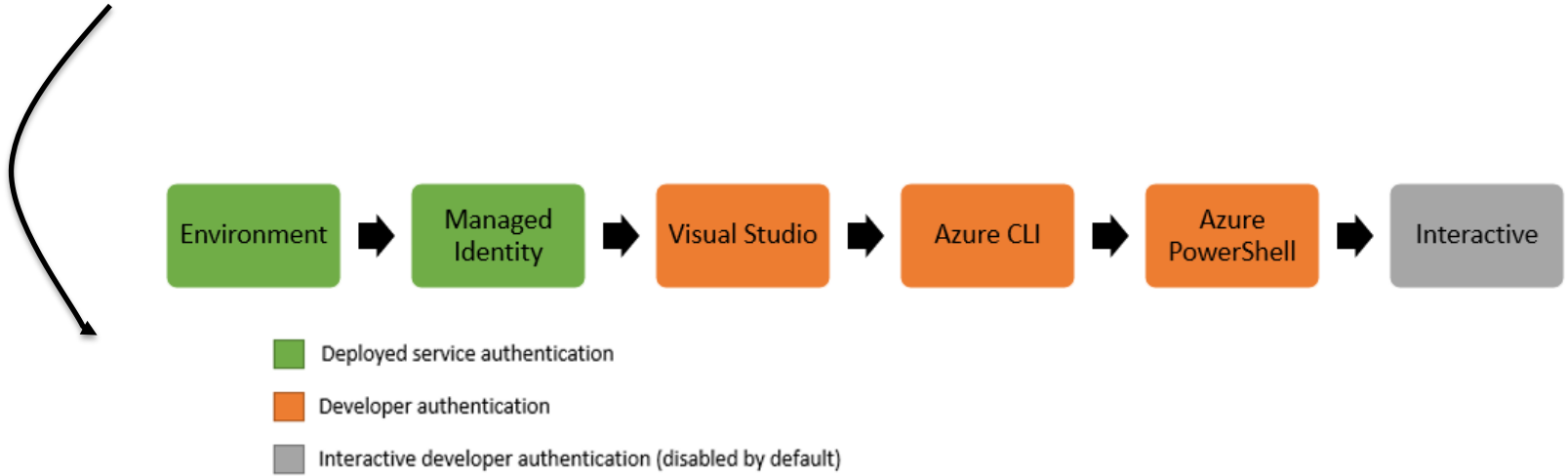
No More Passwords



https://github.com/Azure-Samples/azure-sql-db-who-am-i

Go Passwordless

```
1  var credential = new Azure.Identity.DefaultAzureCredential();
2  var token = await credential.GetTokenAsync(new Azure.Core.TokenRequestContext(new[] { "https://database.windows.net/.default" }));
3  var conn = new SqlConnection(_config.GetConnectionString("AzureSQL"))
4  conn.AccessToken = token;
5  await conn.OpenAsync();
6
```



https://docs.microsoft.com/en-us/dotnet/api/overview/azure/identity-readme

Ad-hoc

DEMO

https://learn.microsoft.com/azure/azure-functions/functions-identity-access-azure-sql-with-managed-identity

AuthN & AuthZ

- AutheNtication (who you are)
  - Very common to have an external authentication using well-known *Identity Providers*
  - For example: Azure AD, Twitter, GitHub, Etc….
  - Or based on the standard protocol OpenID Connect

- AuthoriZation (what you can do)
  - Based on claims that you can present (for example: a role you belong to)
  - De-facto standard today is OAuth2 + JWT
  - JWT: “JSON Web Tokens are an open, industry standard RFC 7519 method for representing claims securely between two parties.”

