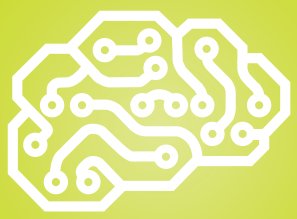
THE MICROSOFT JOURNAL FOR DEVELOPERS

JULY 2017 VOL 32 NO 7

msdn magazine

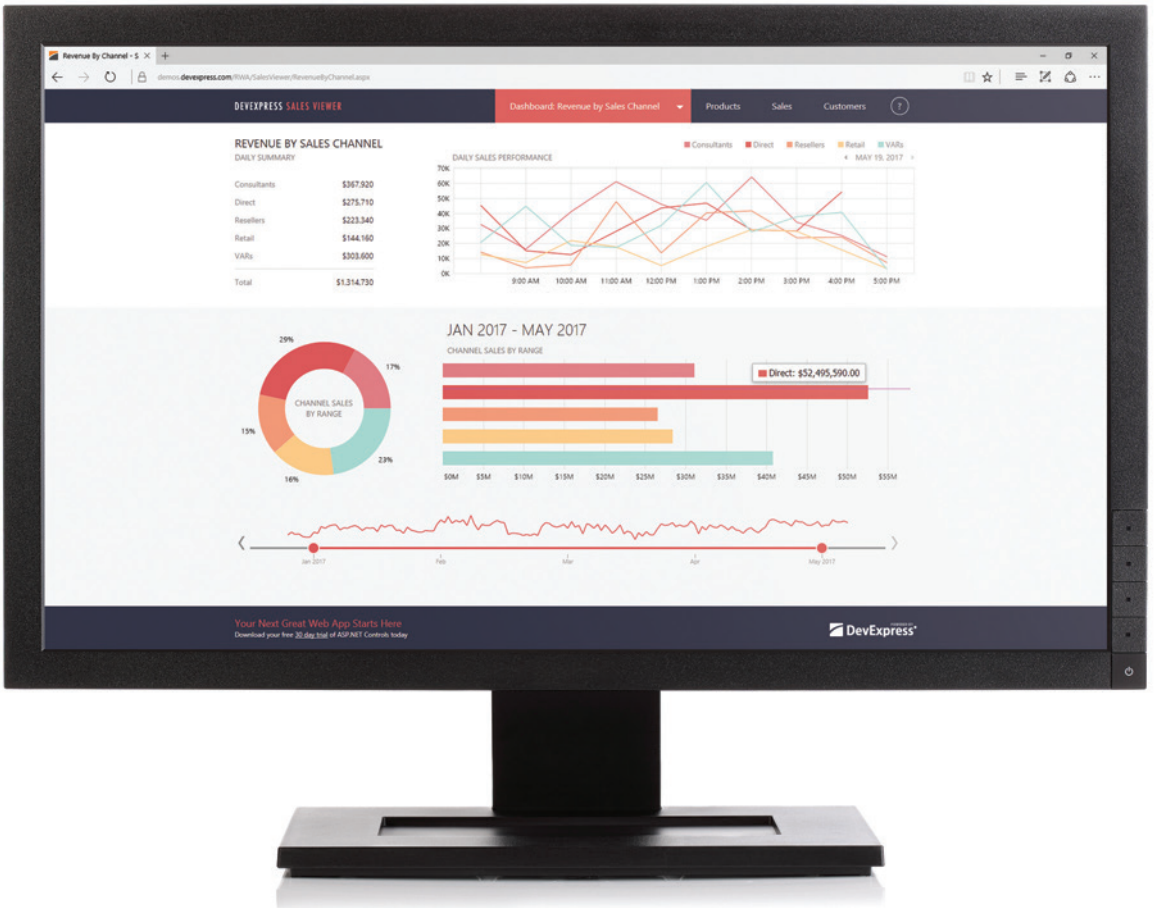


Machine Learning.....18, 22, 30, 34

DevExpress®

Your Next Great Web App Starts Here

From apps that replicate the look and feel of Microsoft Office® 365, to high-impact decision support systems for your enterprise, DevExpress Web Controls for ASP.NET will help you build your best, without limits or compromise.



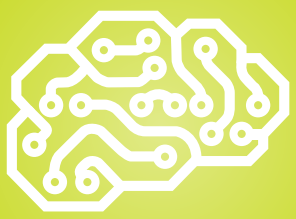
Download your free 30-day trial
an experience the DevExpress difference today.

devexpress.com/try

THE MICROSOFT JOURNAL FOR DEVELOPERS

JULY 2017 VOL 32 NO 7

msdn magazine



Machine Learning.....18, 22, 30, 34

Cognition at Scale with U-SQL on ADLA
Hiren Patel and Shravan Matthur Narayanamurthy 18

Introduction to the Microsoft CNTK v2.0 Library
James McCaffrey 22

Doing Data Science and AI with SQL Server
Wee Hyong Tok 30

Scale Applications with Microsoft Azure Redis and Machine Learning
Stefano Tempesta 34

ASP.NET Core with Angular, Web API and Azure DocumentDB
Chander Dhall 48

COLUMNS

UPSTART
The First Quarter
Krishnan Rangachari, page 6

CUTTING EDGE
Finding the Cheese in ASP.NET Core
Dino Esposito, page 8

DATA POINTS
On-the-Fly SQL Servers with Docker
Julie Lerman, page 12

THE WORKING PROGRAMMER
How To Be MEAN: Angular Ins and Outs
Ted Neward, page 56

MODERN APPS
Launch Other Applications from Your UWP App
Frank La Vigne, page 64

DON'T GET ME STARTED
Live and in Concert
David Platt, page 72

Microsoft


TRAFFIC BY REGION
SPEND
CONVERSIONS
VISITS
UNIQUE VISITORS
2.3M
CONVERSION RATE
10%
CAMPAIGN HEALTH
20%
TOTAL
200K
CONVERSIONS
SALES DASHBOARD
MONTH TO DATE 9/1 - 9/16
$303,876
Actual
$340K
Forecast
REGION | DIVISION
REGION TARGET FORECAST
Americas $873,030
Asia $580,200
Africa $359,876
Australia $200,387
Europe $100,876
TOTAL $2,114,369
$2,114,369
LANDSCAPE VIEW FOR MORE DETAILS
SALES HISTORY
upcoming
starting 4/12 Nova No
starting 4/15 Haddad
starting 5/10 Hancock
hours worked
tasks by resource
Prototype Interactions
Design Research
Oversee Prototyping
Bind Data to Server
Review Prototypes
Review Prototypes
Projected End 3/13
Projected End 3/13
Projected End 3/13
Projected End 3/12
Projected End 3/10
Projected End 3/13
90 hours
55 hours
67 hours
60 hours
35 hours
10 hours
0.56 (-2.12%)
1Y 2Y 5Y 10Y MAX
Stock Price
CHART
Daily Range
Annual Range
Open
Volume
26.50
1.11 M
Adobe Systems Inc
2013-07-26 $47.64
Microsoft Corporation
2013-07-26 $31.62
Intel Corporation
2013-07-26 $23.26
Jun 2013
Jul 2013
Aug 2013
Sales Volume

msdn magazine

JULY 2017 VOLUME 32 NUMBER 7

ID STATEMENT MSDN Magazine (ISSN 1528-4859) is published 13 times a year, monthly with a special issue in November by 1105 Media, Inc., 9201 Oakdale Avenue, Ste. 101, Chatsworth, CA 91311. Periodicals postage paid at Chatsworth, CA 91311-9998, and at additional mailing offices. Annual subscription rates payable in US funds are: U.S. $35.00, International $60.00. Annual digital subscription rates payable in U.S. funds are: U.S. $25.00, International $25.00. Single copies/back issues: U.S. $10, all others $12. Send orders with payment to: MSDN Magazine, P.O. Box 3167, Carol Stream, IL 60132, email MSDNmag@1105service.com or call (847) 763-9560. POSTMASTER: Send address changes to MSDN Magazine, P.O. Box 2166, Skokie, IL 60076. Canada Publications Mail Agreement No: 40612608. Return Undeliverable Canadian Addresses to Circulation Dept. or XPO Returns: P.O. Box 201, Richmond Hill, ON L4B 4R5, Canada.

CORPORATE ADDRESS 1105 Media, 9201 Oakdale Ave. Ste 101, Chatsworth, CA 91311 www.1105media.com

MEDIA KITS Direct your Media Kit requests to Chief Revenue Officer Dan LaBianca, 972-687-6702 (phone), 972-687-6799 (fax), dlabianca@1105media.com

REPRINTS For single article reprints (in minimum quantities of 250-500), e-prints, plaques and posters contact: PARS International Phone: 212-221-9595. E-mail: 1105reprints@parsintl.com. www.magreprints.com/QuickQuote.asp

LIST RENTAL This publication's subscriber list, as well as other lists from 1105 Media, Inc., is available for rental. For more information, please contact our list manager, Jane Long, Merit Direct. Phone: 913-685-1301; E-mail: jlong@meritdirect.com; Web: www.meritdirect.com/1105

Reaching the Staff

Staff may be reached via e-mail, telephone, fax, or mail. A list of editors and contact information is also available online at Redmondmag.com.

E-mail: To e-mail any member of the staff, please use the following form: FirstinitialLastname@1105media.com

Irvine Office (weekdays, 9:00 a.m. – 5:00 p.m. PT)
Telephone 949-265-1520; Fax 949-265-1528
4 Venture, Suite 150, Irvine, CA 92618

Corporate Office (weekdays, 8:30 a.m. – 5:30 p.m. PT)
Telephone 818-814-5200; Fax 818-734-1522
9201 Oakdale Avenue, Suite 101, Chatsworth, CA 91311

The opinions expressed within the articles and other contentsherein do not necessarily express those of the publisher.

Microsoft

BPA WORLDWIDE

1906 2006 100 AMERICAN BUSINESS MEDIA

WPA Western Publications Association

LEADTOOLS®

THE WORLD LEADER IN IMAGING SDKs

BUILD BETTER APPS WITH LEADTOOLS

DOCUMENT

- Document Viewer & Converter
- OCR, MICR, OMR & ICR
- 1D & 2D Barcode
- Forms Recognition
- PDF, DOCX, HTML, SVG, RTF, TXT
- Annotations & TWAIN

MEDICAL

- DICOM, CCOW & HL7
- PACS Client & Server Framework
- DICOMWeb (WADO)
- Web & Desktop Viewers
- Image Processing & Annotations
- Medical 3D (MPR, MIP, VRT)

MULTIMEDIA

- Play, Capture, Convert & DVR
- Media Streaming
- MPEG-2 TS, RTSP, HTML5 & more
- OGG, FLV, ISO, AVI, WebM & more
- Audio & Video Processing
- DirectShow & Media Foundation

.NET Windows API Java WinRT Linux iOS macOS Android JavaScript

DOWNLOAD OUR 60 DAY EVALUATION
WWW.LEADTOOLS.COM

LEAD TECHNOLOGIES INCORPORATED

SALES@LEADTOOLS.COM
800.637.1840

MICHAEL DESMOND

Machine Learning Goes Mainstream

At the Microsoft Build 2017 conference in Seattle in May, Microsoft detailed its deep investments in artificial intelligence (AI), showing how solutions like Cognitive Services and Azure Machine Learning can be leveraged to change the way people work, live and interact. At the core of this push is machine learning—the science of applying data to enable systems to make increasingly better decisions over time.

Machine learning is addressing prominent challenges, from financial fraud detection to manufacturing quality control to autonomous cars. This month in *MSDN Magazine*, we dedicate four articles to the topic of machine learning and explore how developers can employ it in their software development.

Hiren Patel leads off the coverage with "Cognition at Scale with U-SQL on ADLA," which dives into the U-SQL Studio IDE to show how the Microsoft Azure Data Lake Analytics distributed engine can be used to train and score models at scale.

James McCaffrey then provides a look at the Microsoft Cognitive Toolkit (CNTK) v2.0 Library, which lets developers create machine learning prediction models based on deep neural networks—technology that's at the forefront of AI efforts such as Cortana and self-driving automobiles.

"Doing Data Science and AI with SQL Server" by Wee Hyong Tok shows how SQL Server 2016 is bursting with intelligence capabilities that allow developers to run R code right where the data is. Tok walks through building an intelligent application and shows how the R code developed by data scientists can be used in database projects.

Finally, "Scale Applications with Azure Redis and Machine Learning" presents design best practices and code examples for implementing the Azure Redis Cache and tuning the performance of ASP.NET MVC applications to optimize cache hit ratio. Stefano Tempesta shows how smart algorithms processed by machine learning can help reduce cache "miss rate."

'A Must-Have Skill'

Our coverage this month reflects the growing importance of machine learning in software development—and not just in specialized environments. As McCaffrey notes, machine learning is at the foundation of deep learning techniques and AI development. It's also coupled tightly with Big Data and Internet of Things development.

"There's a general consensus among my senior dev colleagues that having at least basic machine learning knowledge is quickly becoming a must-have skill with regard to career growth," McCaffrey says, adding, "Our HR people at Microsoft say that developers with machine learning skills are among the most requested, and most difficult to find."

That value is reflected in the growing membership of Microsoft's internal machine learning community, which stands at more than 5,500 developers, data scientists, architects, consultants and others since its formation in 2014. Alex Blanton is a senior program manager in the Data Group at Microsoft and community manager of the internal machine learning community at Microsoft. He says that more than 3,000 people attended the group's latest Machine Learning, Analytics and Data Science (MLADS) conference in Redmond, which had presentations covering topics like Azure Data Lake, deep learning tools and techniques, Microsoft Bot Framework, security analytics, and more.

The group also hosts frequent online and in-person talks and events, semi-annual machine learning competitions, and an active discussion list used by the community to pose questions and share knowledge across the company. These engagements offer an opportunity to "discover pockets of excellence" in the community, Blanton says, and explore topics as they emerge in the fast-moving machine learning space.

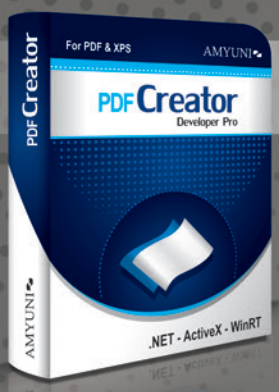
"We pay some attention to connecting machine learning to other domains—for example, an upcoming talk is from a designer talking about how AI and machine learning can impact design," Blanton says.

This issue is just scratching the surface of what developers can expect from Microsoft in the realm of machine learning and AI. I expect we'll see a lot more on these topics in the months and years to come.

Visit us at msdn.microsoft.com/magazine. Questions, comments or suggestions for *MSDN Magazine*? Send them to the editor: mmeditor@microsoft.com.

Switch to Amyuni PDF

AMYUNI



Create and Edit PDFs in .NET, COM/ActiveX, WinRT & UWP

NEW v5.5

- Edit, process and print PDF 1.7 documents
- Create, fill-out and annotate PDF forms
- Fast and lightweight 32- and 64-bit components for .NET and ActiveX/COM
- New Universal Apps DLLs enable publishing C#, C++, CX or Javascript apps to windows Store
- Updated Postscript/EPS to PDF conversion module

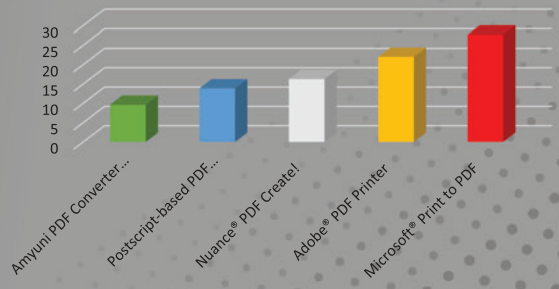
Complete Suite of Accurate PDF Components

- All your PDF processing, conversion and editing in a single package
- Combines Amyuni PDF Converter and PDF Creator for easy licensing, integration and deployment.
- Includes our Microsoft WHQL certified PDF Converter printer driver
- Export PDF documents into other formats such as Jpeg, PNG, XAML or HTML5
- Import and Export XPS files using any programming environment

High Performance PDF Printer for Desktops and Servers

- Print to PDF in a fraction of the time needed with other tools. WHQL tested for all Windows platforms. Version 5.5 updated for Windows 10 support

Benchmark Testing - Amyuni vs Others
Seconds required to convert a document to PDF



Other Developer Components from Amyuni®

- WebkitPDF: Direct conversion of HTML files into PDF and XAML without the use of a web browser or a printer driver
- PDF2HTML5: Conversion of PDF to HTML5 including dynamic forms
- Postscript to PDF Library: For document workflow applications that require processing of Postscript documents
- OCR Module: Free add-on to PDF Creator uses the Tesseract engine for character recognition
- Javascript engine: Integrate a full Javascript interpreter into your applications to process PDF files or for any other need

All development tools available at

www.amyuni.com

AMYUNI Technologies

USA and Canada
Toll Free: 1866 926 9864
Support: 514 868 9227
sales@amyuni.com

Europe
UK: 0800-015-4682
Germany: 0800-183-0923
France: 0800-911-248

The First Quarter

I've noticed that if I do well in my first three months in a new engineering job, I can build an enduring reputation and foundation for success. Here are six strategies I use in those initial months that help win the trust of my coworkers and put me on the right track.

Delayed Start

I push back my official start date, and use the extra time to build connections. So, instead of starting in two weeks, I may start in six weeks. In that one extra month, I meet up with people on the new team, one-on-one. I introduce myself, and just learn more about them. My goal: Establish personal rapport.

I start with the people who interviewed me, because I already know that these people like me. I treat them to coffee or lunch. I ask them open-ended questions like, "What was your process of getting on-boarded onto the team?" and, "How is being an engineer here different from other places you've worked at?"

As I go through this process, I'll discover some people will go out of their way to help me. They'll share insider tips and tricks, and secret knowledge. I remember those people, and I go back to them for more advice once I'm at the job.

I also ask them, "Who else would you recommend I chat with, so I can learn more about the company?"

There are two major advantages to having these chats *before* I start the job. First, once I'm on the clock I won't have this kind of free time. Second, engineers are more frank when I'm not yet on the job. They'll give me the scoop on things like toxic projects to avoid and the company's priorities in ways they can't after I'm in my role.

Managed Intro

My new team will introduce me to everyone, either in person or via e-mail. During these introductions, developers artificially puff up their chests. They try to look more confident and secure than they feel. A side effect: The developers come off boastful.

It's normal to feel insecure when starting any new job. Instead of trying to kill the insecurity with pride and boasting about my great credentials, I embrace it with humility. If I boast about my background, I convey that I think more highly of myself than my new team. It's better to let my performance speak for itself.

Manager Maximization

I start to meet both engineering and *non*-engineering managers, one-on-one. This includes team leads for other dev teams. This also includes managers in product marketing, technical sales, IT, product support, operations and more. When I do this, I stand apart immediately.

Through these meetings, I get a more holistic perspective of the company. I'll know what projects are actually most pressing to the whole company, allowing me to select more meaningful projects. I also can start to build a pipeline of project ideas of my own and, ultimately, accelerate my career growth.

Three Projects

I identify three projects—or, more realistically, parts of three projects—that I can get started on and finish within the first three months. These projects should allow me to learn the ropes of my new role without overwhelming me.

Instead of doing whatever projects land on my desk, I cultivate the courage to say no and exercise discrimination. In fact, as a new employee, I'm actually in the *best* position to say no. For example, I may say, "I don't feel ready to take on that project," "I'm a little overwhelmed at the moment," or even, "I don't think that project will work for me."

Start Delivering

Once I select projects, I don't wait until they're done for some grand reveal. *I* am the grand reveal. The crucial part isn't the project itself, it's how I execute on the project.

For example, in stand-up meetings, instead of vaguely saying, "I'm working on this project," I may instead state three or four actions I did for the project yesterday. This publicizes my effort without boastfulness, and promotes the hundred baby steps that take me to the end goal. The upside: Even if the end result isn't as grand as I expected it to be, it's OK. I've made the most of the journey there.

Cultivate Champions

As I execute on a project, I'll often go to others for help. If someone was particularly helpful, I follow up later and tell them what I did. This way, I signal that not only do I ask for advice, but I also follow it. When I show this kind of gratitude, these people take me under their wing. They're more likely to give me good advice in the future, too.

Some new developers make the mistake of asking others to be their mentors. In reality, mentorship is cultivated *over time*. Sometimes, asking someone to be a mentor is the surest way to *not* have it happen. It burdens them with an artificial, inflated title that they feel they must live up to. It's more effective to have someone become my champion over time, without either of us realizing it. ■

KRISHNAN RANGACHARI *helps brilliant developers have amazing careers. Visit RadicalShifts.com for his free courses.*

n software

We didn't invent the Internet...

...but our components help **you** power the apps that bring it to business.

TOOLS • COMPONENTS • ENTERPRISE ADAPTERS

- E-Business
  AS2, AS4, EDI/X12, OFTP …
- Credit Card Processing
  Authorize.Net, ACH, 3-D Secure …
- Shipping & Tracking
  FedEx, UPS, USPS …
- Accounting & Banking
  QuickBooks, OFX, SAP …
- Internet Business
  Amazon, PayPal, Google …
- Internet Protocols
  FTP, SMTP, IMAP, POP, WebDav …
- Secure Connectivity
  SSH, SFTP, SSL, Certificates …
- Encryption & Certificates
  X.509, OpenPGP, SHA, S/MIME …
- Network Management
  SNMP, MIB, LDAP, Monitoring …
- Compression
  Zip, Gzip, Jar, AES, 7Zip …

Our **Red Carpet Subscription** includes all product lines + updates for one year.