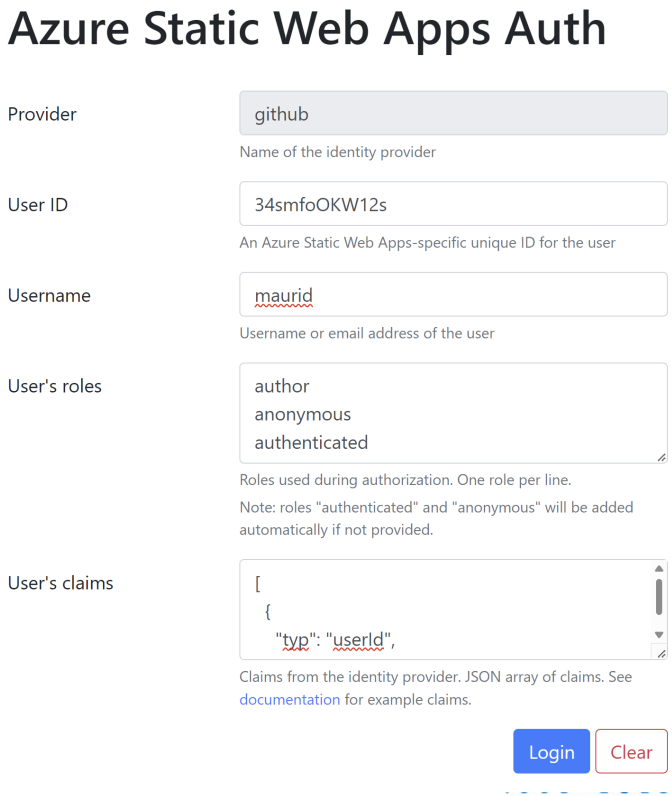
Easy Auth

- Implementing AuthN & AuthZ is a quite complex process. Easy Auth can simplify it *a lot*
  - https://learn.microsoft.com/en-us/azure/app-service/overview-authentication-authorization

- Azure Static Web Apps can use Easy Auth
  - It is enabled by default
  - Pre-Configured providers: AAD, GitHub, Twitter

Auth Mocking with SWA CLI

- SWA CLI allows to mock and read authentication and authorization credentials.
- Just go to “/.auth/login/<auth-provider>”
- Just like it will happen when deployed in Azure



3.0

DEMO

Resilient Connections

- There are no guarantees that a connection with the database will be always available
  - There could be networking issues from on-prem or where your service is hosted
  - There could be DNS issues
  - There could be internet provider issues
  - Azure SQL Database could be not available
    - **very unlikely**, SLA is 99.99% at least

Resilient Connections

To setup retries is your responsibility

Retry Frameworks

- .NET -> built in SqlClient retry policies
  - https://learn.microsoft.com/dotnet/api/microsoft.data.sqlclient.sqlconnection.retrylogicprovider?view=sqlclient-dotnet-standard-5.1
  - https://learn.microsoft.com/sql/connect/ado-net/configurable-retry-logic
- .NET -> Polly.NET
  - EF Core has some Retry Logic embedded already: https://learn.microsoft.com/en-us/ef/ef6/fundamentals/connection-resiliency/retry-logic
- Python -> Tenacity
- JavaScript -> node-retry

Polly

```
var p = Policy
    .Handle<SqlException>(CheckIfTransientError)
    .Or<TimeoutException>()
    .OrInner<Win32Exception>(CheckIfTransientError)
    .WaitAndRetry(5, retryAttempt => TimeSpan.FromSeconds(Math.Pow(2, retryAttempt)),
        (exception, timeSpan, retryCount, context) =>
        {
            var details = string.Empty;
            if (exception is SqlException) {
                var sx = exception as SqlException;
                details = $" [SQL Error: {sx.Number}]";
            }
            Console.WriteLine($"[{options.SampleId:00}] Retry called {retryCount} times, nex
```

```
p.Execute(() =>{
    using (var conn = new SqlConnection(_connectionString))
    {
        var ei = conn.QuerySingle<ExecutionInfo>(query);
    }
});
```

4.0

DEMO

CI/CD Integration

- The Database Migrations Way
  - The scripts are generated for you by the ORM
  - The scripts are applied automatically for you by the ORM (or its tools)
  - EF Core: Migrations
- The Declarative Way
  - You take a “snapshot” of the desired database state
  - A tool will sync your target database to the state you desire
  - Database Project -> DacPac -> SqlPackage

SQL projects: interface to a database schema

- Objects (tables, stored procedures, functions) are validated as a complete model
  - Do all the references match?
  - Does the specified engine version have that capability?
- *Pre/post deployment scripts add additional actions before/after model is applied, eg:
  - Insert static data
  - Add users
- Advanced capabilities: variable evaluation at deployment (SQLCMD), dependent/nested projects