The Market Leader in Internet Communications, Security, & E-Business Components

Each day, as you click around the Web or use any connected application, chances are that directly or indirectly some bits are flowing through applications that use our components, on a server, on a device, or right on your desktop. It's your code and our code working together to move data, information, and business. We give you the most robust suite of components for adding Internet Communications, Security, and E-Business Connectivity to any application, on any platform, anywhere, and you do the rest. For more than 20 years, we have had one goal: to provide the very best connectivity solutions for our professional developer customers. With more than 100,000 developers worldwide using our software and millions of installations in almost every Fortune 500 and Global 2000 company, our business is to connect business, one application at a time.

connectivity powered by

To learn more please visit our website → **www.nsoftware.com**

Microsoft .NET | Java | C++ | Delphi | C++ Builder | ActiveX | Linux/Unix | macOS | Android | iOS | PHP | Xamarin | Node.js

DINO ESPOSITO

Finding the Cheese in ASP.NET Core

Do you know the fable of two mice, two little boys and their cheese? It's not an Aesop's classic fable, but it's a modern account full of motivational points written some 20 years ago by Spencer Johnson under the title, "Who Moved My Cheese?" The story is a metaphor for change and the way we should cope with and adapt to it. I dare say that the first, perhaps unconfessed, sentiment that some developers feel about ASP.NET Core is dismay—like in this fable when the characters reach out to the cheese station only to find it empty.

Who moved my cheese, then? The good news is that there's still a lot of cheese around for every ASP.NET Core developer—probably more cheese than ever—but some of it's no longer in the corridor of the development maze where it was before. Out of metaphor, that means that some common ASP.NET MVC tasks require different programming skills and sometimes a different approach. This month, I'll pick up on actions such as registering the Model-View-Controller (MVC) framework, defining core routes and controllers, and passing global data around.

It's No Longer MVC By Default

A new empty ASP.NET Core project you create in Visual Studio 2015 or Visual Studio 2017 is primarily a Web project, meaning that it produces a server front end for an HTTP-enabled client to call. As pointed out in my May 2017 column (msdn.com/magazine/mt808498), ASP.NET Core is now great at setting up mini-Web servers devoid of the overhead and sophistication of a full Web site. However, ASP.NET Core isn't limited to that and still provides plenty of opportunities to arrange full Web sites through the familiar programming pattern of routes, controllers and views. ASP.NET Core, though, requires an extra initial step before you can start adding routes and creating controllers and lets you register routes through a slightly different approach than in classic ASP.NET MVC. Let's see how it works.

As mentioned, ASP.NET Core is a plain Web front end based on two main segments of code: routing and application model. The transition between segments isn't particularly visible to developers and takes place in the MvcRouteHandler service (see bit.ly/2osQOcs). Once the control leaves the routing repository, the MVC application model can be enabled explicitly. If not, any requests end up being processed in the terminating method of the ASP.NET Core middleware. To add the MVC service to the ASP.NET container, you add a line of code to the ConfigureService method of the startup class:

```
public void ConfigureServices(IServiceCollection services)
{
  services.AddMvc();
}
```

Note that the code requires a reference to an additional package that the IDE of choice (Visual Studio) typically offers to restore for you. The parameter-less version of the AddMvc method uses all default settings for the MVC route handler service. Most of the time, you don't need to change any of those default settings, but surely situations might occur where you just need to make changes. A second overload of the AddMvc method lets you select ad hoc options. The second overload receives an instance of the MvcOptions class as an argument. Here's an example:

```
services.AddMvc(options =>
{
  options.ModelBinderProviders.Add(new MyDateBinderProvider());

  options.SslPort = 345;
});
```

The MvcOptions class is a container of configuration parameters for features you might want to customize in the MVC framework. For example, the previous code snippet adds a new model binder that changes the standard way in which posted dates are mapped to .NET date objects. You can assume that the MyDateBinderProvider class here adds the ability to parse ad hoc strings into valid DateTime objects. In addition, the example specifies the SSL port to be sniffed when any controller class is decorated with the RequireHttps-Attribute. The list of options you can configure isn't limited to the three examples here. The full list can be found at bit.ly/2oK7ETs.

It's worth noting that the AddMvc method is an umbrella method under which a lot of finer-grained services are initialized and added to the pipeline. Because of this, the effect of AddMvc on the memory footprint of the whole application might need some forethought. Not that AddMvc bloats the runtime, but it does quite a few things internally and atomically enables quite a few services. In addition to the core services of the MVC application model, such as routes and controllers, it also enables authentication and authorization, tag helpers, URL resolution helpers, default media type mappings, data annotations, and cross-origin resource sharing (CORS). Furthermore, AddMvc sets up the service to process action results as HTML views and JSON streams and registers the Razor view engine into the MVC system.

Cutting Down the List of Default MVC Services

Especially if you're hosting the application in some sort of cloud configuration, you might want to be very stingy about resources and have the application reference nothing but the bare metal of the ASP.NET framework. Calling AddMvc might give much more than you need sometimes. Let's see how to make the list of

references shorter. The following code is enough to serve plain HTML views to browsers. It should be noted, though, that it doesn't support some more advanced features, including data annotations for form validation and tag helpers:

```
public void ConfigureServices(IServiceCollection services)
{
  var builder = services.AddMvcCore();
  builder.AddViews();
  builder.AddRazorViewEngine();
}
```

Likewise, this configuration won't let your controller methods return formatted JSON data. However, you need one extra line to add that capability, as well:

```
builder.AddJsonFormatters();
```

It's worth noting that some of the services the call to AddMvc automatically enables are useful to have, though not strictly necessary, only if you're exposing a Web API. The services you might want to get rid of are API Explorer and Formatter Mappings and, to some extent, CORS.

Enabling the MVC Service

Adding a service to the pipeline isn't enough: You also have to configure it. This typically happens in the Configure method of the startup class via a method that conventionally takes the name of UseXxx where Xxx is the nickname of the service:

```
public void Configure(IApplicationBuilder app)
{
  app.UseMvc();
}
```

At this point, everything in MVC is completely set up and ready to go except conventional routing. If you decide to go with attribute routing, then you're done. Otherwise, for the MVC service to be effective, you must list the routes the application will recognize and handle. A route is a URL template mapped to a pair made of controller and action names. You can add as many routes as you wish and of nearly any shape you like them to be. ASP.NET MVC, however, provides a default route that serves most of the common scenarios. To enable the default route, you can proceed like this:

```
public void Configure(IApplicationBuilder app)
{
  app.UseMvcWithDefaultRoute();
}
```

The default route is defined as follows and, as you can see, it's nothing more than the old familiar route configuration of classic ASP.NET MVC:

```
routes.MapRoute(
  name: "default",
  template: "{controller=Home}/{action=Index}/{id?}");
```

In classic ASP.NET MVC, a routeless application makes little sense as it would be quite problematic to invoke any behavior from the outside. In ASP.NET MVC Core, instead, routes work side-by-side with the terminating middleware—the Run method of IApplicationBuilder:

```
public void Configure(IApplicationBuilder app)
{
  app.UseMvc(routes => { });

  // Terminating middleware
  app.Run(async (context) =>
  {
    await context.Response.WriteAsync(
      "No configured routes here.");
  })
}
```

Given the preceding code, the application has no configured routes. However, it still reacts to any call by displaying a message through the terminating middleware. In other words, you can list your routes first and then use the terminating middleware as a sort of catch-all route that gently informs about any mistyped or misunderstood URLs.

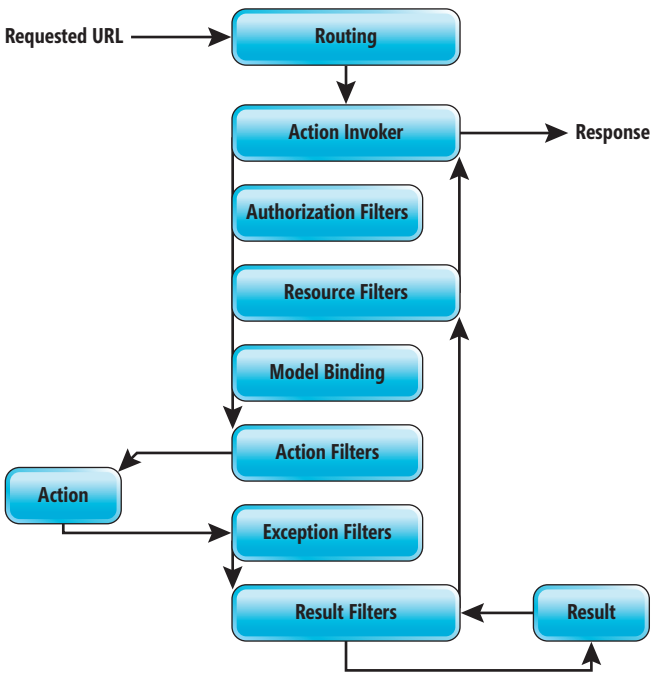


Figure 1 **A Request Flowing Through the ASP.NET Environment**

MVC Controllers

Routing is the first step of the longer process that takes an HTTP request to produce a response. The ultimate result of routing is identifying the controller/action pair that will process any requests not mapped to a physical static file. In ASP.NET Core a controller is the same as it was in classic ASP.NET, namely a class that encapsulates the HTTP context of the request and takes action. The work of a controller is governed by a system component known as the action invoker (see **Figure 1**). The action invoker isn't a new item in the overall architecture of ASP.NET MVC as it was part of the architecture since the early days of classic ASP.NET MVC.

The action invoker injects the HTTP context into the controller's space and the code running within the controller can access it through the handy HttpContext property. To facilitate the process in classic ASP.NET MVC, any controller class must inherit from a base class that contains all the necessary plumbing. In ASP.NET Core, inheriting a controller from a common base class is no longer necessary. A controller class can be a plain old C# object (POCO), as simple as this:

```
public class HomeController
{
  // Some code here
}
```

A POCO controller is a class that can map incoming requests to HTML views, but has no dependency on the HTTP context. In particular, this means that you can't inspect the raw data being posted, including query string and route parameters. The context information, however, is available as a separate plug-in that you attach only to the controllers where you need it. Ultimately, this is another good example of the extreme granularity of the ASP.NET Core framework. As an example, let's see what's required to access

route information from within a POCO controller. Interestingly, the feature to leverage is quite general and applicable and recommended for a number of other common programming activities:

```
public class HomeController
{
  private IActionContextAccessor _accessor;
  public HomeController(IActionContextAccessor accessor)
  {
    _accessor = accessor;
  }
  ...
}
```

The constructor of the controller class can declare a parameter of type IActionContextAccessor. The interface is the key to have context information injected into the controller's space. All that's required to do from within the controller is to save the received instance of the interface type for later use. The following code snippet shows how to access the RouteData object that contains any data tokenized into the handled route URL:

```
public IActionResult Index()
{
  var controller =
    _accessor.ActionContext.RouteData.Values["controller"];
  ...
}
```

Though possible, injecting the IActionContextAccessor service isn't recommended because it performs poorly and is rarely needed. Another way to access route data from within a POCO controller is using the FromRoute attribute to decorate a parameter in the action:

```
public IActionResult Index([FromRoute] string controller)
{
  ...
}
```

However, regardless of effectiveness, who injected a reference to IActionContextAccessor into the controller? That's where another relevant addition to the ASP.NET Core framework fits in—the internal Inversion of Control (IoC) pattern.

Sharing Global Data

Nearly every Web application holds some data to share globally. In classic ASP.NET, a common practice was to load global data at startup and save it (net of possible threading issues) into some global static variables exposed from the application object. It's not the perfect solution for everyone, but quite functional if you're aware of what you were doing. ASP.NET Core provides a better way to do it that guarantees that every application context receives just what it needs, thus reducing the risk of accessing unwanted data in unwanted contexts. Any chunk of global data must be added to the IoC subsystem in the ConfigureServices method. Here's how you would add the action context, for example:

```
public void ConfigureServices(IServiceCollection services)
{
  ...
  services.AddSingleton<IActionContextAccessor, ActionContextAccessor>();
}
```

The AddSingleton<TInterface, T> method registers that any requests for an object assignable to the TInterface type must be resolved through an instance of the concrete type T. Once the association has been mapped in the application startup, controllers will gain access to it by simply declaring a parameter of the type in their constructor.

Global information can then be read at the application startup from a variety of sources, including JSON and text files, databases, and remote services and packaged into a custom type. To make the object globally available you just add it to the IoC system via AddSingleton. The operation takes place only once.

An Ad Hoc Framework for Application Options

As far as global data is concerned, ASP.NET Core isn't limited to basic IoC functions, it also supports options. Options are a feature specifically designed to deal with the initial configuration of the application, namely mostly read-only data to be shared globally:

```
var builder = new ConfigurationBuilder()
  .SetBasePath(env.ContentRootPath)
  .AddJsonFile("MyAppSettings.json", optional: true, reloadOnChange: true);
Configuration = builder.Build();
```

In the constructor of the startup class, the previous code sets a given JSON file as the provider of configuration data. The builder uses the provided information to prepare and return an IConfigurationRoot object to be used to access data. You also declare the following in the startup class:

```
public IConfigurationRoot Configuration { get; }
```

Global data must be retrieved piece by piece through a query API. The options framework, instead, lets you load it into an aptly defined C# class to be passed around through the IoC system. Add the following code to the ConfigureServices method:

```
public void ConfigureServices(IServiceCollection services)
{
  services.AddOptions();
  services.Configure<GlobalConfig>(Configuration.GetSection("Globals"));
  ...
}
```

In the example, GlobalConfig is a custom class you define to be 1:1 with the content of the JSON initial data. The Configure method does a good job of reflection to read the specified segment of JSON into the C# class. As a pleasant side effect of the code, any controller now can be injected options through the following pattern:

```
public class HomeController : Controller
{
  private GlobalConfig Globals { get; }
  public HomeController(IOptions<GlobalConfig> config)
  {
    Globals = config.Value;
  }
  ...
}
```

Any global data can now be accessed via the Globals custom property.

Wrapping Up

As in the Johnson fable, with ASP.NET Core the problem isn't the cheese and its lack thereof. There's plenty of cheese in ASP.NET Core. The problem is the attitude to find it. Apparently, many things are different and not easily figuring out how to do basic things such as loading global data might be frustrating at first. Some ASP.NET MVC cheese has been definitely moved to a different location, but is now even tastier and more abundant. Next month, I'll touch on another relevant piece of cheese that's not where you always get it: forms authentication. ■

Dino Esposito *is the author of "Microsoft .NET: Architecting Applications for the Enterprise" (Microsoft Press, 2014) and "Modern Web Applications with ASP.NET" (Microsoft Press, 2016). A technical evangelist for the .NET and Android platforms at JetBrains, and frequent speaker at industry events worldwide, Esposito shares his vision of software at software2cents@wordpress.com and on Twitter: @despos.*

Thanks to the following Microsoft technical expert for reviewing this article: *Doug Bunting*

Data Quality Made Easy.
Your Data, Your Way.

Melissa provides the full spectrum of data quality to ensure you have data you can trust.

We profile, standardize, verify, match and enrich global **People Data** – name, address, email, phone, and more.

Our data quality solutions are available on-premises and in the Cloud – fast, easy to use, and powerful developer tools and plugins for the **Microsoft® Product Ecosystem.**

Microsoft® SQL Server® Integration Services

Microsoft Dynamics® CRM

Microsoft® .NET

Microsoft Azure

Microsoft® Excel

Start Your Free Trial

www.melissa.com/msft-pd

Melissa Data is now Melissa.
See What's New at www.Melissa.com

1-800-MELISSA

melissa™
GLOBAL INTELLIGENCE

JULIE LERMAN

On-the-Fly SQL Servers with Docker

In last month's column (msdn.com/magazine/mt809115), I used the Visual Studio Code mssql extension to interact with an Azure SQL Database—from my MacBook Pro! I love the cross-platform capability of Visual Studio Code, but I was still depending on Azure to provide me with some flavor of SQL Server. I also mentioned that it is indeed possible to run SQL Server without depending on the cloud or Windows, thanks to SQL Server for Linux and a Docker image that's readily available to drop onto your machine. I'll pause if the existence of SQL Server for Linux is new information for you and you just need a moment.

OK, I hope you've recovered. The Linux version of SQL Server is able to run in a Linux-based Docker container and you can use that container anywhere that Docker is supported.

Running SQL Server in a container is really useful for developers because it makes it easy and quick to spin up a SQL Server instance. You can run different instances of SQL Server that are different versions side by side. You can easily run the container for a while and then when you stop/delete the container, the data all goes away and you can start another clean one. If you want, you can also choose to persist the database files, but still start/stop the container only as you actually need SQL Server to be running.

There are two flavors of SQL Server container images—Linux-based images and Windows-based images. Microsoft provides a Linux-based image for SQL Server 2017 Developer Edition and three Windows-based images: SQL Server 2016 SP1 Express Edition, SQL Server 2016 SP1 Developer Edition and SQL Server 2017 Evaluation Edition. All of these images are available on Docker Hub for free (dockr.ly/2rmSiWs). You can pull and run the Linux-based SQL Server image to create Linux-based containers wherever there's a Docker Engine running, which could be on Linux, Windows or macOS. But you can run Windows-based containers *only* on Windows 10 Anniversary Edition or higher or Windows Server 2016—they can't run on Linux or macOS. Keep in mind that Docker also runs on Azure and AWS, so you can move from development to production in the cloud, as well.

Although I've written a blog post about my first experiments with running SQL Server for Linux in Docker on a Mac (bit.ly/2pZ7dDb), I want to approach this from a different angle—the scenario where you want to share a pre-configured image along with a database. This can allow developers to very quickly get SQL Server and the needed databases on their machines, or even to be used as part of an automated testing environment. I knew it could be done, but I was curious as to how, so I've worked through the basic process and will share it with you here.

Code download available at msdn.com/magazine/0717magcode.

I'll begin by explaining how to get the base image of SQL Server for Linux up and running on your computer under Docker. My example will use Docker for Mac, but you can do the same with the other versions of Docker, as well. Be sure you have the correct version of Docker already installed and running on your computer, and that you set it to use at least 4GB of RAM from the host system. You can find more detailed setup information on my blog post referenced earlier.

> Running SQL Server in a container is really useful for developers because it makes it easy and quick to spin up a SQL Server instance.

In the command or terminal window, you can get the official image by executing:

```
Mac: sudo docker pull microsoft/mssql-server-linux
Windows: docker pull microsoft/mssql-server-windows
```

Once it's installed, you can run the docker images command to see that Docker is aware of this image:

```
REPOSITORY                    TAG     IMAGE ID      CREATED      SIZE
microsoft/mssql-server-linux  late    7b1c26822d    13 days a    1.35 GB
```

Note that if you already have the base image, you can visit dockr.ly/2qTavYr to see if a newer one has been released. If so, you should pull it again. Next, I'll use the docker run command to spin up this image as a container in order to interact with it:

```
docker run -e 'ACCEPT_EULA=Y' -e 'SA_PASSWORD=PasswOrd' -p 1433:1433 -d
--name juliesqllinux microsoft/mssql-server-linux
```

When starting up an mssql-server-linux container the first time, you're required to accept its license agreement, which you do using an environment variable:

```
-e 'ACCEPT_EULA=Y'
```

You also must include an environment variable for a password to accompany the default sa user. The password you create must consist of "at least 8 characters including uppercase, lowercase letters, base-10 digits and/or non-alphanumeric symbols." Optionally,

DevExpress®

DevExpress Spreadsheet for WPF & WinForms

with Chart, Pivot Table and Cell Editor support

Your users already know Microsoft® Excel® and with the new capabilities we've introduced in our Spreadsheet Control, you can create solutions that utilize a "spreadsheet-first" UX for a broad range of use-case scenarios. Imagine sales entry forms that are an actual invoice or an inventory management document with built-in ordering and data entry options.

Give our Spreadsheet a try today and see how easy it is to build Windows® apps that amaze.

Free 30-day trial

devexpress.com/spreadsheet

#UseTheBest

you can name the container instance. I'll call mine juliesqllinux. I'll also specify the port mapping for the host port to container port with -p, as well as a parameter to run this in the background: -d (for detach). If you don't use the -d parameter, the instance will run in the foreground and you won't get a prompt back at your terminal in order to continue executing commands. As usual, I learned this the hard way.

In response to the run command, Docker returns a unique container ID and returns your terminal back to a prompt. It does this in about one second. Think about how long it takes to install SQL Server onto your computer and get it configured. Then let me repeat: Spinning up the container took about one second.

Interacting Directly with the SQL Server from the Command Line

While my last column focused on using the mssql extension for Visual Studio Code to interact with the database, here I'll use a command-line tool to quickly create a simple database, a table and some data, and then run a query to prove it's all working. Although the Microsoft sqlcmd command-line utility is part of the image, I find it easy enough to use it directly from my OS. Sqlcmd is available for Windows (bit.ly/2qKnrmh) and macOS (bit.ly/2qZBS6G). The macOS version is new. While I'll use it in this article, I'm also a fan of the cross-platform sql-cli (bit.ly/2pYOzey).

When starting up sqlcmd, you need to specify the server name, user name and password as part of the command. Additionally, you can specify a particular database, although the utility will use master by default. See the sqlcmd for Windows link mentioned earlier for details on all of the command-line options.

From the command prompt, I'll start up a sqlcmd command with the required parameters. This returns a numbered prompt so I can use it interactively:

```
→  ~ sqlcmd -S localhost -U sa  -P PassW0rd
1>
```

At the prompt I can start entering lines of TSQL, then a final line, Go, to execute. For example, I'll retrieve a list of databases that already exist on the server:

```
1> select name from sys.databases
2> go
name
------
master
tempdb
model
msdb

(4 rows affected)
1>
```

Now, I'll create a new database and execute that. Then I'll use that new database, create a new table and add some data using the commands in **Figure 1**.

I find it a little clunky to work interactively with sqlcmd. Definitely have a look at the new cross-platform mssql-scripter tool (bit.ly/2pSNhoF) that was recently released as a preview.

The final command, select * from people, is there to perform a simple validation that the table and the data do indeed exist after running the other commands. Note that when you remove a Docker container, it's gone completely and this database and data will disappear, as well. However, it's possible to create separate data volumes that can persist the data files even if you destroy the container that's running your SQL Server instance. I walked through the first baby steps of creating persistent data containers in the blog post I mentioned earlier, so I won't repeat that here.

Create a Custom Image That Creates Its Own Database

What I'm more interested in is the idea of creating an image that includes not only a running instance of SQL Server, but also a pre-created database for development and testing, something developers can grab and use quickly. Of particular interest to testers is the ability to have this image accessible for automated testing, where the server and database can be instantly available for a test run, then destroyed and recreated on the fly as needed.

Consider that when I first created the container from the Docker image, the master database was created for me. If you don't include the -d (detach) parameter in the docker run command, you can see the many steps that were performed on the container as it starts up. Creating the master database is only one of those steps. So what you can do is create your own image based on the base mssql-server-linux image (or any base image), then provide additional commands to perform any steps you need in the dockerfile file that goes along with your image.

I'll demonstrate by creating an image that will duplicate the steps I just ran in the terminal. These were the commands to create the new database and a new table and to insert a few rows of data.

I'll need a folder to house the files I'll be creating for my image. There will be a total of four because I'm going to separate specific tasks into different files to keep things organized:

1. SqlCmdScript.Sql: This file will hold the TSQL script with the commands for creating the new database, table and data.
2. SqlCmdStartup.sh: This is a bash file (like a batch file for Linux). It starts up the sqlcmd command-line tool and, as part of the command, runs the SqlCmdScript.Sql file. Remember that sqlcmd is also part of the base image.
3. Entrypoint.sh: This is another bash file. It lists the non-Docker tasks that need to run, and its first task is to execute the SqlCmdStartup.sh file. Then it will start the SQL Server process.
4. Dockerfile: This file (there's no extension) is the definition of how to build the image and how to run containers from the image.

Figure 1 **Adding Data to a New Table**

```
1> create database juliedb
2> go
1> create table dbo.people (PersonId int Primary Key, Name nvarchar(50))
2> insert into people values (1,'julie')
3> insert into people values (2,'giantpuppy')
4> select * from people
5> go

(1 rows affected)

(1 rows affected)
PersonId    Name
----------- --------------------------------------------------
          1 julie
          2 giantpuppy

(2 rows affected)
1>
```

Here's the file listing for SqlCmdScript.sql, the same commands I used earlier when I was working directly from the command line:

```
create database juliedb;
GO
use juliedb;
create table people (PersonId int Primary Key, Name nvarchar(50));
insert into people values (1,'julie');
insert into people values (2,'giantpuppy');
select * from people
```

Next is the SqlCmdStartup.sh. Again, this is where I start up the sqlcmd utility and tell it to run the script I just listed:

```
#wait for the SQL Server to come up
sleep 20s

#run the setup script to create the DB and the schema in the DB
/opt/mssql-tools/bin/sqlcmd -S localhost -U sa -P PasswOrd -d master -i
SqlCmdScript.sql
```

> I'm using just a simple set of parameters, although there's a lot that you can control when building images.

The next file is entrypoint.sh, where I tell Docker what external processes to run. This is where I trigger the SqlCmdStartup bash script that runs sqlcmd, as well as the bash script inside the base image that starts up the SQL Server database. I combine the two commands using the & character:

```
#start the script to create the DB and data then start the sqlserver
./SqlCmdStartup.sh & /opt/mssql/bin/sqlservr
```

Notice that I'm running sqlcmd first. This is important and I struggled with this for a long time because of a misunderstanding. When Docker encounters a command that then completes, it stops the container. Originally, I ran the server startup first, then the sqlcmd. But when Docker finished the sqlcmd, it decided it had finished its job and shut down. In contrast, when I run the server startup second, the server is just a long-running process, therefore, Docker keeps the container running until something else tells it to stop.

If, like me, you're curious what's in the sqlservr.sh script, take a look at bit.ly/2qJ9mGe, where I blogged about the file listing.

Finally, here's the Dockerfile, which does a number of things:

```
FROM microsoft/mssql-server-linux

ENV SA_PASSWORD=PasswOrd
ENV ACCEPT_EULA=Y

COPY entrypoint.sh entrypoint.sh
COPY SqlCmdStartup.sh SqlCmdStartup.sh
COPY SqlCmdScript.sql SqlCmdScript.sql

RUN chmod +x ./SqlCmdStartup.sh

CMD /bin/bash ./entrypoint.sh
```

It first identifies the base image (mssql-server-linux), which, if not found on your machine, will be automatically pulled from Docker Hub. Then, it sets the environment variables so I don't have to do that in the docker run command. Dockerfile then copies my two bash files and the SQL script file into the image. Next, it runs the chmod command, which permits my bash file to run inside the container. Finally, I instruct Docker what to do at the time that a new container is created from my image by specifying the CMD command to call the entrypoint bash script, which will in turn run sqlcmd with my TSQL and then start up SQL Server.

Build the New Image

With all of this in place, it's time to build my new image with the docker build command:

```
docker build -t julielinuximage .
```

I'm using just a simple set of parameters, although there's a lot that you can control when building images. I'm using only the -t parameter here, which will force a default tag on the image, though you can also specify tags for versioning and other purposes. Then I specify the name of the image and, finally, with the period at the end, I let Docker know that the Dockerfile I want to build from is in the current folder.

At this point, I have the image that I can publish somewhere to share with my team. Doing so requires a registry, but you don't have to store your images on the Docker Hub. You can create a registry on your own network, use a public or private registry on Azure, or choose one of the other myriad options for hosting Docker images. However, for this article, I'll continue to just work locally.

My next step is to run a container from my new image. Remember that because I put my environment variables in the Dockerfile, I don't need to include

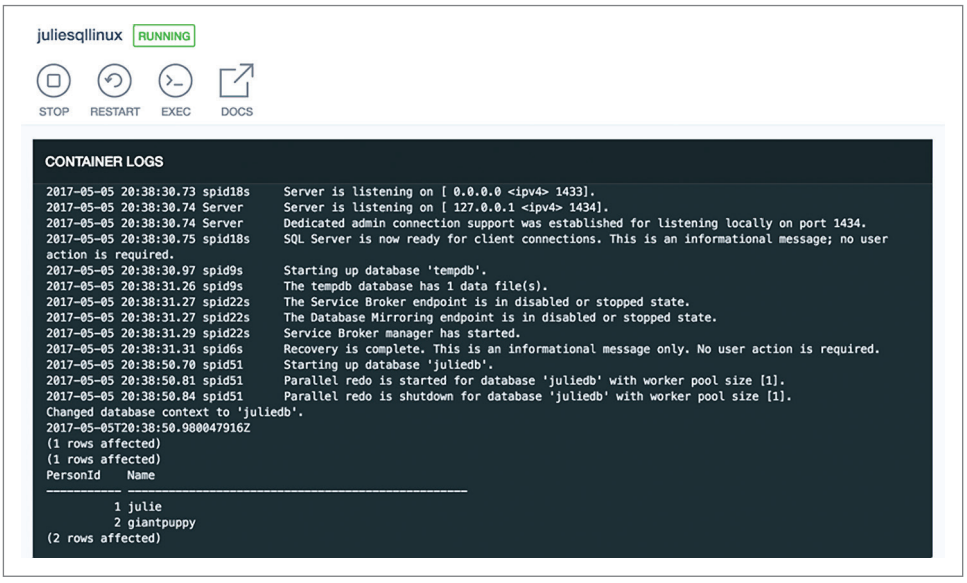


Figure 2 **The Final Lines of the Log Captured in Kitematic After Instantiating the Container with the Docker Run Command**

them in the docker run command. I'll again use the -d parameter so the container will run in the background and I can continue to use the same terminal window to work with the container:

```
docker run -d  -p 1433:1433  --name juliesqllinux julielinuximage
```

This command specifies the port to run this on, the name to apply to the running container (--name juliesqllinux) and the name of the image to use (julielinuximage).

The docker ps command lets me confirm that it's now running. I'm also running the Kitematic UI (included with the Docker client application installer), which shows the state of Docker containers and also displays logs. This means I can see not only all of the tasks accomplished by the sqlservr.sh script, but evidence of my juliedb database, tables and data being inserted. I can even see the results of the query at the end of my sql script to display the data from the people table.

Figure 2 shows the end of the logs after running this container.

Now, just like before, I can interact with the container using the command-line tools installed directly on my computer.

I have to start sqlcmd again, then I call a few commands to check that I have access to the juliedb database my image created. **Figure 3** shows my entire interaction, with my commands in bold and the response in standard font. The commands shown are starting up the sqlcmd utility and connecting to the database server, listing the databases, using juliedb, listing its tables, querying the people table and then quitting out of the sqlcmd utility.

Skip the Script, Just Provide the Database File

If you have a large database (or multiple databases) to share, you might prefer not to include all of the scripts for the schema and data. Another approach is to pre-create the database files, include those mdf files with the image, and in the Dockerfile be sure to copy the files into the container and then run TSQL to attach the files to the server. SQL Server DBA Andrew Pruski wrote a great blog post about this approach, which you can find at bit.ly/2pUxQdP. I've also done this myself in the Pluralsight course I'm currently building about the mssql extension.

Figure 3 **Interacting with Data in the Container That Pre-Created a Database**

```
→ sqlcmd -S localhost -U sa  -P Passw0rd
1> select name from sys.databases
2> go
name
--------------------
master
tempdb
model
msdb
juliedb

(5 rows affected)
1> use juliedb
2> go
Changed database context to 'juliedb'.
1> select name from sys.tables
2> go
name
-----------------
people

(1 rows affected)
1> select * from people
2> go
PersonId    Name
----------- ----------------
          1 julie
          2 giantpuppy

(2 rows affected)
1> quit
→
```

> There's certainly a lot more to learn and benefit from providing SQL Server in Linux and Windows containers.

For Devs or DevOps

With these two approaches, you now have a simple means of sharing a pre-configured database along with a SQL Server across your team, allowing everyone to have a locally running server and a local copy of the database. But thanks to the container, nobody needs to install SQL Server on their computer or execute any scripts to get the necessary database set up for use. It's frighteningly easy and fast.

I also mentioned using this for DevOps scenarios, perhaps with automated tests that require a database. I hope it's not a big leap from what you've seen here to imagine having your build or test process spin up a container that has everything needed for your operation to quickly and easily interact with a SQL Server database. One example of this is in the MSSQL-Node-Docker demo app (bit.ly/2qT6RgK) created by Travis Wright, a program manager on the SQL Server engineering team. The demo was "created to show how SQL Server can operate in a DevOps scenario where an application developer can check in code to GitHub and then trigger a build in Red Hat OpenShift to deploy the changes automatically as pods (containers)."

There's certainly a lot more to learn and benefit from providing SQL Server in Linux and Windows containers. It's amazing to me that you can now "install" SQL Server almost anywhere and you don't have to be a Microsoft platform developer to benefit from what is one of the most powerful relational databases on the planet. As somewhat of a newbie to Docker, I was curious about the ability to create an image that included my own database. And I'm always happy to share the results of my explorations. I learned a lot and hope that your curiosity isn't just sated, but piqued enough to explore the possibilities even further. ■

Julie Lerman *is a Microsoft Regional Director, Microsoft MVP, software team mentor and consultant who lives in the hills of Vermont. You can find her presenting on data access and other topics at user groups and conferences around the world. She blogs at thedatafarm.com/blog and is the author of "Programming Entity Framework," as well as a Code First and a DbContext edition, all from O'Reilly Media. Follow her on Twitter: @julielerman and see her Pluralsight courses at juliel.me/PS-Videos.*

Thanks to the following Microsoft technical experts for reviewing this article: *Sanjay Nagamangalam and Travis Wright*

File Format APIs

Working with Files?

CREATE CONVERT PRINT

MODIFY COMBINE

FREE TRIAL

Aspose.Total

Manipulate Word, Excel, PDF, PowerPoint, Outlook and more than 100 other file formats in your applications without installing Microsoft Office.

DOC, XLS, PDF, PPT, MSG, BMP, PNG, XML and many more!

Platforms supported: .NET, Java, Cloud, Android, SharePoint, Reporting Services, and JasperReports

ASPOSE
File Format APIs

CONTACT US

US: +1 903 306 1676
EU: +44 141 628 8900
AU: +61 2 8006 6987

sales@asposeptyltd.com

Try for FREE at
www.aspose.com

Cognition at Scale with U-SQL on ADLA

Hiren Patel and Shravan Matthur Narayanamurthy

Companies providing cloud-scale services have an ever-growing need to store and analyze massive data sets. Such analysis to transform data into insight is becoming increasingly valuable. For instance, analyzing telemetry from a service to derive insights into the investments that most improve service quality, analyzing usage patterns over time to detect changes in user behavior (engagement/churn), analyzing sensor data to perform preventative maintenance—all these are extremely important, as has become very apparent to us while running massive services like Bing and Cosmos. Most of these analyses involve feature engineering and modeling. With storage getting cheaper, there are no longer any constraints on the amount of data we can collect, which means we soon reach the limits of traditional single-node data processing engines and require a distributed processing platform to do machine learning tasks on massive datasets. Furthermore, machine learning models are usually built or used in applications or pipelines that involve processing raw data—deserializing data, filtering out unnecessary rows and columns, extracting features, and transforming them to a form amenable for modeling. To express such operations easily, users need a programming model that offers a degree of compositional freedom that's typical of declarative languages.

This article discusses:

- Machine learning using U-SQL in ADLA
- Cognition with U-SQL
- Using a pre-trained model

Technologies discussed:

U-SQL, Azure Data Lake Analytics, U-SQL Extension-Cognitive/R/Python

U-SQL is a declarative language that provides the expressibility necessary for advanced analytics tasks, like machine learning and operating seamlessly on cloud-scale data. It also offers the following advantages:

- The resemblance of U-SQL to SQL reduces the learning curve for users. It offers easy extensibility with user-defined operators, the ability to reuse existing libraries and the flexibility to choose different languages (C#, Python or R) to develop custom algorithms.
- Users can focus on business logic while the system takes care of data distribution and task parallelism, along with execution plan complexities.
- U-SQL has built-in support for machine learning.

Machine Learning Using U-SQL in ADLA

Building intelligent features into applications requires some form of prediction capability. There are two ways to go:

Build your own model: You first preprocess the telemetry or any kind of raw data into a shape suitable for modeling, then train a model on the pre-processed data and use this trained model for prediction in applications. The Azure Data Lake Analytics (ADLA) engine makes all the preprocessing possible in an efficient manner. It allows you to build machine learning models that cover a wide variety of scenarios, from building regression models to image classification via R and Python extensions, and enables you to build models using built-in, efficient, massively parallelable distributed machine learning algorithms. (We'll discuss how to train a model using U-SQL in a future article.)

Using a pre-trained model for scoring: Suppose you have a pre-trained model but want to score large amounts of data efficiently. U-SQL can handle this pleasingly parallel task very well. U-SQL allows user-defined operators (UDOs) where you provide

only a per-partition or per-row function and the system takes care of distributing the data and the tasks in the most efficient manner within given resource constraints.

Training a good model requires a lot of data and machine learning expertise—both of which are rare commodities. To help, U-SQL packages many of the machine learning models that power the Microsoft Cognitive Services. Built by some of the leading minds in the industry, these models are trained against massive data sets, and are highly performant and accurate. This integration of the cognitive models in U-SQL lets you easily add intelligent features—such as emotion detection, face and speech recognition; language understanding and sentiment analysis—to applications that work on massive amounts of data.

In this article, we'll concentrate on how to use these pre-trained models to build intelligent applications using U-SQL, using either a pre-trained model or a built-in cognitive model.

Cognition with U-SQL

U-SQL provides built-in support for the following cognitive models, allowing you to build applications with powerful algorithms using just a few lines of code:

Face detects one or more human faces in an image, along with face attributes that contain machine learning-based predictions based on features such as age, emotion and gender.

Emotion analyzes facial expressions in an image to detect a range of emotions, currently including anger, contempt, disgust, fear, happiness, neutrality, sadness and surprise.

Image tagging returns information about visual content found in an image. It can be used along with descriptions and domain-specific models to identify content in an image.

Optical character recognition (OCR) detects and extracts handwritten text from images of notes, letters, whiteboards and so forth, and returns a machine-readable character stream.

Sentiment analysis detects sentiment using classification techniques based on the input text.

Key phrases extraction identifies key phrases, topics and language from the input text.

Landmark detection finds landmarks in an image. This model recognizes 9,000 natural and man-made landmarks from around the world.

Cognition at Scale with U-SQL

Suppose you want to find out if the human population, in general, is happy when there are animals around them. One way to do this is to examine pictures posted by people who have animals with them and analyze the emotion of those people. In Microsoft, the set of Bing-crawled images would represent a valid dataset for this task, but the sheer scale of the data set would make this simple task cumbersome without an intelligent data-processing engine. Let's see how U-SQL makes this easy. First, however, you'll have to manually enable cognitive capabilities in your Azure Data Lake account.

Registering Cognitive Capabilities in U-SQL To get started with the Python, R, and Cognitive extensions, open your Data Lake Analytics account in the Azure Portal and click on Sample Scripts.

If you haven't installed them already, you'll see a notification at the top of the Sample Scripts blade for U-SQL Advanced Analytics.

Figure 1 **Cognition at Scale Example**

```
REFERENCE ASSEMBLY ImageCommon;
REFERENCE ASSEMBLY ImageEmotion;
REFERENCE ASSEMBLY ImageTagging;

// Load in images
@imgs =
  EXTRACT FileName string, ImgData byte[]
  FROM @"/usqlext/samples/cognition/{FileName:*}.jpg"
  USING new Cognition.Vision.ImageExtractor();

// Extract the number of objects and tags from each image
@objects =
  PROCESS @imgs
    PRODUCE FileName,
      NumObjects int,
      Tags string
    READONLY FileName
    USING new Cognition.Vision.ImageTagger();

// Extract all the images with dog and cat
@pets =
  SELECT FileName, T.Tag
    FROM @objects
    CROSS APPLY
    EXPLODE(SqlArray.Create(Tags.Split(';'))) AS T(Tag)
    WHERE T.Tag.ToString().Contains("dog") OR T.Tag.ToString().Contains("cat");

// Detect emotions from human face
@emotions =
  PROCESS @imgs
    PRODUCE FileName string,
      NumFaces int,
      Emotion string
    READONLY FileName
    USING new Cognition.Vision.EmotionAnalyzer();

// Correlation to find all the images which has both human and animals
@preres =
  SELECT @pets.FileName, Emotion
    FROM @pets
    JOIN @emotions
    ON @pets.FileName == @emotions.FileName;

// Distribution of human emotions when animals are in the image
@result =
  SELECT Emotion, COUNT(FileName) AS frequency
    FROM @preres
    GROUP BY Emotion;

OUTPUT @result
  TO @"/my/cognition/output/sample_dog_cat.csv"
  USING Outputters.Csv();
```

Click it to begin the installation process.

Once you've chosen to install the extensions, the system will copy U-SQL extension-related files into the default Azure Data Lake Store (ADLS) associated with your ADLA account. A notification that files are being copied will appear near the Notification icon in the upper right of the page. When the files are copied, you'll see an updated notification that the file copying was successful and that a special U-SQL Job was submitted to finish the registration. You can find the special job and its status by using View All Jobs in the upper-left corner of the Azure Portal. The Job normally will take a few minutes to complete.

At that point, you can discover what the job did by browsing the catalog for the master database. The job simply registers advanced analytic cognitive assemblies in the master database, which you can see by using the Data Explorer.

Seeing these assemblies in the master database is evidence that your ADLA account is setup correctly and you can now write a U-SQL script that uses cognitive functions to build intelligent applications.

Using Cognitive Functions in U-SQL As described earlier, the assemblies and models that power the Cognitive Services have been integrated with U-SQL, allowing you to run simple query statements over millions of images and process them using cognitive functions. The overall method for using these cognitive capabilities at scale in U-SQL is simply:

- Use the REFERENCE ASSEMBLY statement to include the cognitive functions in the U-SQL script.
- Use the EXTRACT operation to load data into a rowset.
- Use the PROCESS operation to apply various Cognitive functions.
- Use the SELECT operation to apply transformations to the predictions.
- Use the OUTPUT operation to store the result into persistent store.

Let's continue with the scenario described earlier, which involves processing a large number of images and analyzing the emotion of people when there are animals in the image. **Figure 1** shows an example script for completing this scenario. In the example, we use the Vision cognitive functions, which enables us to understand what's in an image and returns a set of tags that identify objects. For the sample script in **Figure 1**, we're using a subset of the images from the team's 1 million images dataset.

In this simple U-SQL query, we're doing some very powerful things. First, we're extracting images into the byte array column using the system-provided ImageExtractor, and then loading them into rowsets. Next, we extract all the tags from those images using the built-in ImageTagger. Then we filter the images, finding those that have "cat" or "dog" tags. Using the system-provided EmotionAnalyzer, we next extract the faces and associated emotions from these images, then find all the images that have a human along with a dog or a cat. Finally, we output the distribution of human emotions in those images.

To demonstrate the scalability of U-SQL, we executed the same script on the full data set with 1 million images. As soon as we submit the script, in a matter of seconds, thousands of containers in ADLA spring to action to start processing these images, as shown in the **Figure 2**.

You can easily extend this example to get other interesting insights, like the most frequently occurring pairs of tags, objects that appear together most often and so on. Furthermore, you can also detect age, gender, and landmarks from these images using other cognitive functions. For your reference, we've added the code snippets in **Figure 3** to describe how to use other built-in cognitive functions in U-SQL applications.