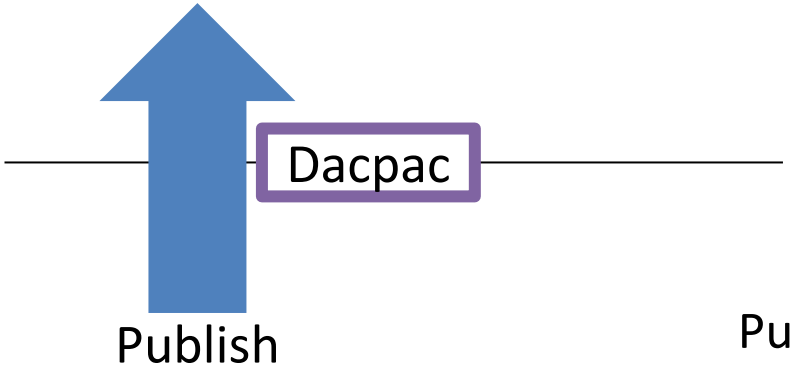
Continuous integration (CI)

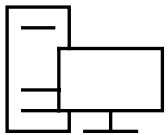
- tSQLt

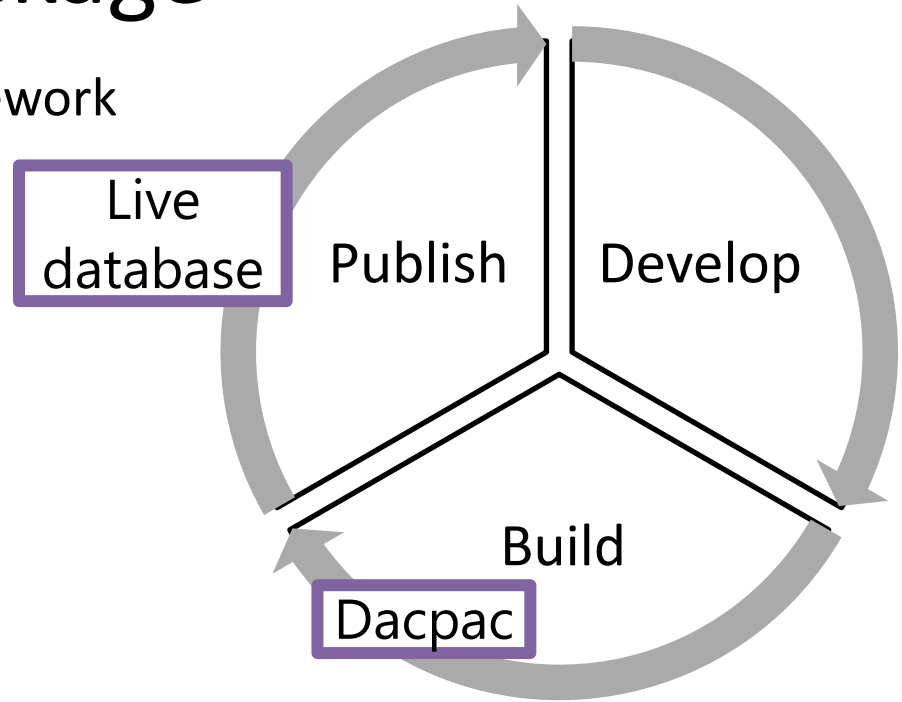
Anatomy of a Dacpac

SqlPackage

Command line interface to DacFx framework







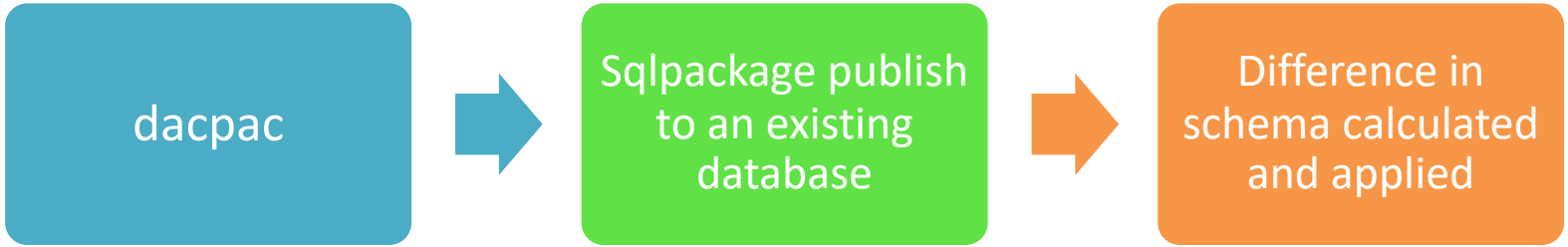
Publish is an incremental operation

Updates the existing database schema to match the dacpac

Example: Publish and create new database


```
CREATE TABLE Customer
(
    FirstName NVARCHAR(100) NOT NULL,
    LastName NVARCHAR(100) NOT NULL
)
```

Example: Publish to existing database




```
CREATE TABLE Customer
(
    FirstName NVARCHAR(100) NOT NULL,
    LastName NVARCHAR(100) NOT NULL,
    Region NVARCHAR(100) NULL
)
```

Deployment Plan

```
ALTER TABLE Customer
ADD Region NVARCHAR(100)
```

1993 – 2023

Build/Publish Example (SDK-style)

```yaml
jobs:
  build:
    # SDK sqlproj build on Windows + Linux
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2

      - name: Azure SQL Deploy
        uses: Azure/sql-action@v2
        with:
          # The connection string, including auth info, for the Azure SQL Server database
          connection-string: ${{ secrets.AZURE_SQL_CONNECTION_STRING }}
          # Path to DACPAC file to deploy
          path: database/AdventureWorksLT.sqlproj