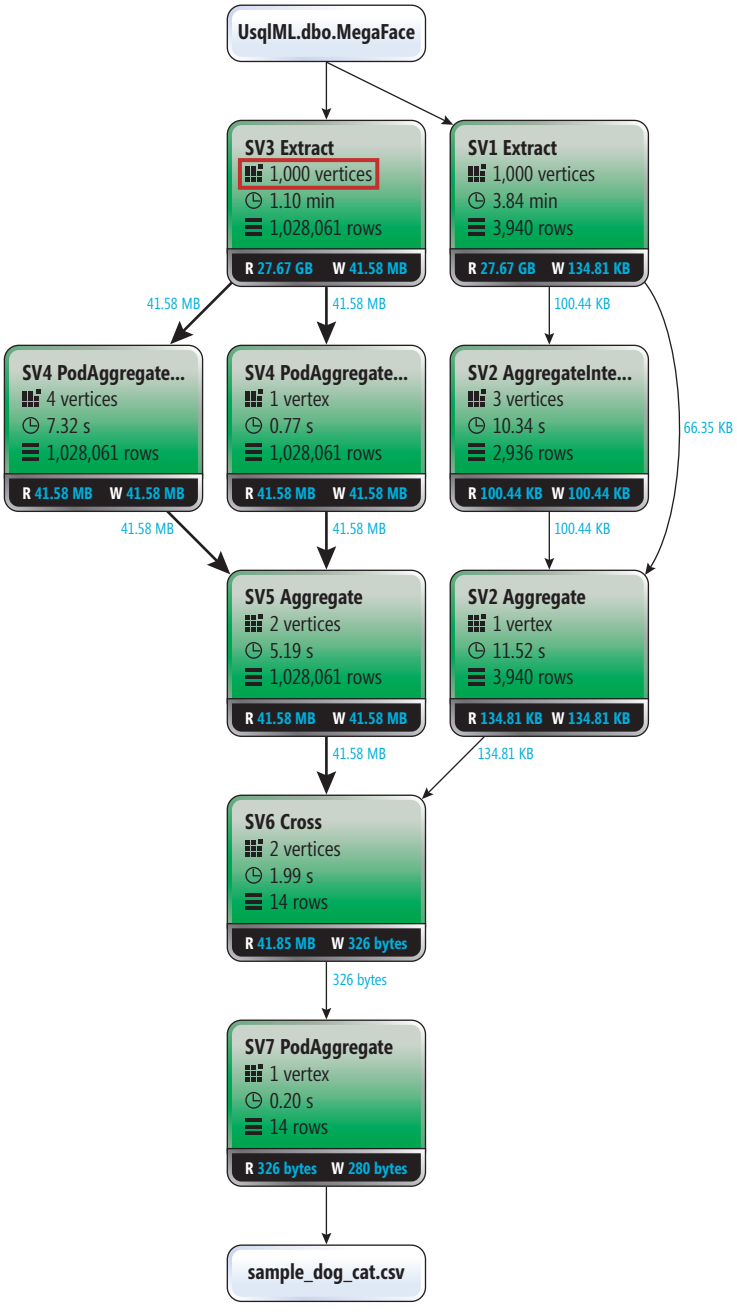


Figure 2 **Job Execution Graph**

Using a Pre-Trained Model

Most traditional machine learning algorithms assume that the data processed to train a model isn't too large to store in the RAM of one computer. Thus, most of the time users need only a single-box environment to train their models. Furthermore, it's relatively common to have only a small amount of label data on which a model is trained. The R and Python languages have emerged as the industry standard for open source, as well as proprietary predictive analytics. R and Python together provide many capabilities, such as flexibility, rich graphics and statistics-oriented features, along with an ecosystem of freely available packages that account for much of its growing popularity. Thus, many developers uses R and Python to do single-box predictive analytics.

Once trained, a model is applied to massive data sets that frequently eclipse the size of the training data by orders of magnitude. In the following section, we'll describe how to use an existing model that was trained to do prediction using a local R/Python environment on a massive amount of data, using the U-SQL extension on ADLA.

Consider an arboretum that has an inventory of many species of flowers from all around the world. Now the organization wants to find and classify types of iris flowers in its inventory. The arboretum's data scientist trained a model to label types of iris flowers using R on a single machine, but they have a great many species of flowers from all over the world and the pre-trained model can't complete this simple task of identifying an iris flower. What's needed is an intelligent, scalable data processing engine. The overall process to use these U-SQL R extensions to do prediction at scale is simply:

- Use the REFERENCE ASSEMBLY statement to include the R U-SQL extension to run R code in the U-SQL script.
- Use the DEPLOY RESOURCE operation to upload the pre-trained model as a resource on executing nodes.
- Use DECLARE to inline the R script in the U-SQL script.
- Use the EXTRACT operation to load data into a rowset.
- Use the Extension.R.Reduce function to run the R script to score each row in rowset using the uploaded pre-trained model.
- Use the OUTPUT operation to store the result into a persistent store.

Figure 4 shows the U-SQL script that carries out this process.

In this simple U-SQL query, we're using the U-SQL R extension to do scoring at scale. The R and Python U-SQL extensions get automatically installed and registered with ADLA account database when you install the U-SQL Advance Analytics Extension. In the U-SQL script, we first deploy the pre-existing model, which was trained using R on a single machine. This highlights the fact that it wasn't trained using ADLA/U-SQL framework. Next, we extract and de-serialize the iris dataset into columns using the system-provided .csv format extractor, Extractors.cvs, and load the data into rowsets. Next, we generate a random number that will be used later to partition data to enable parallel processing. Then, we use the U-SQL R extension UDO Extension.R.Reducer and pass the R script that does the prediction, along with the model. Finally, we output the confidence interval for each flower from the inventory.

We started with a simple U-SQL script to understand the content of images, which is typically considered opaque. The script automatically scales across hundreds of machines to transform images efficiently into actionable insights that can power intelligent applications. We also showcase how you can reuse an existing model that was trained using the popular R/Python environment and apply the model to do prediction on a massive amount of data using U-SQL R Extension. This is what can power the intelligence revolution. ■

Figure 3 **Code Snippets for Other Cognitive APIs Supported in U-SQL**

```
// Estimate age and gender for human faces
@faces =
  PROCESS @imgs
  PRODUCE FileName,
    NumFaces int,
    FaceAge string,
    FaceGender string
    READONLY FileName
  USING new Cognition.Vision.FaceDetector();

// Apply OCR
@ocrs =
  PROCESS @imgs
  PRODUCE FileName,
    Text string
  READONLY FileName
  USING new Cognition.Vision.OcrExtractor();

// Sentiment Analysis on War and Peace
@sentiment =
  PROCESS @WarAndPeace
  PRODUCE No, Year, Book, Chapter,
    Text, Sentiment string,
          Conf double
  READONLY No,
    Year,
    Book,
    Chapter,
    Text
  USING new Cognition.Text.SentimentAnalyzer(true)
```

Figure 4 **Using Pre-Existing Model in U-SQL Script**

```
REFERENCE ASSEMBLY [ExtR];

DEPLOY RESOURCE @"/usqlext/samples/R/my_model_LM_Iris.rda";

// R script to score using pre trained R model
DECLARE @MyRScript =
  @"
    load(""my_model_LM_Iris.rda"")
    outputToUSQL=data.frame(predict(lm.fit, inputFromUSQL, interval=""confidence""))
  ";

DECLARE @PartitionCount int = 10;


@InputData =
  EXTRACT SepalLength double,
    SepalWidth double,
    PetalLength double,
    PetalWidth double,
    Species string
  FROM @"/usqlext/samples/R/iris.csv";
  USING Extractors.Csv();

@ExtendedData =
  SELECT Extension.R.RandomNumberGenerator.GetRandomNumber(@PartitionCount) AS Par,
    SepalLength,
    SepalWidth,
    PetalLength,
    PetalWidth
  FROM @InputData;

// Predict Species
@RScriptOutput= REDUCE @ExtendedData
  ON Par
  PRODUCE Par,
    fit double,
      lwr double,
      upr double
  READONLY Par
  USING
    new Extension.R.Reducer(command:@MyRScript , rReturnType:"dataframe",
      stringsAsFactors:false);

OUTPUT @RScriptOutput
  TO @"/Output/LMPredictionsIris.txt"
  USING Outputters.Tsv();
```

Hiren Patel is *a senior technical program manager at Microsoft. He has been part of the Big Data group since 2011 and worked on designing and developing various aspect of the Cosmos/ADLA distributed execution engine, including language, optimizer, runtime and scheduling.*

Shravan Matthur Narayanamurthy *is a senior engineering manager at Microsoft leading the Big Data Machine Learning team. He has several years of experience researching and developing machine learning algorithms that operate at cloud scale and distributed systems.*

Thanks to the following Microsoft technical experts who reviewed this article: *Saveen Reddy and Michael Rys*

Introduction to the Microsoft CNTK v2.0 Library

James McCaffrey

The Microsoft Cognitive Toolkit (CNTK) is a powerful, open source library that can be used to create machine learning prediction models. In particular, CNTK can create deep neural networks, which are at the forefront of artificial intelligence efforts such as Cortana and self-driving automobiles.

CNTK version 2.0 is much, much different from version 1. At the time I'm writing this article, version 2.0 is in Release Candidate mode. By the time you read this, there will likely be some minor changes to the code base, but I'm confident they won't affect the demo code presented here very much.

In this article, I'll explain how to install CNTK v2.0, and how to create, train and make predictions with a simple neural network. A good way to see where this article is headed is to take a look at the screenshot in **Figure 1**.

Disclaimer: CNTK version 2.0 is in Release Candidate mode. All information is subject to change.

This article discusses:

- Installing CNTK v2.0
- Understanding neural networks
- The structure of the demo program
- Creating, training and testing a neural network
- Measuring error and accuracy
- Making predictions

Technologies discussed:

Microsoft Cognitive Toolkit, Python, NumPy

Code download available at:

msdn.com/magazine/0717magcode

The CNTK library is written in C++ for performance reasons, but v2.0 has a new Python language API, which is now the preferred way to use the library. I invoke the iris_demo.py program by typing the following in an ordinary Windows 10 command shell:

```
> python iris_demo.py 2>nul
```

The second argument suppresses error messages. I do this only to avoid displaying the approximately 12 lines of CNTK build information that would otherwise be shown.

The goal of the demo program is to create a neural network that can predict the species of an iris flower, using the well-known Iris Data Set. The raw data items look like this:

```
5.0 3.5 1.3 0.3 setosa
5.5 2.6 4.4 1.2 versicolor
6.7 3.1 5.6 2.4 virginica
```

There are 150 data items, 50 of each of three species: setosa, versicolor and virginica. The first four values on each line are the predictor values, often called attributes or features. The item-to-predict is often called the class or the label. The first two feature values are a flower's sepal length and width (a sepal is a leaf-like structure). The next two values are the petal length and width.

Neural networks work only with numeric values, so the data files used by the demo encode species as setosa = (1,0,0), versicolor = (0,1,0) and virginica = (0,0,1).

The demo program creates a 4-2-3 neural network; that is, a network with four input nodes for the feature values, two hidden processing nodes and three output nodes for the label values. The number of input and output nodes for a neural network classifier are determined by the structure of your data, but the number of hidden processing nodes is a free parameter and must be determined by trial and error.

You can think of a neural network as a complex mathematical prediction equation. Neural network training is the process of

determining the constants that define the equation. Training is an iterative process and the demo performs 5,000 training iterations, using 120 of the 150 iris data items.

After training, the prediction model is applied to the 30 iris data items that were held out of the training process. The model had a classification error of 0.0667, which means that the model incorrectly predicted the species of 0.0667 * 30 = 2 flower items and, therefore, correctly predicted 28 items. The classification error on a holdout test set is a very rough estimate of how well you'd expect the model to do when presented with a set of new, previously unseen data items.

Next, the demo program uses the trained neural network to predict the species of a flower with features (6.9, 3.1, 4.6, 1.3). The prediction is computed and displayed in terms of probabilities: (0.263, 0.682, 0.055). Notice that the three values sum to 1.0. Because the middle value, 0.682, is the largest, the prediction maps to (0,1,0), which in turn maps to versicolor.

The remainder of the output shown in **Figure 1** displays the values of the constants that define the neural network prediction

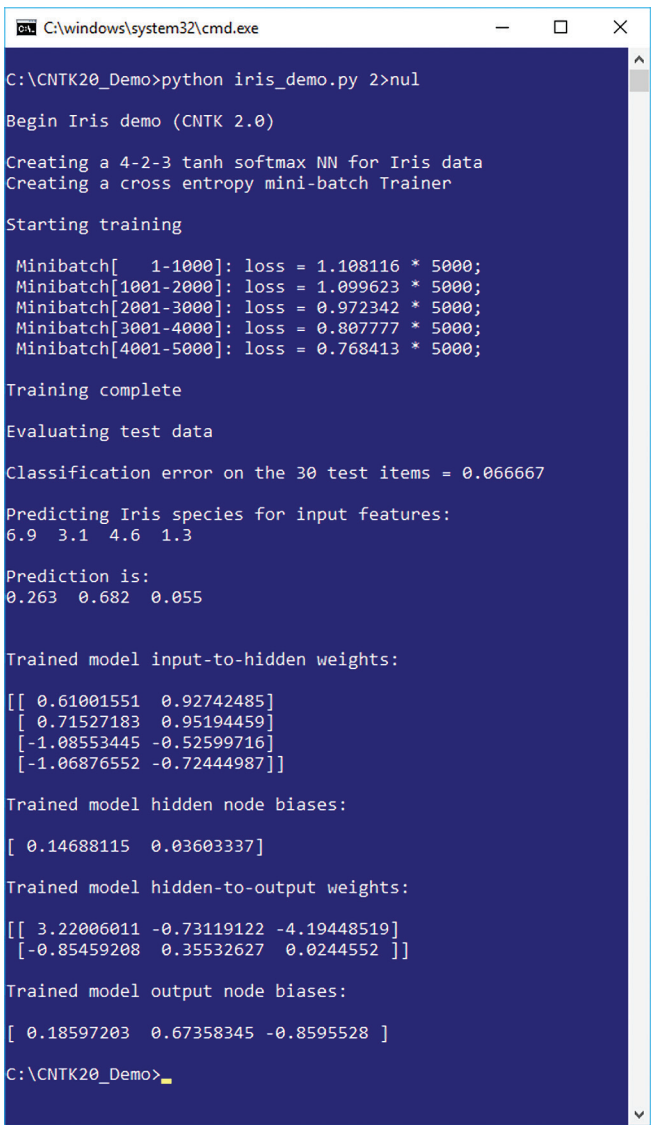


Figure 1 **CNTK v2.0 in Action**

model. I'll explain where those values come from, and what they can be used for, shortly.

This article makes no particular assumptions about your knowledge of neural networks, or CNTK or Python. Regardless of your background, you should be able to follow along without too much trouble. The complete source code for the iris_demo.py program is presented in this article, and is also available in the accompanying download.

Installing CNTK v2.0

There are several ways to install CNTK, but I'll describe the simplest approach. The first step is to install a CNTK-compatible version of Anaconda onto your Windows machine.

At the time I wrote this article, CNTK v2.0 RC1 required Anaconda (with Python 3), version 4.1.1, 64-bit, which contains Python version 3.5.2 and NumPy 1.11.1. So I went to the Anaconda Download site (which you can easily find with an Internet search), then to the archives page and found a self-extracting executable installer file named Anaconda3-4.1.1-Windows-x86_64.exe and double-clicked on it.

The CNTK library and documentation is hosted on GitHub at github.com/Microsoft/CNTK. I strongly advise you to review the current CNTK system requirements, especially the version of Anaconda, before trying to install CNTK.

The Anaconda install process is very slick and I accepted all the default installation options. You might want to take note of the Anaconda installation location because CNTK will go there, too.

By far the easiest way to install CNTK is indirectly, by using the Python pip utility program. In other words, you don't need to go to the CNTK site to install it, though you do need to go to the CNTK installation directions to determine the correct installation URL. In my case that URL was:

https://cntk.ai.PythonWheel/CPU-Only/cntk-2.0rc1-cp35-cp35m-win_amd64.whl

The URL you'll want to use will definitely be different by the time you read this article. If you're new to Python, you can think of a .WHL file (pronounced "wheel") as somewhat similar to a Windows .MSI installer file. Notice the CPU-Only part of the URL. If you have a machine with a supported GPU, you can use it with a dual CPU-GPU version of CNTK.

Once you determine the correct URL, all you have to do is launch an ordinary Windows command shell and type:

```
> pip install <url>
```

Installation is very quick, and files are placed in the Anaconda directory tree. Any install errors will be immediately and painfully obvious, but you can check a successful installation by typing the following at a command prompt and you should see the CNTK version displayed:

```
> python -c "import cntk; print(cntk.__version__)"
```

Understanding Neural Networks

CNTK operates at a relatively low level. To understand how to use CNTK to create a neural network prediction model, you have to understand the basic mechanics of neural networks. The diagram in **Figure 2** corresponds to the demo program.

The network input layer has four nodes and holds the sepal length and width (6.9, 3.1) and the petal length and width (4.6, 1.3) of a

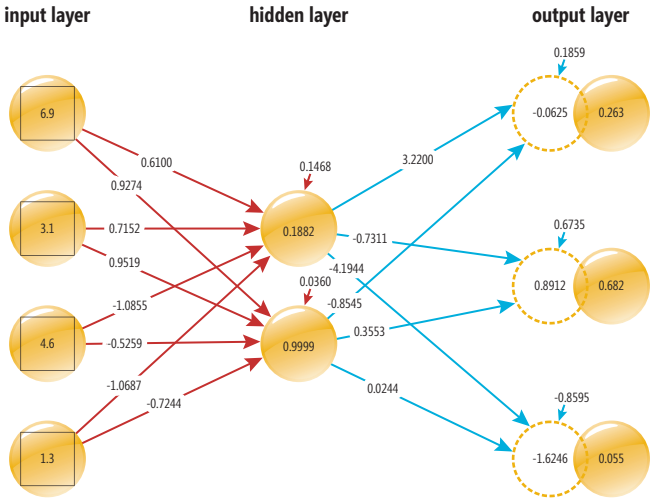


Figure 2 **Neural Network Input-Output Mechanism**

flower of an unknown species. The eight arrows connecting each of the four input nodes to the two hidden processing nodes represent numeric constants called weights. If nodes are 0-base indexed with [0] at the top, then the input-to-hidden weight from input[0] to hidden[0] is 0.6100 and so on.

Similarly, the six arrows connecting the two hidden nodes to the three output nodes are hidden-to-output weights. The two small arrows pointing into the two hidden nodes are special weights called biases. Similarly, the three output nodes each have a bias value.

The first step in the neural network input-output mechanism is to compute the values of the hidden nodes. The value in each hidden node is the hyperbolic tangent of the sum of products of input values and associated weights, plus the bias. For example:

```
hidden[0] = tanh( (6.9)(0.6100) +
                  (3.1)(0.7152) +
                  (4.6)(-1.0855) +
                  (1.3)(-1.0687) + 0.1468 )

          = tanh(0.1903)
          = 0.1882
```

The value of the hidden[1] node is calculated in the same way. The hyperbolic tangent function, abbreviated tanh, is called the hidden layer activation function. The tanh function accepts any value, from negative infinity to positive infinity, and returns a value between -1.0 and +1.0. There are several choices of activation functions supported by CNTK. The three most common are tanh, logistic sigmoid and rectified linear unit (ReLU).

Figure 3 **Demo Program Structure**

```
# iris_demo.py
import cntk as C
...
def my_print(arr, dec):
def create_reader(path, is_training, input_dim,
  output_dim):
def save_weights(fn, ihWeights, hBiases,
  hoWeights, oBiases):
def do_demo():
def main():
  print("\nBegin Iris demo (CNTK 2.0) \n")
  np.random.seed(0)
  do_demo()  # all the work is done in do_demo()
if __name__ == "__main__":
  main()
```

Computing the output node values is similar to the process used to compute hidden nodes, but a different activation function, called softmax, is used. The first step is to compute the sum of products plus bias for all three output nodes:

```
pre-output[0] = (0.1882)(3.2200) + (0.9999)(-0.8545) + 0.1859
              = -0.0625

pre-output[1] = (0.1882)(-0.7311) + (0.9999)(0.3553) + 0.6735
              = 0.8912

pre-output[2] = (0.1882)(-4.1944) + (0.9999)(0.0244) + (-0.8595)
              = -1.6246
```

The softmax value of one of a set of three values is the exp function applied to the value, divided by the sum of the exp function applied to all three values. So the final output node values are computed as:

```
output[0] = exp(-0.0625) / exp(-0.0625) + exp(0.8912) + exp(-1.6246)
          = 0.263

output[1] = exp(0.8912) / exp(-0.0625) + exp(0.8912) + exp(-1.6246)
          = 0.682

output[2] = exp(-1.6246) / exp(-0.0625) + exp(0.8912) + exp(-1.6246)
          = 0.055
```

The purpose of softmax is to coerce the preliminary output values so they sum to 1.0 and can be interpreted as probabilities.

OK, but where do the values of the weights and biases come from? To get the values of the weights and biases, you must train the network using a set of data that has known input values and known, correct, output values. The idea is to use an optimization algorithm that finds the values for the weights and biases that minimizes the difference between the computed output values and the correct output values.

Demo Program Structure

The overall structure of the demo program is shown in **Figure 3**.

The demo program has a function named main that acts as an entry point. The main function sets the seed of the global random number generator to 0 so that results will be reproducible, and then calls function do_demo that does all the work.

Helper function my_print displays a numeric vector using a specified number of decimals. The point here is that CNTK is just a library, and you must mix program-defined Python code with calls to the various CNTK functions. Helper function create_reader returns a special CNTK object that can be used to read data from a data file that uses the special CTF (CNTK text format) formatting protocol.

Helper function save_weights accepts a filename, a matrix of input-to-hidden weights, an array of hidden node biases, a matrix of hidden-to-output weights, and an array of output node biases, and writes those values to a text file so they can be used by other systems.

The complete listing for the demo program, with a few minor edits, is presented in **Figure 4**. I use an indent of two-space characters instead of the more common four, to save space. Also, all normal error-checking code has been removed.

The demo program begins by importing the required Python packages and modules. I'll describe the modules as they're used in the demo code.

Setting Up the Data

There are two basic ways to read data for use by CNTK functions. You can format your files using the special CTF format and then use

BEST SELLER

Help & Manual Professional | from $586.04

ec software

Help and documentation for .NET and mobile applications.