          action: Publish
          # additional SqlPackage.exe arguments, leave all existing indexes in place
          arguments: /df:DatabaseProject/Log.txt /p:DropIndexesNotInSource=False
```

Infrastructure as code (IaC)

- ARM templates
- Bicep
- Terraform

5.0

DEMO

Vue3

- If project is complex enough, split the UI in reusable, self contained, “components”
  - .vue files – Single File Components (SFCs)
- SFCs require a *build* step
  - A project can be initialized using

```
npm init vue@latest
```

- Vue has (a recently updated) build tool: https://vitejs.dev/
  - SWA CLI supports Vue tool ☺
- Debug Vue using Vue DevTools

Vue3 – Some Notes

- Fix for hot-reload and WSL2. Update *vite.config.js*

```
server: {
  hmr: true,
  watch: {
    usePolling: true
  }
},
```

- Make sure you have installed node modules: `npm install`

Vue3 resources to get started

- Getting Started using the CDN
- Component Basics
  - Passing data between components: “Props”
- Routing
- Book: “Vue.js 3 By Example”

6.0

DEMO

Scaling Out

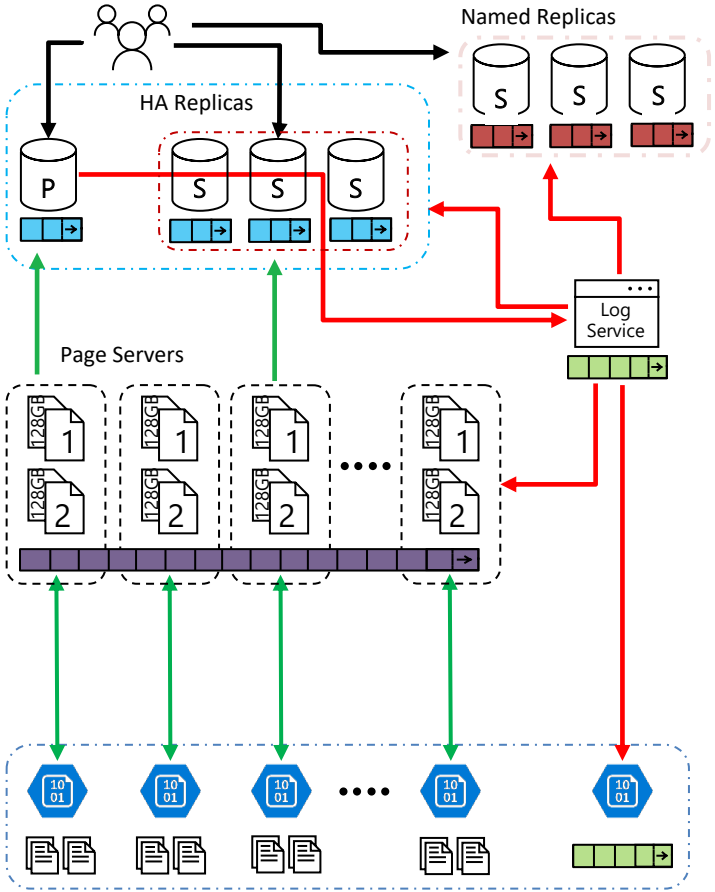
- It is (generally) easy to scale-out the backend services
  - Make sure services are stateless
  - Create as many instances as required
  - Use a routing/load balancing solution (Azure can do that for you)

- For databases is a bit more complex, as state cannot be pushed somewhere else
  - They *are* the state!
  - But with some refactoring it can be easy to scale-out reads
    - Reads are usually 70% to 80% of an OLTP workload

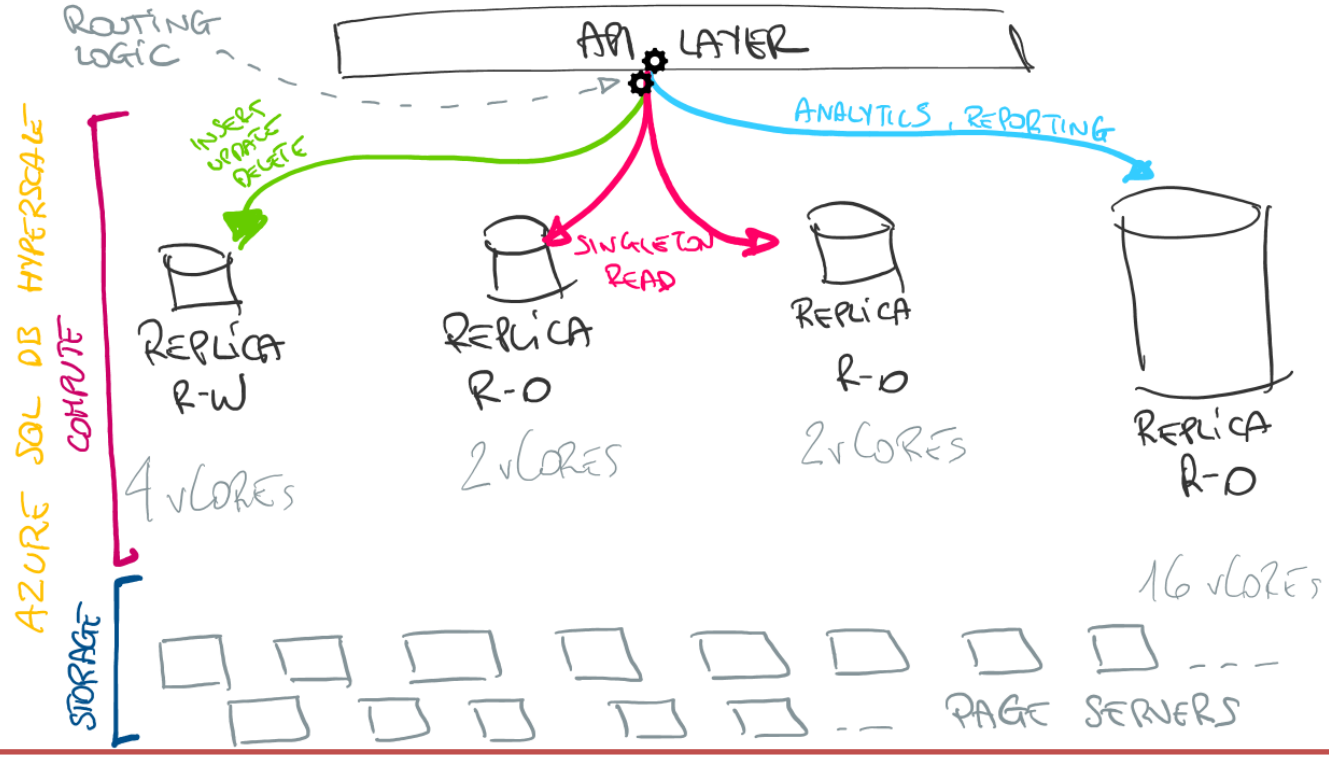
Azure SQL DB Hyperscale

- Separation of compute & Storage.
- Compute with Non-covering SSD cache
- Externalized log service
- Paired page servers, fully covering SSD
- Redundant data and log in Azure Storage
- 0 to 4 secondary HA replicas
  - Higher SLA with HA replica
- 0-30 Named replicas for read scale
- Backup/Restore via snapshots