- Powerful features in an easy, accessible and intuitive user interface
- As easy to use as a word processor, but with all the power of a true WYSIWYG XML editor
- Single source, multi-channel publishing with conditional and customized output features
- Output to responsive HTML, CHM, PDF, MS Word, ePUB, Kindle or print
- Styles and Templates give you full design control

BEST SELLER

Aspose.Total for .NET | from $2,939.02

ASPOSE File Format APIs

Every Aspose .NET API in one package.

- Programmatically manage popular file formats including Word, Excel, PowerPoint and PDF
- Work with charts, diagrams, images, project plans, emails, barcodes, OCR, Visio, OneNote, 3D and CAD files alongside many more document management features in your .NET applications
- Common uses also include mail merging, adding barcodes to documents, building dynamic reports on the fly and extracting text from most document types

BEST SELLER

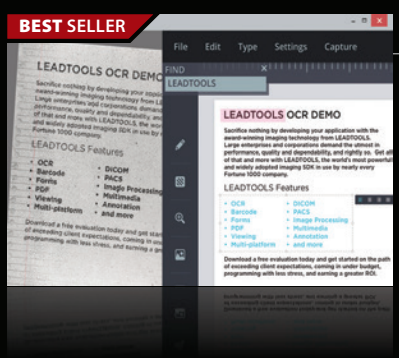
DevExpress DXperience 17.1 | from $1,439.99

DevExpress

The complete range of DevExpress .NET controls and libraries for all major Microsoft platforms.

- WinForms - New TreeMap control, Chart series types and Unbound Data Source
- WPF - New Wizard control and Data Grid scrollbar annotations
- ASP.NET - New Vertical Grid control, additional Themes, Rich Editor Spell Checking and more
- Windows 10 Apps - New Hamburger Sub Menus, Splash Screen and Context Toolbar controls
- CodeRush - New debug visualizer expression map and code analysis diagnostics

BEST SELLER



LEADTOOLS Document Imaging SDKs V19 | from $2,995.00 SRP

LEAD TECHNOLOGIES

Add powerful document imaging functionality to desktop, tablet, mobile & web applications.

- Universal document viewer & conversion framework for PDF, Office, CAD, TIFF & more
- OCR, MICR, OMR, ICR and Forms Recognition supporting structured & unstructured forms
- PDF SDK with text edit, hyperlinks, bookmarks, digital signature, forms, metadata
- Barcode Detect, Read, Write for UPC, EAN, Code 128, Data Matrix, QR Code, PDF417
- Zero-footprint HTML5/JavaScript UI Controls & Web Services

We accept purchase orders.
Contact us to apply for a credit account.

US Headquarters
ComponentSource
650 Claremore Prof Way
Suite 100
Woodstock
GA 30188-5188
USA

European Headquarters
ComponentSource
2 New Century Place
East Street
Reading, Berkshire
RG1 4ET
United Kingdom

Asia / Pacific Headquarters
ComponentSource
7F Kojimachi Ichihara Bldg
1-1-8 Hirakawa-cho
Chiyoda-ku
Tokyo, 102-0093
Japan

Sales Hotline - US & Canada:
(888) 850-9911
www.componentsource.com

GSA Schedule Contract GS-35F-0188R

Figure 4 **Complete Demo Program**

```
# iris_demo.py
# Anaconda 4.1.1 (Python 3.5, NumPy 1.11.1)
# CNTK 2.0 RC1

# Use a one-hidden layer simple NN with 2 hidden nodes
# to classify the Iris Dataset.
# This version uses the built-in Reader functions and
# data files that use the CTF format.
# trainData_cntk.txt - 120 items (40 each class)
# testData_cntk.txt - remaining 30 items

import numpy as np
import cntk as C
from cntk import Trainer  # to train the NN
from cntk.learners import sgd, learning_rate_schedule, \
  UnitType
from cntk.ops import *  # input_variable() def
from cntk.logging import ProgressPrinter
from cntk.initializer import glorot_uniform
from cntk.layers import default_options, Dense
from cntk.io import CTFDeserializer, MinibatchSource, \
  StreamDef, StreamDefs, INFINITELY_REPEAT

# =====

def my_print(arr, dec):
  # print an array of float/double with dec decimals
  fmt = "%." + str(dec) + "f" # like %.4f
  for i in range(0, len(arr)):
    print(fmt % arr[i] + '  ', end='')
  print("\n")

def create_reader(path, is_training, input_dim, output_dim):
  return MinibatchSource(CTFDeserializer(path, StreamDefs(
    features = StreamDef(field='attribs', shape=input_dim,
      is_sparse=False),
    labels = StreamDef(field='species', shape=output_dim,
      is_sparse=False)
  )), randomize = is_training,
    max_sweeps = INFINITELY_REPEAT if is_training else 1)

def save_weights(fn, ihWeights, hBiases,
  hoWeights, oBiases):
  f = open(fn, 'w')
  for vals in ihWeights:
    for v in vals:
      f.write("%s\n" % v)
  for v in hBiases:
    f.write("%s\n" % v)
  for vals in hoWeights:
    for v in vals:
      f.write("%s\n" % v)
  for v in oBiases:
    f.write("%s\n" % v)
  f.close()

def do_demo():
  # create NN, train, test, predict
  input_dim = 4
  hidden_dim = 2
  output_dim = 3

  train_file = "trainData_cntk.txt"
  test_file = "testData_cntk.txt"

  input_Var = C.ops.input(input_dim, np.float32)
  label_Var = C.ops.input(output_dim, np.float32)

  print("Creating a 4-2-3 tanh softmax NN for Iris data ")
  with default_options(init = glorot_uniform()):
    hLayer = Dense(hidden_dim, activation=C.ops.tanh,
      name='hidLayer')(input_Var)
    oLayer = Dense(output_dim, activation=C.ops.softmax,
      name='outLayer')(hLayer)
  nnet = oLayer

  # ---------------------------------

  print("Creating a cross entropy mini-batch Trainer \n")
  ce = C.cross_entropy_with_softmax(nnet, label_Var)
  pe = C.classification_error(nnet, label_Var)

  fixed_lr = 0.05
  lr_per_batch = learning_rate_schedule(fixed_lr,
    UnitType.minibatch)
  learner = C.sgd(nnet.parameters, lr_per_batch)
  trainer = C.Trainer(nnet, (ce, pe), [learner])

  max_iter = 5000  # 5000 maximum training iterations
  batch_size = 5   # mini-batch size  5
  progress_freq = 1000  # print error every n minibatches

  reader_train = create_reader(train_file, True, input_dim,
    output_dim)
  my_input_map = {
    input_Var : reader_train.streams.features,
    label_Var : reader_train.streams.labels
  }
  pp = ProgressPrinter(progress_freq)

  print("Starting training \n")
  for i in range(0, max_iter):
    currBatch = reader_train.next_minibatch(batch_size,
      input_map = my_input_map)
    trainer.train_minibatch(currBatch)
    pp.update_with_trainer(trainer)
  print("\nTraining complete")

  # ---------------------------------

  print("\nEvaluating test data \n")
  reader_test = create_reader(test_file, False, input_dim,
    output_dim)
  numTestItems = 30
  allTest = reader_test.next_minibatch(numTestItems,
    input_map = my_input_map)
  test_error = trainer.test_minibatch(allTest)
  print("Classification error on the 30 test items = %f"
    % test_error)

  # ---------------------------------

  # make a prediction for an unknown flower
  # first train versicolor = 7.0,3.2,4.7,1.4,0,1,0
  unknown = np.array([[6.9, 3.1, 4.6, 1.3]],
    dtype=np.float32)
  print("\nPredicting Iris species for input features: ")
  my_print(unknown[0], 1)  # 1 decimal

  predicted = nnet.eval( {input_Var: unknown} )
  print("Prediction is: ")
  my_print(predicted[0], 3)  # 3 decimals

  # ---------------------------------

  print("\nTrained model input-to-hidden weights: \n")
  print(hLayer.hidLayer.W.value)
  print("\nTrained model hidden node biases: \n")
  print(hLayer.hidLayer.b.value)

  print("\nTrained model hidden-to-output weights: \n")
  print(oLayer.outLayer.W.value)
  print("\nTrained model output node biases: \n")
  print(oLayer.outLayer.b.value)

  save_weights("weights.txt", hLayer.hidLayer.W.value,
    hLayer.hidLayer.b.value, oLayer.outLayer.W.value,
    oLayer.outLayer.b.value)

  return 0  # success

def main():
  print("\nBegin Iris demo (CNTK 2.0) \n")
  np.random.seed(0)
  do_demo()  # all the work is done in do_demo()

if __name__ == "__main__":
  main()

# end script

-----------------------------------------------------------------------
```

built-in CNTK reader functions, or you can use data in non-CTF format and write a custom reader function. The demo program uses the CTF data format approach. File trainData_cntk.txt looks like:

```
|attribs 5.1 3.5 1.4 0.2 |species 1 0 0
...
|attribs 7.0 3.2 4.7 1.4 |species 0 1 0
...
|attribs 6.9 3.1 5.4 2.1 |species 0 0 1
```

You specify the feature (predictor) values by using the "|" character followed by a string identifier, and the label values in the same way. You can use whatever you like for identifiers.

To create the training data, I go to the Wikipedia entry for Fisher's Iris Data, copy and paste all 150 items into Notepad, select the first 40 of each species, and then do a bit of edit-replace. I use the leftover 10 of each species in the same way to create the testData_cntk.txt file. The create_reader function that uses the data files is defined as:

```
def create_reader(path, is_training, input_dim, output_dim):
  return MinibatchSource(CTFDeserializer(path, StreamDefs(
    features = StreamDef(field='attribs', shape=input_dim,
      is_sparse=False),
    labels = StreamDef(field='species', shape=output_dim,
      is_sparse=False)
  )), randomize = is_training,
    max_sweeps = INFINITELY_REPEAT if is_training else 1)
```

You can think of this function as boilerplate for CTF files. The only thing you'll need to edit is the string identifiers ("attribs" and "species" here) used to identify features and labels.

Creating a Neural Network

The definition of function do_demo begins with:

```
def do_demo():
  input_dim = 4
  hidden_dim = 2
  output_dim = 3

  train_file = "trainData_cntk.txt"
  test_file = "testData_cntk.txt"

  input_Var = C.ops.input(input_dim, np.float32)
  label_Var = C.ops.input(output_dim, np.float32)
...
```

The meanings and values of the first five variables should be clear to you. Variables input_Var and label_Var are created using the built-in function named input, located in the cntk.ops package. You can think of these variables as numeric matrices, plus some special properties needed by CNTK.

The neural network is created with these statements:

```
print("Creating a 4-2-3 tanh softmax NN for Iris data ")
with default_options(init = glorot_uniform()):
  hLayer = Dense(hidden_dim, activation=C.ops.tanh,
    name='hidLayer')(input_Var)
  oLayer = Dense(output_dim, activation=C.ops.softmax,
    name='outLayer')(hLayer)
nnet = oLayer
```

The Dense function creates a fully connected layer of nodes. You pass in the number of nodes and an activation function. The name parameter is optional in general, but is needed if you want to extract the weights and biases associated with a layer. Notice that instead of passing an array of input values for a layer into the Dense function, you append an object holding those values to the function call.

When creating a neural network layer, you should specify how the values for the associated weights and biases are initialized, using the init parameter to the Dense function. The demo initializes weights and biases using the Glorot (also called Xavier initialization) mini-algorithm implemented in function glorot_uniform. There are several alternative initialization functions in the cntk.initializer module.

The statement nnet = oLayer creates an alias for the output layer named oLayer. The idea is that the output layer represents a single layer, but also the output of the entire neural network.

Training the Neural Network

After training and test data have been set up, and a neural network has been created, the next step is to train the network. The demo program creates a trainer with these statements:

```
print("Creating a cross entropy mini-batch Trainer \n")
ce = C.cross_entropy_with_softmax(nnet, label_Var)
pe = C.classification_error(nnet, label_Var)

fixed_lr = 0.05
lr_per_batch = learning_rate_schedule(fixed_lr,
  UnitType.minibatch)
learner = C.sgd(nnet.parameters, lr_per_batch)
trainer = C.Trainer(nnet, (ce, pe), [learner])
```

The most common approach for measuring training error is to use what's called cross-entropy error, also known as log loss. The main alternative to cross-entropy error for numeric problems similar to the Iris demo is the squared_error function.

After training has completed, you're more interested in classification accuracy than in cross-entropy error—you want to know how many correct predictions the model makes. The demo uses the built-in classification_error function.

There are several optimization algorithms that can be used to minimize error during training. The most basic is called stochastic gradient descent (SGD), which is often called back-propagation. Alternative algorithms supported by CNTK include SGD with momentum, Nesterov and Adam (adaptive moment estimation).

The mini-batch form of SGD reads in one subset of the training items at a time, calculates the calculus gradients, and then updates all weights and bias values by a small increment called the learning rate. Training is often highly sensitive to the values used for the learning rate. After a CNTK trainer object has been created, the demo prepares training with these statements:

```
max_iter = 5000
batch_size = 5
progress_freq = 1000

reader_train = create_reader(train_file, True,
  input_dim, output_dim)
my_input_map = {
  input_Var : reader_train.streams.features,
  label_Var : reader_train.streams.labels
}
pp = ProgressPrinter(progress_freq)
```

The SGD algorithm is iterative, so you must specify a maximum number of iterations. Note that the value for the mini-batch size should be between 1 and the number of items in the training data.

The reader object for the trainer object is created by a call to create_reader. The True argument that's passed to create_reader tells the function that the reader is going to be used for training data rather than test data and, therefore, that the data items should be processed in random order, which is important to avoid training stagnation.

The my_input_map object is a Python two-item collection. It's used to tell the reader object where the feature data resides (input_Var) and where the label data resides (label_Var). Although you

can print whatever information you wish inside the main training loop, the built-in ProgressPrinter object is a very convenient way to monitor training. Training is performed with these statements:

```
print("Starting training \n")
for i in range(0, max_iter):
  currBatch = reader_train.next_minibatch(batch_size,
    input_map = my_input_map)
  trainer.train_minibatch(currBatch)
  pp.update_with_trainer(trainer)
print("\nTraining complete")
```

In each training iteration, the next_minibatch function pulls a batch (5 in the demo) of training items, and uses SGD to update the current values of weights and biases.

Testing the Network

After a neural network has been trained, you should use the trained model on the holdout test data. The idea is that given enough training time and combinations of learning rate and batch size, you can eventually get close to 100 percent accuracy on your training data. However, excessive training can over-fit and lead to a model that predicts very poorly on new data.

```
print("\nEvaluating test data \n")
reader_test = create_reader(test_file, False, input_dim,
  output_dim)
numTestItems = 30
allTest = reader_test.next_minibatch(numTestItems,
  input_map = my_input_map)
test_error = trainer.test_minibatch(allTest)
print("Classification error on the 30 test items = %f"
  % test_error)
```

The next_minibatch function examines all 30 test items at once. Notice that you can reuse the my_input_map object for the test data because the mapping to input_Var and label_Var is the same as to the training data.

Making Predictions

Ultimately, the purpose of a neural network model is to make predictions for new, previously unseen data.

```
unknown = np.array([[6.9, 3.1, 4.6, 1.3]],
  dtype=np.float32)
print("\nPredicting Iris species for features: ")
my_print(unknown[0], 1)  # 1 decimal
predicted = nnet.eval( {input_Var: unknown} )
print("Prediction is: ")
my_print(predicted[0], 3)  # 3 decimals
```

The variable named unknown is an array-of-array-style numpy matrix, which is required by a CNTK neural network. The eval function accepts input values, runs them through the trained model using the neural network input-output process and the resulting three probabilities (0.263, 0.682, 0.055) are displayed.

In some situations it's useful to iterate through all test items and use the eval function to see exactly which items were incorrectly predicted. You can also write code that uses the numpy.argmax function to determine the largest value in the output probabilities and explicitly print "correct" or "wrong."

Exporting Weights and Biases

The demo program concludes by fetching the trained model's weights and biases, and then displays them to the shell, as well as saves them to a text file. The idea is that you can train a neural network using CNTK, then use the trained model weights and biases in another system, such as a C# program, to make predictions.

The weights and bias values for the hidden layer are displayed like this:

```
print("\nTrained model input-to-hidden weights: \n")
print(hLayer.hidLayer.W.value)
print("\nTrained model hidden node biases: \n")
print(hLayer.hidLayer.b.value)
```

Recall that a CNTK network layer is a named object (hLayer), but that an optional name property was passed in when the layer was created (hidLayer). The tersely named W property of a named layer returns an array-of-arrays-style matrix holding the input-to-hidden weights. Similarly, the b property gives you the biases. The weights and biases for the output layer are obtained in the same way:

```
print("\nTrained model hidden-to-output weights: \n")
print(oLayer.outLayer.W.value)
print("\nTrained model output node biases: \n")
print(oLayer.outLayer.b.value)
```

The values of the $(4 * 2) + (2 * 3) = 14$ weights, and the $(2 + 3) = 5$ biases, are saved to text file, and function do_demo concludes, like so:

```
...
  save_weights("weights.txt", hLayer.hidLayer.W.value,
  hLayer.hidLayer.b.value, oLayer.outLayer.W.value,
  oLayer.outLayer.b.value)
  return 0  # success
```

The program-defined save_weights function writes one value per line. The order in which the values are written (input-to-hidden weights, then hidden biases, then hidden-to-output weights, then output biases) is arbitrary, so any system that uses the values from the weights file must use the same order.

Wrapping Up

If you're new to neural networks, the number of decisions you have to make when using CNTK might seem a bit overwhelming. You need to decide how many hidden nodes to use, pick a hidden layer activation function, a learning optimization algorithm, a training error function, a training weight-initialization algorithm, a batch size, a learning rate and a maximum number of iterations.

However, in most cases, you can use the demo program presented in this article as a template, and experiment mostly with the number of hidden nodes, the maximum number of iterations, and the learning rate. In other words, you can safely use tanh hidden layer activation, cross-entropy for training error, Glorot initialization for weights and biases, and a training mini-batch size that is roughly 5 percent to 10 percent of the number of training items. The one exception to this is that instead of using the SGD training optimization algorithm, even though it's the most commonly used, I suggest using the Adam algorithm.

Once you become familiar with CNTK basics, you can use the library to build very powerful, advanced, deep neural network architectures such as convolutional neural networks (CNNs) for image recognition and long short-term memory recurrent neural networks (LSTM RNNs) for the analysis of natural language data. ■

Dr. James McCaffrey *works for Microsoft Research in Redmond, Wash. He has worked on several Microsoft products, including Internet Explorer and Bing. Dr. McCaffrey can be reached at jamccaff@microsoft.com.*

Thanks to the following Microsoft technical experts who reviewed this article: *Chris Lee and Sayan Pathak*

Doing Data Science and AI with SQL Server

Wee Hyong Tok

Data is an important asset for every business. Whether you're in the retail, health care, telecommunications, utilities or financial businesses, you're familiar with the following two use cases:

- In an online transaction processing (OLTP) scenario, transactional data is stored in a database. The transaction data is produced by various line-of-business (LOB) applications.
- In a data warehousing scenario, data is acquired from various heterogeneous data sources, transformed and cleansed, and loaded into data warehouses. The consolidated data provides the single source of truth for business reporting and dashboards. At the same time, it also enables interactive analysis via multi-dimensional online analytical processing (OLAP) cubes, and tabular models.

Getting from raw data to insights empowers business decision makers to gain a deeper understanding into each aspect of the business and helps them react to new business situations quickly. For example, consider a retail scenario. The business analyst notices that sales are dropping for specific retail stores. The business analyst wants to drill down to understand the details on what's causing the drop in sales. By being able to run the analysis (aggregating, joining of data from multiple data sources, filtering and so on) on a large amount of data, it enables deep analysis of customer behavior and trends in the retail stores. Microsoft SQL Server powers these mission-critical applications.

Many companies have started on digital transformation to modernize their data platform to keep pace with the ever-growing requirements on the type of data that needs to be stored and the volume in which the data is being acquired.

As part of this digital transformation, advanced analytics plays an important role. Specifically, companies have been either building up data science teams within their companies or leveraging external resources to do data science. They use data science to distill data assets into nuggets of gold that can help them proactively deliver personalized customer experiences (personalized Web sites, product recommendations, customer lifetime value and so on), reduce downtime for equipment (predicting remaining useful lifetime) and more. The potential use of data science and how it can literally change businesses is exciting.

Some common use cases (non-exhaustive) of data science include the following:

Identifying Loan Credit Risk: A lending institution (a credit bureau) might want to leverage loan credit risk models to determine the borrowers that are likely to default and reduce the number of loans given to these high-risk borrowers.

Managing Customer Churn: Customer churn models have been used extensively (in retail and by telecommunication providers). For example, customers leveraging mobile services offered by telecommunication providers have a rich variety of choices and can easily switch between service providers. Managing customer churn is important to reduce customer acquisition costs and maintain a high-quality service. In addition, retail companies are using

This article discusses:

- Serving AI with data
- Built-in AI capabilities of SQL Server
- How to develop R/Python AI stored procedures

Technologies discussed:

SQL Server, R and Python Tools for Visual Studio, Data Science, Artificial Intelligence, R and Python

churn models to predict customers that are most likely to churn and to understand the key factors that lead to those churns.

Reducing Hospital Readmission: Reducing readmission rates for various medical conditions (heart attack, pneumonia, coronary artery bypass surgery, to name a few) is important to hospitals. In the United States, hospitals face penalties if the readmission rate is too high. Hospitals leverage predictive models for predicting patients that are more susceptible to being readmitted within 30 days. This helps them understand the root causes for the 30-day readmission, and helps them work toward addressing them.

This presents an exciting opportunity for database professionals and developers to either work with data scientists, or put on a data scientist hat to build predictive models that can help to assess credit loan risk, manage customer churn, reduce hospital admissions and more. The possibilities for developers to turn all these raw data assets sitting in the database to golden, strategic insights is exciting.