Azure SQL DB Hyperscale



Tag-Based Routing



https://github.com/Azure-Samples/azure-sql-db-named-replica-oltp-scaleout

Bulk Operations

- Use bulk operations to operate on significant amount of rows
- Performance impact is HUGE
  - Every operation as a bit of overhead
  - Tiny overhead * Many small operations = Big overhead
  - Tiny overhead * A few big operations = Very Small overhead
- For Azure SQL + .NET = SqlBulkCopy Class

Bulk Operations

```
Running 1000 INSERT

Running *MULTIPLE BATCHES* sample
Elapsed: 3.413 secs

Running *SINGLE-BATCH-LOOKALIKE* sample
Elapsed: 2.251 secs

Running *TVP* sample
Elapsed: 0.15 secs

Running *JSON* sample
Elapsed: 0.186 secs

Running *Row Constructors* sample
Elapsed: 0.38 secs

Running *BulkCopy* sample
Elapsed: 0.116 secs
```

Bulk Operations

- EF Core
- Dapper
  - Not supported by default, need to add a package
- Manual
  - (use SqlBulkCopy class)

Ad-hoc

DEMO

Alternatives

- Database
  - Postgres, MySQL, MongoDB, Cosmos DB
- Frontend
  - React, Next.Js, Svelte
- Backend
  - Python, Node (JS/TS), Go, Rust

Conclusion

- We went through *a lot* today
- Learned the foundations of Fullstack/Jamstack
  - Architecture and technologies
- Learned how to use Azure to build a Fullstack/Jamstack solution
  - Got a solid understanding of all the moving pieces
- Understand alternatives
- Felt empowered and happy

30th ANNIVERSARY VSLive! 1993 – 2023

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!

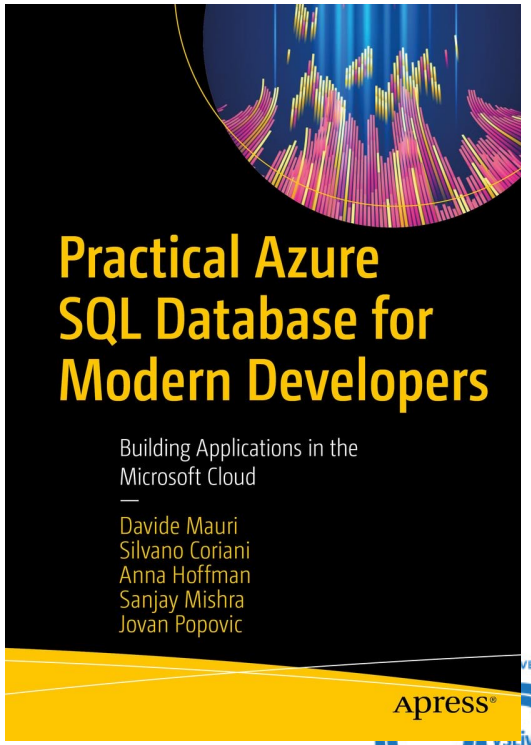
Thank you!

- Brian Spendolini
  bspendolini@microsoft.com
- Drew Skwiers-Koballa
  drskwier@microsoft.com

A book for the modern developer

- A developer-focused book, to help you leverage all the relational and post-relational features that Azure SQL has, to easily create fast, scalable and secure applications

- Lots of samples and discussions taking into account different languages:
  - Python, .NET, Java
  - …and more!



Additional Resources

- https://github.com/azure-samples/azure-sql-db-dynamic-schema/
- https://github.com/Azure-Samples/azure-sql-db-named-replica-oltp-scaleout
- https://github.com/yorek/awesome-azure-sql
- https://docs.microsoft.com/en-us/samples/browse/?products=azure-sql-database

Resources

- The primary repo: https://github.com/Azure-Samples/azure-sql-db-fullstack-serverless-kickstart
- Azure SQL DevCorner Blog: https://devblogs.microsoft.com/azure-sql/