This article shows how you can work with data science and artificial intelligence (AI) with SQL Server. You'll also learn how to jump-start your journey of using R and Python with SQL Server.

Why Doing Data Science with SQL Server Matters

What does doing data science mean and why does it matter to the database person? Today, most data scientists first figure out how to connect to many data sources (databases included), bring the data out from the database, and use the historical data to train and subsequently test the machine learning models that they've built.

A typical approach used by data scientists is to read the data from the database into the client that they're using for building the model. Once the data is read, they combine the data with other data sources. For data scientists developing the models using R, packages like dplyr are commonly used for performing aggregation, joins and for filtering. With the data transformed into the right shape, data scientists continue the data modeling process, and start doing feature engineering. As part of feature engineering, new features (such as adding new columns in a SQL Server table) might get created, existing features might get transformed (scaling it to -1 to 1, or 0 to 1, applying logarithmic transformation, computing the z-score, binnning the data, and so on) or removed. Feature engineering plays a very important role in laying the groundwork needed for a good predictive model. Once all these steps are completed, the data scientist develops the models and validates it using test data before figuring out an operationalization plan for the model to be deployed to production so that applications can consume them.

At this point, as a database person, you might ask, "Why do I need to move the data out from the database to do data science? Should we push the processing of joins and aggregations (Sum, Min, Max and so on) into the database?"

Why does it make sense to do this in the database? First, data movement is expensive. If the machine learning models can run where the data is stored, this removes the need to move data between the database and the client application. Second, a new working copy of the data is extracted from the database and stored external to the database. The implication is that many of the security policies and audits that apply to data stored in the database can no longer be enforced. Third, if the computation of joins and aggregations can be done where the data is located, you can leverage decades of database innovations (leveraging indexes—clustered and non-clustered, in-memory tables, column stores, high-availability and so on). If training the model can be done where the data is stored, it can lead to performance improvements.

In addition, if the data science project involves working with spatial data, temporal data or semi-structured data, you can leverage SQL Server capabilities that let you do this efficiently. For example, if you're working on a data science project (say a land-use classification problem) where you must manipulate spatial data, the geography and geometry data types in SQL Server will provide a good way to store the spatial data. Once it's stored as spatial data in SQL Server, you can leverage SQL Server spatial functions to query for nearest neighbors, compute the spatial distance using different spatial reference systems and more. The spatial data can be indexed to facilitate efficient query processing.

As a database professional and developer, you have tremendous knowledge and value to bring to a data science project. By doing data science and AI where the data resides, there are many benefits. These include being able to take advantage of the enterprise-grade performance, scale, security and reliability that you've come to expect from SQL Server over the years. More important, you eliminate the need for expensive data movement.

Figure 1 illustrates the difference between doing data science and AI outside of the database. From the application perspective, a developer doesn't need to learn new methods to tap the power of AI teeming in SQL Server. It connects to it the same way it connects to a database today, and invokes SQL Server-stored procedures, which encapsulates the R or Python code. The stored procedure has just become an intelligent AI stored procedure.

Another important consideration in data science projects is operationalization. The predictive model that has been developed by data scientists needs to be deployed into a production environment for it to be used by applications. With the release of SQL Server 2016 R Services, you can wrap R code as part of the stored procedures. After training is done, the predictive models are stored as varbinary(max) in a database table.

An application (.NET, Java, Node.js and more) would connect to SQL Server and invoke the stored procedures to use the predictive model for making predictions on new instances of data. Continuing the momentum, SQL Server 2017 CTP2 added Python support. You now have the best of multiple worlds: the ability to write code in R or Python, leverage the rich set of R and Python libraries for machine learning and deep learning, and consume the predictive models in any application.

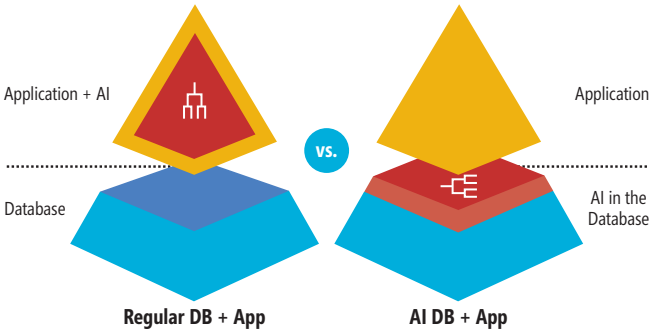


Figure 1 **Doing Data Science and AI Where the Data Is Stored**

Using Stored Procedures for Machine Learning and AI

By encapsulating the machine learning and AI models as part of the SQL Server stored procedure, it lets SQL Server serve AI with the data. There are other advantages for using stored procedures for operationalizing machine learning and AI (ML/AI). These include:

Applications can leverage existing database drivers to connect to SQL Server: Many programming languages have database drivers that enable them to connect to SQL Server. These database drivers (OLEDB, ODBC, JDBC, MSPHPSQL and Node.js Driver for SQL Server) are used by application developers to develop cutting-edge applications that "talk" to SQL Server.

In addition, companies might have existing LOB applications that are already operational. By leveraging ML/AI-stored procedures in SQL Server, these LOB applications can easily evolve into intelligent applications. With the R or Python code needed to work with the ML/AI models encapsulated in stored procedures, application developers can now leverage their ML/AI stored procedures as is (without requiring new libraries or learning new database access patterns). To the application layer, the intelligent ML/AI stored procedure behaves just like any SQL Server stored procedure.

Being backed by the full spectrum of SQL Server enterprise-ready capabilities: Some considerations include ...

- Where can I host the model for doing inference?
- Which users can access the ML/AI model?
- When the model is used for prediction (aka "inference"), it might need to access specific data in the database. In what security context should the model and associated R/Python code execute?
- How can I ensure the R/Python code doesn't use up all the SQL Server resources?

SQL Server provides enterprise-ready capabilities from using row-level security to limit the data that can be accessed, to providing database admins with both server and database scoped database audits, to enabling ownership-chaining for SQL Server securable, to being able to sign stored procedures with a certificate or asymmetric key, resource governance and more. These enterprise-ready SQL Server capabilities can be used by the ML/AI stored procedures as is, without requiring the data scientist to reinvent the wheel for serving data at scale. Most important, the DBAs today can leverage their existing skills to secure and manage the ML/AI stored procedures.

Mature development tools to develop the ML/AI stored procedure: As a database developer, you can develop the stored procedure and the R and Python code in one place: Visual Studio. With the availability of SQL Server Data Tools for Visual Studio, R Tools for Visual Studio, and Python Tools for Visual Studio, you can do development of the T-SQL, R, or Python code, check it into a source control system, write unit tests, automate testing, and perform code review, and more. Database developers and data scientists can now work together to develop the ML/AI stored procedures, each focusing on their respective areas of expertise.

Steps to Get Started with SQL Server, Data Science and AI

There has never been a more exciting time and better opportunity for us as database professionals and developers to work with data science and AI with SQL Server. You can get started in three easy steps:

Install SQL Server 2016 or SQL Server 2017 CTP2. When installing SQL Server 2017 CTP2, you select the type of in-database machine learning services that you want to install. You can choose to use R, Python or both. Once SQL Server completes setup, you'll be able to start using R or Python code as part of stored procedures, as shown in **Figure 2**.

Note: If you're using SQL Server 2016, only R-Services (In-Database) will be shown at setup.

You can refer to bit.ly/2qXoyyC for more information on setting up R and Python with SQL Server.

Enable external script. To use R or Python code in the stored procedure, you'll need to configure SQL Server to allow external scripts. To enable external scripts, you'll need to run the sp_configure and reconfigure commands (after the T-SQL code is successfully executed, you'll need to restart the SQL Server service):

```
exec sp_configure  'external scripts enabled', 1
reconfigure  with override
```

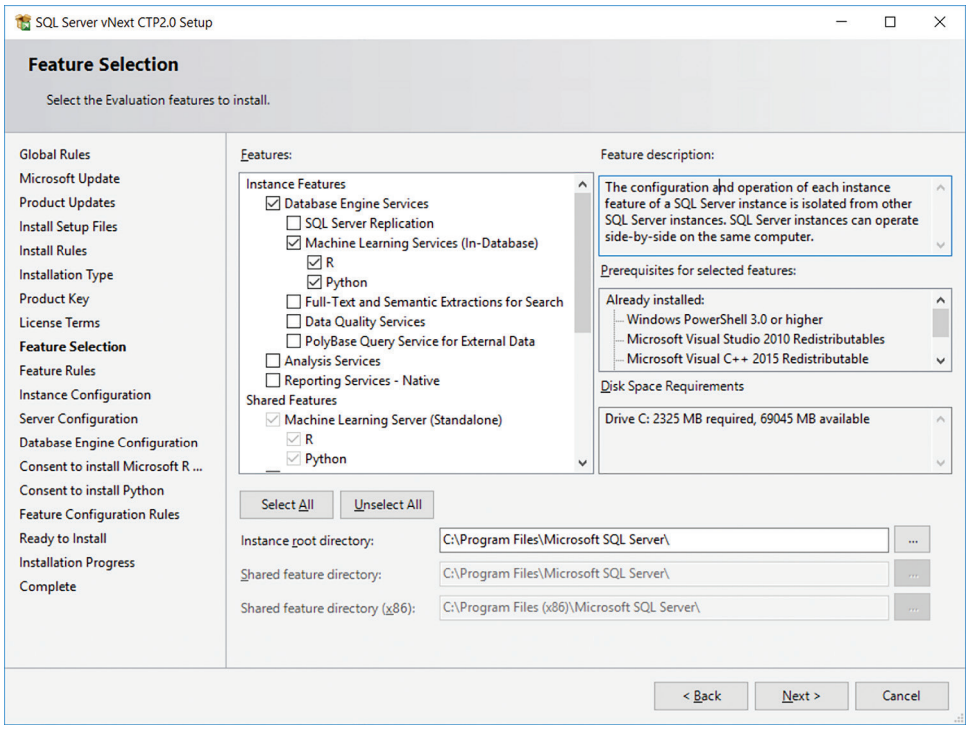


Figure 2 **Using SQL Server 2017 CTP2 Setup to Install Machine Learning Services (In-Database)**

After running the commands, you'll see the following output:

```
Configuration option 'external scripts enabled' changed from 1 to 1. Run
the RECONFIGURE statement to install.
```

Hello, AI! You're now ready to do data science and AI with SQL Server. Using either Visual Studio, Visual Studio Code or SQL Server Management Studio, you can connect to the SQL Server instance with R or Python. You can run the code provided in **Figure 3** to make sure that R and Python are installed correctly. The code will output "Hello AI."

If you need to install additional R libraries, you should set lib.SQL as the location of the SQL Server library. Similarly, if you need to install additional Python libraries, you can do a pip install of the relevant Python libraries and set the PYTHONPATH to point to where Python has been installed.

You can find the default locations of the R library files at <Drive>:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\R_SERVICES\library and Python library files at <Drive>:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\PYTHON_SERVICES\library.

Let's dive deeper into what's in a stored procedure (with R code) that's used for training a customer churn classification model. **Figure 4** shows a stored procedure that trains a customer churn classification model by building an ensemble of decision trees using R. **Figure 5** shows a similar stored procedure that trains a similar churn model using Python. Database developers leverage familiar skills of using T-SQL queries to select data from SQL server tables. The data is used as inputs to the R or Python code.

Figure 3 **Code to Ensure R and Python Are Installed Correctly**

```
Code for R
exec sp_execute_external_script  @language =N'R',
           @script=N'OutputDataSet<-InputDataSet',
           @input_data_1 =N'select ''Hello AI'' as txt'
           with result sets (([txt] nvarchar(8)));
go

Code for Python
exec sp_execute_external_script  @language =N'Python',
           @script=N'OutputDataSet = InputDataSet',
           @input_data_1 =N'select ''Hello AI'' as txt'
           with result sets (([txt] nvarchar(8)));
```

Figure 4 **Creating a Stored Procedure to Train a Decision Forest Model Using R**

```
CREATE PROCEDURE trainRDecisionForestModel AS
  BEGIN
    execute sp_execute_external_script @language = N'R',
    @script = N'
      require("RevoScaleR");
      labelVar = "churn"
      trainVars <- rxGetVarNames(telcoCDR_Data_train)
      trainVars <- trainVars[!trainVars %in% c(labelVar)]

      temp <- paste(c(labelVar, paste(trainVars, collapse = "+")), collapse = "~")
      formula <- as.formula(temp)
      rx_forest_model <- rxDForest(formula = formula,
                                   data = telcoCDR_Data_train,
                                   nTree = 8, maxDepth = 32, mTry = 2,
                                   minBucket=1, replace = TRUE, importance = TRUE,
                                   seed=8, parms=list(loss=c(0,4,1,0)))

      rxDForest_model <- data.frame(
        payload = as.raw(serialize(rx_forest_model, connection=NULL))); '
        @input_data_1 = N'select * from telcoCDR_Data_train'
        @input_data_1_name = N'telcoCDR_Data_train'
        @output_data_1_name = N'rxDForest_model'
      with result sets ((model varbinary(max)));
end;
```

Figure 5 **Creating a Stored Procedure to Train a Random Forest Model Using Python**

```
CREATE PROCEDURE trainPythonRandomForestModel (@trained_model
varbinary(max) OUTPUT) AS
 BEGIN
  execute sp_execute_external_script @language = N'Python',
  @script = N'
    df = churn_train_data

    # Get all the columns
    columns = df.columns.tolist()

    # Specify the label column
    target = "churn"

    import numpy as np
    from sklearn.ensemble import RandomForestClassifier

    churn_model = RandomForestClassifier(n_estimators=20, max-depth=5)
    churn_model.fit(df[columns], df[target])

    import pickle

    #Serialize the model as a binary object
    trained_model = pickle.dumps(lin_model)

    @input_data_1 = N'select "TotalMinutesUsedLastMonth", "State",
      "CallDropRate", "UnPaidBalance", "TotalCallDuration", "TotalDataUsageMB"
      from dbo.telco_churn_data where Year = 2017'
    @input_data_1_name = N'churn_train_data'
    @params = N'@trained_model varbinary(max) OUTPUT'
    @trained_model = @trained_model OUTPUT;
END;
```

Once the model is trained, it's serialized and returned as varbinary(max). The model can then be stored in a SQL Server table.

The complete sample code for a customer churn model can be found at aka.ms/telcochurnsamples.

With SQL Server 2017 CTP2, you can run Python code in the stored procedures. By supporting Python in SQL Server, it opens up new opportunities for you to use many of the deep learning toolkits (CNTK, TensorFlow and more), which provide Python APIs. In addition, the deep-learning toolkits enable you to specify the use of GPUs when training your model. You can now use SQL Server 2017 to perform intensive deep-learning jobs on text, images, and unstructured data that are stored in SQL Server, and then operationalize and do inference with SQL Server. Very exciting!

Wrapping Up

SQL Server has evolved over the years into a top-notch, enterprise-ready, scalable and hybrid data platform. This lets companies build intelligent, mission-critical applications, backed by decades of database innovations from indexes, spatial indexes, in-memory, column stores, high availability, resource governance and more. With the SQL Server 2017 release, with built-in R and Python support, SQL Server is in a unique position to fuel innovations that database professionals and developers can co-create with the data science and AI communities. The possibilities are endless. ■

Wee Hyong *has worn many hats in his career: developer, program/product manager, data scientist, researcher and strategist. His gamut of experience spanning industry and research has given him unique abilities to help organizations accelerate their digital transformations using data science and artificial intelligence. You can follow him on Twitter: @weehyong.*

Thanks to the following Microsoft technical expert for reviewing this article: *Joy Qiao*

Scale Applications with Microsoft Azure Redis Cache and Machine Learning

Stefano Tempesta

In a multi-tier application, bottlenecks can occur at any of the connection points between two tiers: at business logic and data access layers, client and service layers, presentation and storage layers, and so on. Large-scale applications can implement various levels of caching of information for improving performance and increasing scalability. Caching can be configured in memory or on some more permanent form of storage, in different sizes, and in diverse geographic locations. The open source Redis engine, as implemented in Azure, lets you intuitively configure and manage all these aspects, and use a variety of programming languages.

This article presents design best practices and code examples for implementing the Azure Redis Cache and tuning the performance of ASP.NET MVC applications, optimizing cache hit ratio and reducing "miss rate" with smart algorithms processed by Azure Machine Learning.

This article discusses:

- Connecting to an instance of Redis Cache in Azure
- Typical Cache Design Patterns
- Demand Estimation in Azure Machine Learning
- Consuming an Azure Machine Learning Web service

Technologies discussed:

Microsoft Azure Redis Cache and Machine Learning, ASP.NET MVC, Linear Regression

Code download available at:

bit.ly/2qkV65u

Azure Redis Cache

Let's start by saying that Redis is not a Microsoft product. Redis is an open source project freely available for download from the Web site redis.io. Everyone can download the cache engine and install it on their servers. But Microsoft offers this, and much more, as a service in Azure. You can create a new Redis Cache instance in Azure in a few minutes and be ready to connect to it from your application.

What makes Redis different from other caching frameworks is its support for specialized data types, besides the typical key-value string pair, common in other cache engine implementations. You can run atomic operations on these types, such as appending to a string, incrementing the value in a hash, pushing an element to a list, computing set intersection, union and difference, or getting the member with highest ranking in a sorted set.

From an operational perspective, the Redis implementation in Azure lets you replicate cache instances in a two-node primary/secondary configuration, entirely managed by Microsoft. Redis also supports master-subordinate replication, with fast non-blocking first synchronization, auto-reconnection on net split, and so forth. Just to expand on the replication feature, which, for me, is a differentiation point:

- Redis replication is non-blocking on the master side. This means that the master will continue to handle queries when one or more slaves perform the initial synchronization.
- Replication is non-blocking on the slave side. While the slave is performing the initial synchronization, it can handle queries using the old version of the dataset.

Optionally, Redis supports persistence. Redis persistence lets you save data stored in Redis cache permanently to an allocated

storage in Azure. You can also take snapshots and back up the data, which you can reload in case of a failure.

Last, Azure Redis Cache comes with important monitoring capabilities that, when enabled, provide insights on the utilization of the cache, in terms of cache hits and misses, used storage, and so on, as shown in **Figure 1**.

Connecting to Redis Cache in .NET

Once Redis Cache is configured in Azure, you define a unique URL to access it from a software application and obtain a key for authentication. These two pieces of information are necessary to establish a connection to the Redis Cache engine from your application. Let's build an ASP.NET MVC application then, which stores objects in Redis Cache.

A common option for storing the cache access credentials is the Web.config file of the ASP.NET MVC application. In the <appSettings> section, you can simply add a key:

```
<add key="CacheConnection" value="<instance-
  name>.redis.cache.windows.net,abortConnect=true,ssl=false,password=<instance-key>"/>
```

Parameter abortConnect is set to true, which means that the call won't succeed if a connection to the Azure Redis Cache can't be established. You might opt for a secured connection over HTTPS by setting the ssl parameter to true.

You also need to add one of the following NuGet packages to your projects:

- **StackExchange.Redis:** A .NET implementation of a Redis Cache client, which provides an easy-to-use interface to Redis commands.
- **Newtonsoft.Json:** The popular JSON framework for deserializing objects to JSON and allowing storage in a Redis Cache database.

In its simplified implementation, the MVC application stores and retrieves contact details from the Redis Cache by defining CRUD actions in a Controller. If the object isn't found in the cache, it will be restored from the database and then stored in the cache for future access.

Your Contact model is defined as follows, with a few properties to enter name, e-mail address and country of origin:

```
public class Contact
{
  public Guid Id { get; set; }

  [DisplayName("Contact Name")]
  public string Name { get; set; }

  [DataType(DataType.EmailAddress)]
  public string Email { get; set; }

  public string Country { get; set; }
}
```

You now add a new Contacts Controller to the application, choosing to add views using Entity Framework. As you'll see, however, you'll add a layer of fetching data from the cache first, before hitting the database.

Connecting to the Redis Cache, then, is purely a matter of defining a connection using the connection string stored in the Web.config file. Your ContactsController class will look something like this:

```
public class ContactsController : Controller
{
  static string cacheConnectionString =
    ConfigurationManager.AppSettings["CacheConnection"].ToString();

  ConnectionMultiplexer connection =
    ConnectionMultiplexer.Connect(cacheConnectionString);
```

Now, this is far from being a good practice for defining a connection to any storage system, as hardcoding the ConnectionMultiplexer class inside the Controller's code clearly creates a highly coupled dependency. Ideally, you'd inject this dependency using an Inversion of Control library. However, for the sake of keeping things simple and straight in this example, the ConnectionMultiplexer class is all you need to obtain a connection to an instance of Redis Cache. The ConnectionMultiplexer, defined in the StackExchange.Redis namespace, works as a factory by exposing a static Connect method, and returning an instance of itself as a live connection to the defined Redis Cache.

A different approach to sharing a ConnectionMultiplexer instance in your application is to have a static property that returns a connected instance. This provides a thread-safe way to initialize only a single connected ConnectionMultiplexer instance, which can be shared in a singleton class. By masking the ConnectionMultiplexer behind a Lazy object, you also obtain just-in-time (JIT) allocation of the connection when actually used by a Controller's action:

```
static Lazy<ConnectionMultiplexer> lazyConnection =
  new Lazy<ConnectionMultiplexer>(() =>
{
  return ConnectionMultiplexer.Connect(cacheConnectionString);
});

static ConnectionMultiplexer Connection => lazyConnection.Value;
```

Reading from and Writing to a Redis Cache Instance

Now that you've established a connection to a Redis Cache instance, you can access it in the read-and-write actions of the MVC Controller. The Get method of a ContactManager class checks for an instance of the object identified by its ID in cache, and if not found, will retrieve it from the database and allocate it in Redis for future access, as shown in **Figure 2**.

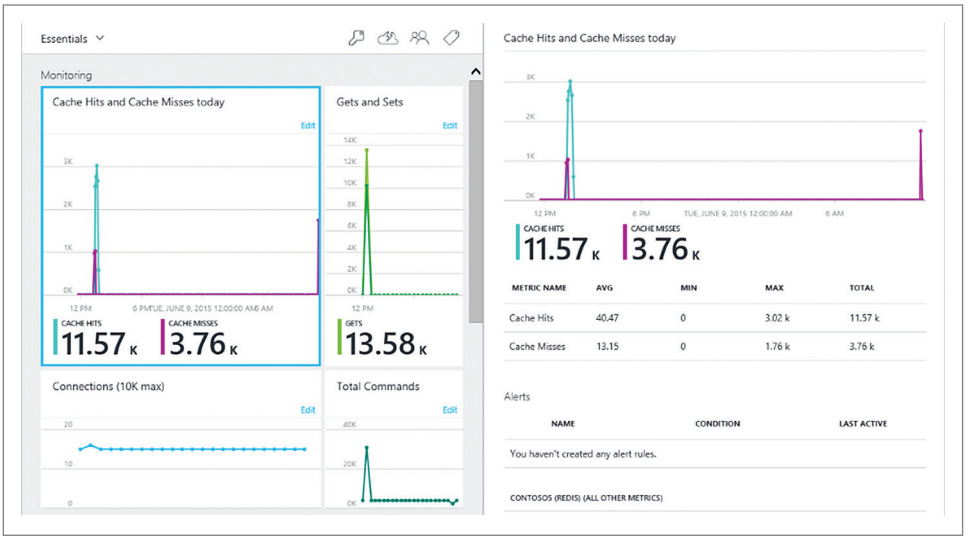


Figure 1 **Redis Insights in Microsoft Azure**

Figure 2 **Get Method in the ContactManager Class**

```
public async Task<Contact> Get(Guid id)
{
  IDatabase cache = cacheContext.GetDatabase();

  var value = cache.HashGet(cacheKeyName, id.ToString());

  // Return the entry found in cache, if any
  // HashGetAsync returns a null RedisValue if no entry is found
  if (!value.IsNull)
  {
    return JsonConvert.DeserializeObject<Contact>(value.ToString());
  }

  // Nothing found in cache, read from database
  Contact contact = databaseContext.Contacts.Find(id);

  // Store in cache for next use
  if (contact != null)
  {
    HashEntry entry = new HashEntry(
      name: id.ToString(),
      value: JsonConvert.SerializeObject(contact));
    await cache.HashSetAsync(cacheKeyName, new[] { entry });
  }

  return contact;
}
```

From the cache context, which identifies a connection to Redis Cache, you'll obtain a reference to the data storage inside Redis itself, through the GetDatabase method. The returned IDatabase object is a wrapper around the Redis cache commands. Specifically, the HashGet method executes the HGET command (bit.ly/2pM0000) to retrieve an object stored against the specified key (the object ID). The HGET command returns the value identified by a unique key in a named hash collection, if existing, or a null value otherwise. As key in the cache, you can use the object's ID (a GUID), consistently with the same ID stored at database level, and managed by Entity Framework.

If an object is found at the indicated key, it's deserialized from JSON into an instance of the Contact model. Otherwise, the object is loaded from the database, using the Entity Framework Find by ID, and then stored in cache for future use. The HashSet method, and more precisely its async variant, is used for storing a JSON serialized version of the Contact object.

Similar to this approach, the other CRUD methods are implemented around the HashSet method for creating and updating objects in Redis Cache, and the HashDelete method for removing them.

The complete source code is available in the associated code download at bit.ly/2qkV65u.

Cache Design Patterns

A cache typically contains objects that are used most frequently, in order to serve them back to the client without the typical overhead of retrieving information from a persistent storage, like a database. A typical workflow for reading objects from a cache consists of three steps, and is shown in **Figure 3**:

1. A request to the object is initiated by the client application to the server.
2. The server checks whether the object is already available in cache, and if so, returns the object immediately to the client as part of its response.
3. If not, the object is retrieved from the persistent storage, and then returned to the client as in Step 2.

In both cases, the object is serialized for submission over the network. At cache level, this object might already be stored in serialized format, to optimize the retrieval process.

You should note that this is an intentionally simplified process. You might see additional complexity if you check for cache expiration based on time, dependent resources and so on.

This configuration is typically called a Level 1 (L1) cache, as it contains one level of cache only. L1 caches are normally used for Session and Application state management. Although effective, this approach isn't optimal when dealing with applications that move large quantities of data over multiple geographies, which is the scenario that we want to optimize. First of all, large data requires large caches to be effective, which in turn are memory-intensive, thus requiring expensive servers with a big allocation of volatile memory. In addition, syncing nodes across regions implies large data transfers, which, again, is expensive and introduces delays in availability of the information in the subordinate nodes.

A more efficient approach to caching objects in data-intensive applications is to introduce a Level 2 (L2) cache architecture, with a first cache smaller in size that contains the most frequently accessed objects in the larger dataset, and a second cache, larger in size, containing the remaining objects. When the object isn't found in the first-level cache, it's retrieved from the second level, and eventually refreshed periodically from the persistent storage. In a geographically

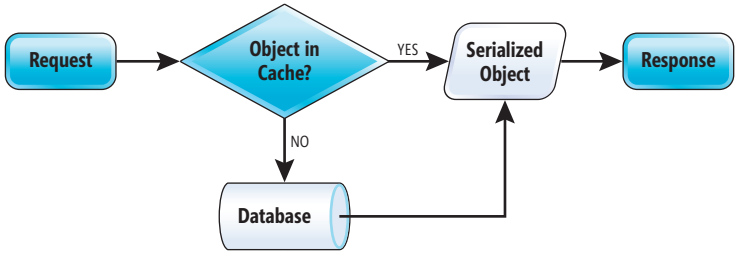


Figure 3 **Level 1 Cache Workflow**

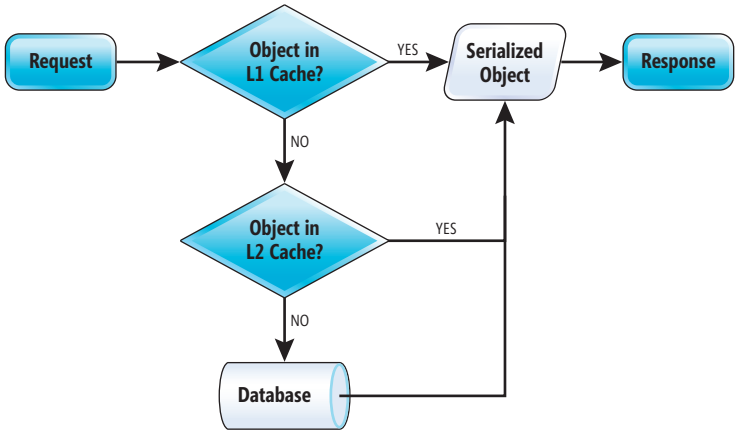


Figure 4 **Level 2 Cache Workflow**

distributed environment, the L2 caches are synced across datacenters, and the L1 cache resides on the master server, as shown in **Figure 4**.

The challenge, then, is to define what goes in the L1 cache, what goes in the L2 cache, and with what frequency the regional nodes should be synced to optimize performance and storage of the cache instances. Performance of a cache is measured as "hit ratio" and "miss ratio." The hit ratio is the fraction of accesses that are a hit (object found in cache) over all of the requests. The miss ratio is the fraction of accesses that are a miss (object not found in cache), or the remaining of the hit ratio to 100 percent.

$$hit\ ratio = \frac{hits}{hits + misses}$$

Figure 5 **Hit Ratio**

With a mathematical formula, you can express the hit ratio as that shown in **Figure 5**.

The miss ratio is expressed as "1 - hit ratio."

To optimize the performance of a cache instance, you want to increase the hit ratio and decrease the miss ratio. Irrespective of adopting an L1 or L2 cache architecture, there are different techniques for improving a cache performance, by pre-fetching data in cache on a regular basis to JIT caching, or allocation of the most used objects based on counters.

A prediction technique based on a machine learning algorithm is called Demand Estimation. Based on patterns of usage of objects, the Demand Estimation algorithm predicts the likelihood that an object will be used and, therefore, it can be allocated in cache before a request is submitted to increase the chance of a hit.

I'll focus on the implementation of the Demand Estimation machine learning algorithm in the context of a data-oriented application, observing what objects are typically accessed, and populating the cache with the most used ones, as predicted by the algorithm's outcome.

Machine Learning

Machine learning is a technique of data science that helps computers learn from existing data in order to forecast future behaviors, outcomes and trends. A machine learning system is one that uses data to make a prediction of some sort. Common techniques include logistic regression and neural network classification. Note that artificial intelligence (AI) is closely related, but generally refers to predictions that are associated with human behavior such as vision and speech. These predictions can make applications look smarter. For example, when you shop online, machine learning services might recommend other products you'd like based on what you've already purchased. When your credit card is swiped, another machine learning service compares your transaction against a database of millions of transactions for an anomaly of behaviors and helps detect a potential fraud.

In the context of a data-oriented application, you start from analyzing the hit ratio in the implemented cache and easily identify some patterns of regular access to specific objects over a period of time, as shown in **Figure 6**. For each object type (Contact, Interest, Brochure and so on), you can also drill down to the individual objects allocated over time and have better insights on the elapsing data flow of your application. But say your business is very seasonal and seasons change according to geography, and specific campaigns might influence traffic, too. So how do you scale your cache hit estimation and create a cache of objects that are most commonly accessed under certain conditions? You implement predictive analysis and demand estimation in Azure Machine Learning.

Predictive analytics uses math formulas that analyze historical data to identify patterns of use to forecast future demand. For your machine learning model to provide predictions, the model must first learn from known data in a process known as "training." During training, data is evaluated by the machine learning algorithm, which analyzes the distribution and type of data, looking for rules and patterns that can be used in later prediction.

Once the training phase is completed, "scoring" is the process of applying a trained model to new data to generate predictions. Making predictions, though, might be error-prone, also for machines. Would the predicted data have a good proportion of true results to total cases, or have high accuracy (a characteristic called "generalization")? Or would it be consistent with the patterns identified in the historical data and give consistent results ("precision")? You need to quality check your prediction. "Evaluation" is the QA of the scoring process.

Demand Estimation in Azure Machine Learning Studio

Azure Machine Learning Studio is a cloud predictive analytics service that makes it possible to quickly create and deploy predictive models as analytics solutions. It's possible to work from a ready-to-use library of algorithms or start with a blank experiment, use them to create models, and deploy the predictive solution as a REST service. Azure Machine Learning Studio is available at studio.azureml.net.

The entire process, or "project" in Azure Machine Learning Studio, consists of the following:

- Importing your initial dataset; that is, the data on which the training process will be based. For our example, it's page views over time and by region.

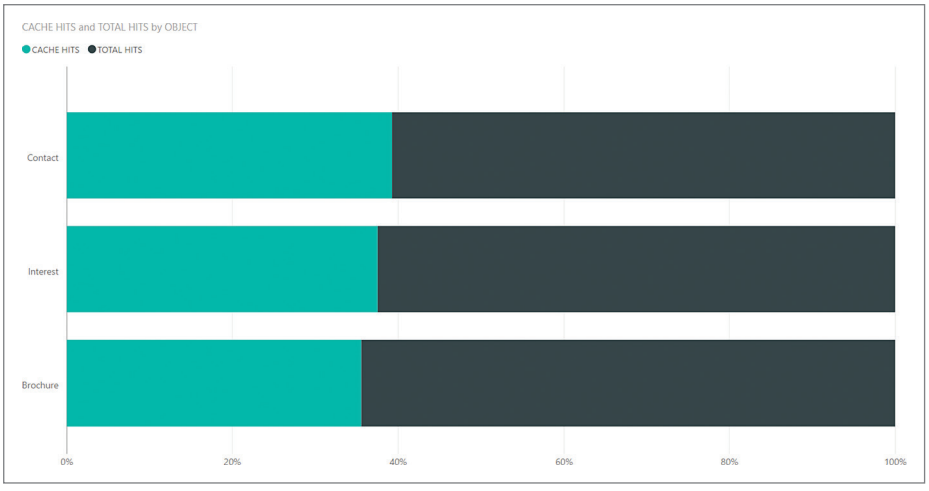


Figure 6 **Cache Hits by Object Type**

- Defining the Machine Learning "experiment" that will generate the estimated outcome by choosing the appropriate algorithm for the process.
- Executing the Training, Scoring and Evaluation processes to build the predictive model.
- Deploying a REST service that external applications can consume to obtain the predicted outcome "as a service." This is pure Machine Learning as a Service (MLaaS).

Let's go through these steps in detail.

Dataset

To develop and train a predictive analytics solution in Azure Machine Learning Studio, it's necessary to import a dataset to analyze. There are several options for importing data and using it in a machine learning experiment:

1. Upload a local file. This is a manual task. Azure Machine Learning Studio, at the time of this writing, supports the upload of .csv and .tsv files, plain text (.txt), SVM Light (.svmlight), Attribute Relation File Format (.arff), R Object or Workspace (.RData), and .zip files.
2. Access data from an existing data source. The currently supported data sources are a Web URL using HTTP, Hadoop using HiveQL, Azure Blob or Table storage, Azure SQL Database or SQL Server on Azure Virtual Machine, on-premises SQL Server database, and any OData feed.
3. From another Azure Machine Learning experiment saved as a dataset.

More information about importing training data into Azure Machine Learning Studio from various data sources is available at bit.ly/2pXhgus.

Experiment

Once the dataset is available, and assuming you call it "cache hits," it's possible to build the machine learning experiment that will analyze data, identify patterns of usage of objects in cache and forecast the future demand. This estimation, expressed as a number of hits under certain conditions, will be used for populating the L2 cache in Redis.

Building a new experiment consists of defining a flow of tasks that are executed by the machine learning engine. In its more linear representation, a demand estimation experiment includes the following steps:

- Add the dataset as the starting point of the experiment. On the left-hand panel, existing datasets are available under Saved Datasets | My Datasets.
- Optionally, apply a transformation to the existing dataset. This can be done for several reasons, including data cleansing or improving data accuracy. You'll use the Select Columns in Dataset (bit.ly/2qTUQZ3) and the Split Data (bit.ly/2povZKP) modules to reduce the number of columns to analyze, and divide your dataset into two distinct sets. This is a necessary step when you want to separate data into training and testing sets, so that you can evaluate a model on a holdout dataset. The two modules are available under the Data Transformation section.
- You can now train the model on the first split of the dataset by adding the Train Model (bit.ly/2qU2JON) module as a next step in the experiment flow, and connect it to the left-hand side connection point of the Split Data module (the fraction of rows to consider for training). The Train Model module can be found under Machine Learning | Train. In the Column Set option, select the label column in the training dataset. The column should contain the known values for the class or outcome you want to predict. This is a numeric data type, in the example project the cache hits by object, on a given day.
- Training a model requires to connect and configure one of the classifications or regression models provided in Azure Machine Learning Studio. You'll use the Linear Regression (bit.ly/2qj01ol) module available in the Machine Learning | Initialize Model | Regression section.

Linear Regression

When a value is predicted, as with cache hits, the learning process is called "regression." Training a regression model is a form of supervised machine learning (bit.ly/2qj01ol). That means you must provide a dataset that contains historical data from which to learn patterns. The data should contain both the outcome you're trying to predict, and related factors (variables). The machine learning model uses the data to extract statistical patterns and build a model.

Regression algorithms (bit.ly/2qj6uiX) are algorithms that learn to predict the value of a real function for a single instance of data.

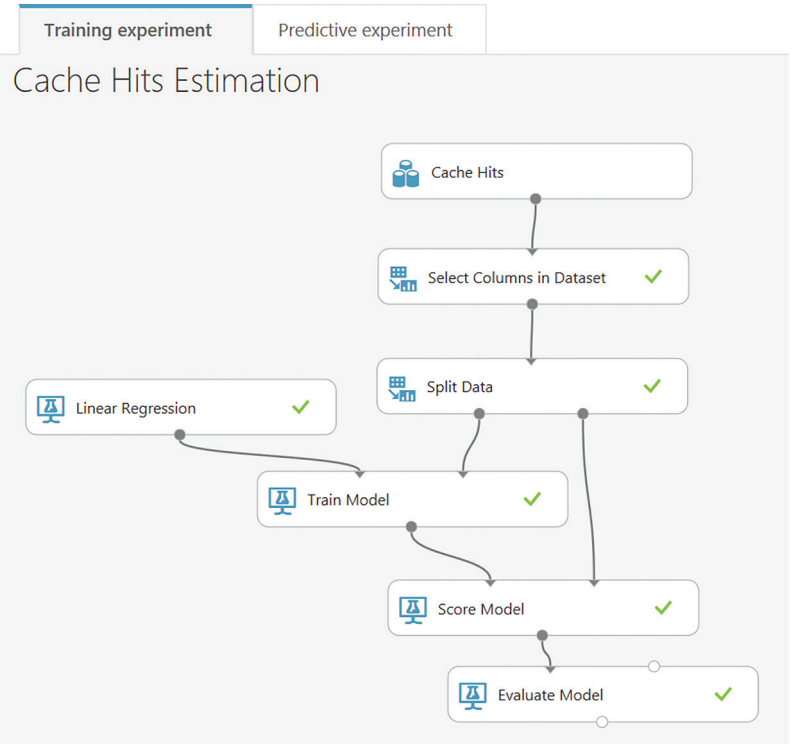


Figure 7 **Cache Hits Estimation Experiment**

GROUPDOCS
Document Manipulation APIs

Manipulating Files?

APIs to view, export, annotate, compare, sign, automate and search documents in your applications.

GroupDocs.Total

GroupDocs.**Viewer**
GroupDocs.**Annotation**
GroupDocs.**Conversion**
GroupDocs.**Comparison**
GroupDocs.**Signature**
GroupDocs.**Assembly**
GroupDocs.**Metadata**
GroupDocs.**Search**
GroupDocs.**Text**

Try for Free

.NET Libraries

Java Libraries

Cloud APIs

Contact Us:
US: +1 903 306 1676
EU: +44 141 628 8900
AU: +61 2 8006 6987
sales@asposeptyltd.com

Visit us at www.groupdocs.com

GROUPDOCS
Document Manipulation APIs

Figure 8 **Request Body of the Machine Learning Service**

```
{
  "Inputs": {
    "inputData": {
      "ColumnNames": [
        ...
      ],
      "Values": [
        [
          ...
        ],
        [
          ...
        ]
      ]
    }
  },
  "GlobalParameters": {}
}
```

Figure 9 **Response Body of the Machine Learning Service**

```
{
  "Results": {
    "outputData": {
      "type": "DataTable",
      "value": {
        "ColumnNames": [
          "Scored Labels"
        ],
        "ColumnTypes": [
          "Numeric"
        ],
        "Values": [
          [
            "0"
          ],
          [
            "0"
          ]
        ]
      }
    }
  }
}
```

These algorithms can incorporate input from multiple features by determining the contribution of each feature of the data to the regression function. Once the regression algorithm has trained a function based on already labeled data, the function can be used to predict the label of a new (unlabeled) instance. More information on how to choose algorithms for Azure Machine Learning is available at bit.ly/2gs06PE.

Scoring and Evaluation

"Scoring" is the process of applying a trained model to new data to generate predictions and other values. The Score Model module (bit.ly/1lpX2Ed), available in the Machine Learning | Score | Score Model section, will predict the number of cache hits according to the selected features, as shown in **Figure 7**.

Figure 10 **Consuming the Machine Learning Service in the Microsoft .NET Framework**

```
using (var client = new HttpClient())
{
  var scoreRequest = new
  {
    Inputs = new Dictionary<string, StringTable>() {
      {
        "inputData",
        new StringTable()
        {
          ColumnNames = new string[] { "Date", "Object", "Hits" },
          Values = new string[,] {  { "YYYY/MM/DD", "GUID", "#" }  }
        }
      },
    },
    GlobalParameters = new Dictionary<string, string>()
    {
    }
  };

  client.DefaultRequestHeaders.Authorization =
    new AuthenticationHeaderValue("Bearer", apiKey);
  client.BaseAddress = new Uri(serviceEndpoint);

  HttpResponseMessage response =
    await client.PostAsJsonAsync(string.Empty, scoreRequest);

  if (response.IsSuccessStatusCode)
  {
    string result = await response.Content.ReadAsStringAsync();
  }
}
```

After the Scoring step, you can now connect the scored dataset to the Evaluate Model module (bit.ly/1SL05By) to generate a set of metrics used for evaluating the model's accuracy (performance). Consider the Evaluation step as your QA process: You want to make sure that the predicted values are as accurate as possible by reducing the amount of error. A model is considered to fit the data well if the difference between observed and predicted values is small. The Evaluate Model module is available in the Machine Learning | Evaluate section.

Service

A prediction of cache hits would be pointless if you couldn't access this information and use it to optimize the pre-allocation of objects in cache. Access to the outcome of the predictive experiment is via a Web service that Azure Machine Learning generates and hosts at a public URL. It's a REST endpoint that accepts a POST request, with an authorization bearer in the header, and a JSON input message in the body.

The authorization bearer is a key that authorizes a client application to the consumer service. The request body contains the input parameters to pass to the service, as specified in the predictive experiment. The format looks like that shown in **Figure 8**.

The service's response is a JSON message containing the scored label, as shown in **Figure 9**.

Using HttpClient for establishing an HTTP connection to the service, it's trivial to access the Web service and read the predicted outcome:

- Input parameters are passed as a collection of strings.
- The API key is assigned a bearer value in the request's header.
- The message is sent to the endpoint as a POST in JSON format.
- The response is read as a string, again in JSON format.

Figure 10 shows the code input for consuming the Machine Learning Service in the Microsoft .NET Framework.

The full source code is available at bit.ly/2qz4gtm.

Wrapping Up

Observing cache hits for objects over several weeks generated a dataset that could be used in a machine learning project to identify access patterns and predict future demand. By exposing a Web service that can be consumed by an external integration workflow (running on Azure Logic Apps, for example), it's possible to obtain predictions of demand on specific objects and pre-allocate them in Redis cache before they're requested in order to minimize miss ratio. The observed improvement was of nearly 20 percent better hit ratio, passing from about 60 percent to almost 80 percent in the L2 cache. This has helped sizing the L2 cache accordingly, and by using the regional syncing capability of Azure Redis Cache, its minimized sync time between distributed nodes by a similar proportion (20 percent shorter duration). ■

Stefano Tempesta *is a Microsoft MVP and technology ambassador, as well as a chapter leader of CRMUG for Italy and Switzerland. A regular speaker at international conferences, including Microsoft Ignite, NDC, API World and Developer Week, Tempesta's interests span across cloud, mobile and Internet of Things. He can be reached via his personal Web site at tempesta.space.*

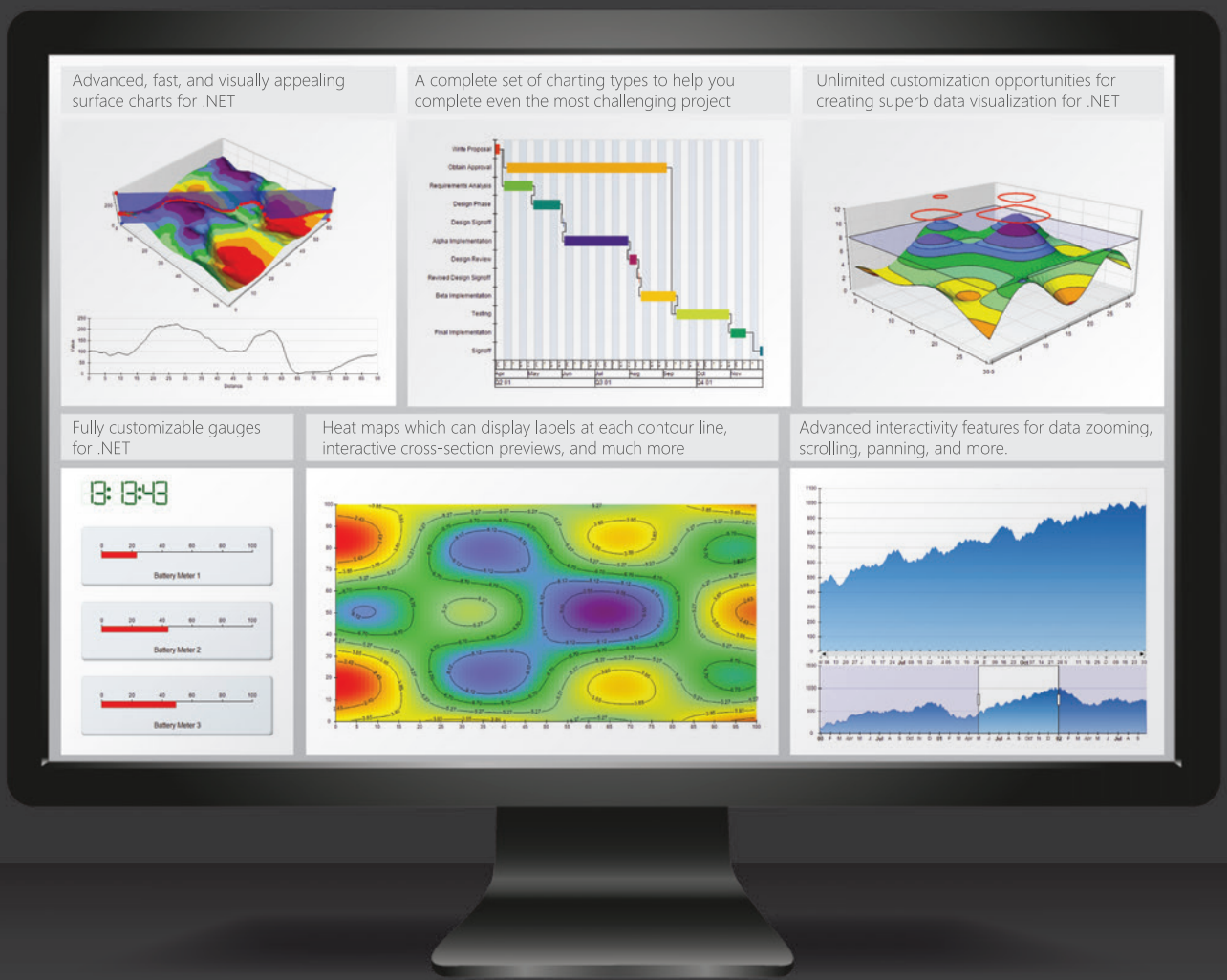
Thanks to the following Microsoft technical expert for reviewing this article: *James McCaffrey*

NEVRON

www.nevron.com

180+ CHART TYPES • 2D & 3D CHARTING TYPES • LINEAR & RADIAL GAUGES • ADVANCED INTERACTIVE FUNCTIONALITY • HARDWARE ACCELERATED RENDERING HUNDREDS OF EXAMPLES WITH SOURCE CODE • PREMIUM SUPPORT

NEVRON CHART FOR .NET



ADVANCED & FAST

Part of NEVRON VISION for .NET

Learn more at **www.nevron.com** today

solution available for

Microsoft ASP.net | Microsoft Silverlight | WPF | WinForms | SharePoint | Microsoft SQl Server | Xamarin

www.nevron.com | email@nevron.com | +1 888-201-6088 (Toll free, USA and Canada)

Visual Studio Partner

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

REDMOND
AUGUST 14-18, 2017
MICROSOFT HEADQUARTERS

JOIN US AT MICROSOFT HEADQUARTERS THIS SUMMER

- Rub elbows with blue badges
- Experience life on campus
- Enjoy lunch in the Commons and visit the Company Store
- Networking event on Lake Washington, Wednesday, August 16
- And so much more!

SUNDAY, AUG 13: PRE-CON HANDS-ON LABS

Choose From:

- Angular
- Dev Ops with ASP.NET Core/EF Core
- SQL Server 2016

NEW! Only $645!

SPACE IS LIMITED

EVENT PARTNER
Microsoft

GOLD SPONSORS
smartsheet
Syncfusion

SUPPORTED BY
Visual Studio
msdn magazine
Visual Studio MAGAZINE

PRODUCED BY
1105 MEDIA
YOUR GROWTH. OUR BUSINESS.

INTENSE TRAINING FOR DEVELOPERS, ENGINEERS, PROGRAMMERS, ARCHITECTS AND MORE!

Development Topics include:

- AngularJS
- ASP.NET Core
- Azure
- Analytics
- DevOps
- .NET Framework
- Software Practices
- SQL Server
- Visual Studio 2017
- Web API
- UWP
- Xamarin

TURN THE PAGE FOR FULL AGENDA DETAILS

Register by July 14 and Save $300*

Use promo code VSLRED2

*Savings based on 5 day packages only.

REGISTER NOW

ROCK YOUR CODE TOUR 2017

REDMOND
AUG 14-18

"I liked that there was representation of Android and iOS, as well as, Microsoft. I prefer working with Microsoft tech/code but can't ignore Android and iOS." *– Chris Nacey, Site Crafting*

"This is the first conference that I have attended @Microsoft Headquarters and it truly has been a fantastic experience. Definitely exceeded my expectations and I look forward to coming back." *– David Campbell, G3 Software*

vslive.com/redmondmsdn

CONNECT WITH US

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

linkedin.com – Join the "Visual Studio Live" group!

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

REDMOND
AUGUST 14-18, 2017
MICROSOFT HEADQUARTERS

Explore the Microsoft Campus during Visual Studio Live! Redmond 2017!

Scott Hanselman, Keynote Speaker

START TIME	END TIME
8:00 AM	9:00 AM
9:00 AM	6:00 PM

START TIME	END TIME
7:00 AM	8:00 AM
8:00 AM	12:00 PM
12:00 PM	2:00 PM
2:00 PM	5:30 PM
7:00 PM	9:00 PM

START TIME	END TIME
7:30 AM	8:00 AM
8:00 AM	9:15 AM
9:30 AM	10:45 AM
10:45 AM	11:15 AM
11:15 AM	12:15 PM
12:15 PM	1:30 PM
1:30 PM	2:45 PM
3:00 PM	4:15 PM
4:15 PM	5:45 PM

START TIME	END TIME
7:30 AM	8:00 AM
8:00 AM	9:15 AM
9:30 AM	10:45 AM
11:00 AM	12:00 PM
12:00 PM	1:30 PM
1:30 PM	2:45 PM
2:45 PM	3:15 PM
3:15 PM	4:30 PM
6:15 PM	8:30 PM

START TIME	END TIME
7:30 AM	8:00 AM
8:00 AM	9:15 AM
9:30 AM	10:45 AM
11:00 AM	12:15 PM
12:15 PM	2:15 PM
2:15 PM	3:30 PM
3:45 PM	5:00 PM

START TIME	END TIME
7:30 AM	8:00 AM
8:00 AM	5:00 PM

MICROSOFT SET LIST: DETAILS DROPPING SOON!

With all of the announcements sure to come out of Build, we'll be finalizing the FULL Track of Microsoft-led sessions shortly.

Microsoft Speakers are noted with a

Be sure to check vslive.com/redmondmsdn for session updates!

ROCK YOUR CODE TOUR 2017

Register by July 14 and Save $300*

Use promo code VSLRED2

*Savings based on 5 day packages only.

REGISTER NOW

EVENT PARTNER
Microsoft

GOLD SPONSORS
smartsheet
Syncfusion

SUPPORTED BY
Visual Studio
msdn magazine
Visual Studio MAGAZINE

PRODUCED BY
1105 MEDIA
YOUR GROWTH. OUR BUSINESS.

REDMOND AGENDA AT-A-GLANCE

Microsoft Set List	ALM / DevOps	Cloud Computing	Database and Analytics	Native Client	Software Practices	Visual Studio / .NET Framework	Web Client	Web Server

NEW Full Day Hands-On Labs: Sunday, August 13, 2017 *(Separate entry fee required)*

Pre-Conference HOL Workshop Registration - Coffee and Morning Pastries

HOL01 Full Day Hands-On Lab: Busy Developer's HOL on Angular - *Ted Neward*	**HOL02** Full Day Hands-On Lab: DevOps with ASP.NET Core and EF Core - *Benjamin Day & Brian Randell*	**HOL03** Full Day Hands-On Lab: Developer Dive into SQL Server 2016 - *Leonard Lobel*

Visual Studio Live! Pre-Conference Workshops: Monday, August 14, 2017 *(Separate entry fee required)*

Pre-Conference Workshop Registration - Coffee and Morning Pastries

M01 Workshop: Modern Security Architecture for ASP.NET Core - *Brock Allen*	**M02** Workshop: Distributed Cross-Platform Application Architecture - *Jason Bock & Rockford Lhotka*	**M03** Workshop: Big Data, BI and Analytics on The Microsoft Stack - *Andrew Brust*

Lunch @ The Mixer - Visit the Microsoft Company Store & Visitor Center

M01 Workshop Continues - *Brock Allen*	**M02** Workshop Continues - *Jason Bock & Rockford Lhotka*	**M03** Workshop Continues - *Andrew Brust*

Dine-A-Round Dinner

Visual Studio Live! Day 1: Tuesday, August 15, 2017

Registration - Coffee and Morning Pastries

T01 Go Mobile With C#, Visual Studio, and Xamarin - *James Montemagno*	**T02** Angular 101: Part 1 - *Deborah Kurata*	**T03** New SQL Server 2016 Security Features for Developers - *Leonard Lobel*	**T04** What's New in Visual Studio 2017 - *Robert Green*	**T05** Microsoft Set List: Details Dropping Soon
T06 Building Connected and Disconnected Mobile Apps - *James Montemagno*	**T07** Angular 101: Part 2 - *Deborah Kurata*	**T08** No Schema, No Problem! Introduction to Azure DocumentDB - *Leonard Lobel*	**T09** Getting to the Core of .NET Core - *Adam Tuliper*	**T10** Microsoft Set List: Details Dropping Soon

Sponsored Break - Visit Exhibitors

KEYNOTE: Microsoft's Open Source Developer Journey – *Scott Hanselman, Principal Community Architect for Web Platform and Tools, Microsoft*

Lunch - Visit Exhibitors

T11 Take the Tests: Can You Evaluate Good and Bad Designs? - *Billy Hollis*	**T12** Assembling the Web - A Tour of WebAssembly - *Jason Bock*	**T13** Unit Testing & Test-Driven Development (TDD) for Mere Mortals - *Benjamin Day*	**T14** To Be Announced	**T15** Microsoft Set List: Details Dropping Soon
T16 A Developers Introduction to HoloLens - *Billy Hollis & Brian Randell*	**T17** To Be Announced	**T18** Spans, Memory, and Channels - Making .NET Code Fast - *Jason Bock*	**T19** Entity Framework Core for Enterprise Applications - *Benjamin Day*	**T20** Microsoft Set List: Details Dropping Soon

Microsoft Ask the Experts & Exhibitor Reception—Attend Exhibitor Demos, *Sponsored by* smartsheet

Visual Studio Live! Day 2: Wednesday, August 16, 2017

Registration - Coffee and Morning Pastries

W01 Roll Your Own Dashboard in XAML - *Billy Hollis*	**W02** Migrating to ASP.NET Core - A True Story - *Adam Tuliper*	**W03** Hacker Trix - Learning from OWASP Top 10 - *Mike Benkovich*	**W04** Distributed Architecture: Microservices and Messaging - *Rockford Lhotka*	**W05** Microsoft Set List: Details Dropping Soon
W06 Customizing Your UI for Mobile Devices: Techniques to Create a Great User Experience - *Laurent Bugnion*	**W07** User Authentication for ASP.NET Core MVC Applications - *Brock Allen*	**W08** From Containers to Data in Motion, Tour d'Azure 2017 - *Mike Benkovich*	**W09** Agile: You Keep Using That Word... - *Philip Japikse*	**W10** Microsoft Set List: Details Dropping Soon

General Session: To Be Announced

Birds-of-a-Feather Lunch - Visit Exhibitors

W11 Building Cross-platform App. Dev. with CLSA.NET - *Rockford Lhotka*	**W12** Securing Web APIs in ASP.NET Core - *Brock Allen*	**W13** Tactical DevOps with VSTS - *Brian Randell*	**W14** Agile Failures: Stories from The Trenches - *Philip Japikse*	**W15** TypeScript and the Future of JavaScript - *Jordan Matthiesen & Bowden Kelly*

Sponsored Break - Exhibitor Raffle @ 2:55 pm (Must be present to win)

W16 Building Truly Universal Applications with Windows, Xamarin and MVVM - *Laurent Bugnion*	**W17** Integrating AngularJS & ASP.NET MVC - *Miguel Castro*	**W18** Get Started with Git and GitHub - *Robert Green*	**W19** SOLID – The Five Commandments of Good Software - *Chris Klug*	**W20** Using Angular 2, JavaScript, and TypeScript to Build Fast and Secure Mobile Apps - *Jordan Matthiesen*

Set Sail! VSLive!'s Seattle Sunset Cruise - Advanced Reservation & $10 Fee Required

Visual Studio Live! Day 3: Thursday, August 17, 2017

Registration - Coffee and Morning Pastries

TH01 Lessons Learned from Real World Xamarin.Forms Projects - *Nick Landry*	**TH02** Build Real-Time Websites and Apps with SignalR - *Rachel Appel*	**TH03** "Aurelia vs "Just Angular" a.k.a "The Framework Formerly Known as Angular 2" - *Chris Klug*	**TH04** Go Serverless with Azure Functions - *Eric D. Boyd*	**TH05** Microsoft Set List: Details Dropping Soon
TH06 Creating Great Looking Android Applications Using Material Design - *Kevin Ford*	**TH07** Hard Core ASP.NET Core - *Rachel Appel*	**TH08** Database Lifecycle Management and the SQL Server Database - *Brian Randell*	**TH09** Breaking Down Walls with Modern Identity - *Eric D. Boyd*	**TH10** Microsoft Set List: Details Dropping Soon
TH11 Software Engineering in an Agile Environment - *David Corbin*	**TH12** Enriching MVC Sites with Knockout JS - *Miguel Castro*	**TH13** Power BI: Analytics for Desktop, Mobile and Cloud - *Andrew Brust*	**TH14** Bots are the New Apps: Building Bots with ASP.NET WebAPI & Language Understanding - *Nick Landry*	**TH15** Microsoft Set List: Details Dropping Soon

Lunch @ The Mixer - Visit the Microsoft Company Store & Visitor Center

TH16 Classic Software Design Principles and Why They Are Still Important - *David Corbin*	**TH17** Getting Started with Aurelia - *Brian Noyes*	**TH18** Big Data with Hadoop, Spark and Azure HDInsight - *Andrew Brust*	**TH19** Extend and Customize the Visual Studio Environment - *Walt Ritscher*	**TH20** Microsoft Set List: Details Dropping Soon
TH21 End-to-End Dependency Injection & Testable Code - *Miguel Castro*	**TH22** Securing Client Apps with IdentityServer - *Brian Noyes*	**TH23** Continuous Integration and Deployment for Mobile using Azure Services - *Kevin Ford*	**TH24** Windows Package Management with NuGet and Chocolatey - *Walt Ritscher*	**TH25** Microsoft Set List: Details Dropping Soon

Visual Studio Live! Post-Conference Workshops: Friday, August 18, 2017 *(Separate entry fee required)*

Post-Conference Workshop Registration - Coffee and Morning Pastries

F01 Workshop: Building Modern Web Apps with Azure - *Eric D. Boyd*	**F02** Workshop: Data-Centric Single Page Apps with Aurelia, Breeze, and Web API - *Brian Noyes*

Speakers and sessions subject to change

vslive.com/redmondmsdn

CONNECT WITH US

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

linkedin.com – Join the "Visual Studio Live" group!

ASP.NET Core with Angular, Web API and Azure DocumentDB

Chander Dhall

With more people migrating to .NET Core and from AngularJS 1.x to Angular 2+, these technologies have become essential, along with other Microsoft technologies in Web API and Azure DocumentDB.

The Yeoman generator aspnetcore-spa (bit.ly/2q10dFn) supports JavaScript frameworks such as Angular, React, and VueJS, as well as other features such as Webpack (bit.ly/2osyQXJ) and Hot module replacement (bit.ly/2oKsKNs). Yeoman was used to start the project I'll be discussing in this article. Note that the current generator defaults to a Visual Studio 2017 project.

The code discussed in this article can be found at bit.ly/2q1aSzR. There, you'll find a basic shopping cart implemented using ASP.NET Core, Angular, Web API and Azure DocumentDB.

This article discusses:

- Brief overview of Azure DocumentDB and its features
- Building block technologies that when used together can create enterprise-level applications
- Detailed look into constructing a Web API Controller
- Detailed look into the Angular JavaScript Framework, its core foundation and syntatical features

Technologies discussed:

Web API, Angular, Azure DocumentDB, ASP.NET Core

Web API

A solution that tends to get overlooked but plays a crucial role is the Web API. Using Visual Studio, I created an empty API Controller. You'll notice upon creation inheriting from the Controller and a Route attribute is decorating the API Controller class:

```
[Route("api/Carts")]
public class CartsController : Controller {}
```

API Controllers no longer inherit from APIController, but from Controller, just like an MVC Controller. The differences between the old API Controller and MVC Controller are minimal. In fact, in previous versions of MVC, some people used MVC Controllers as Web API Controllers by simply returning a JsonResult in their action.

Attribute Routing has been supported in the past. Scaffolding has it now and is called Route instead of RoutePrefix.

Utilizing Attributes

In your empty controller, you only have the need for some simple CRUD operations, so you'll create a simple GetById action method:

```
public Cart GetById(int id){ return _carts.FirstOrDefault(x => x.Id == id); }
```

Using default Web API routing conventions, you can get Cart with Id=5 by making a call to the route api/Carts/5. This is a commonly used route template constructed by the route prefix defined earlier in your class; this action method took an int only as a parameter. One way you can identify which HTTP verbs are needed to call this method is to look at the prefix of the action method

itself (*Get*ById). Here, the prefix is identified to use a GET call, for the same applies to *Put*CartItem, *Patch*CartItemQuantity, *Delete*CartItem and *Post*Checkout.

Those last few action methods could work, but even if they did, how readable would they be? And more important, if a new person on your team looked only at PostCheckout, he might not know it refers to a POST call to submit your cart for checkout, or if he would interpret the method as called after (or post) you had already checked out. For readability and to make your method more developer-friendly, you'll decorate your method with attributes.

Verb attributes give developers the ability to decorate action methods to specify what HTTP actions can be performed. With the GetById method, you can make it so that only GET calls can access it:

```
[HttpGet]
public Cart GetById(int id){ return _carts.FirstOrDefault(x => x.Id == id); }
```

Attribute Routing takes precedence over naming by convention. For example, if you decide to change the name of the method from GetById to PutById, it would still only allow the HTTP GET verb. Additionally, more than one verb can be applied on a method. In the following example you can see two different ways of doing that. Other than personal preference on how you want to code it, there's no difference between those two implementations:

```
[HttpGet]
[HttpPost]
public Cart FunctionName(int id){ //Implementation }

[HttpGet, HttpPost]
public Cart FunctionName (int id){ //Implementation }
```

Route Attributes help when Web API Methods start to become more complex. It's also good practice to use them on the simple methods, as well, because they'll continue the process of making methods more readable and, at the same time, create more constrained protection from adverse calls.

As a reminder, because the controller is already decorated with a route prefix, you don't need to repeat that at the method level. This information makes the route attribute on your current method simple. The curly brackets in your template identify that a parameter is to be passed into your route. Previously, this was the number 5:

```
[HttpGet]
[Route("{id}")]
public Cart GetById(int id){ return _carts.FirstOrDefault(x => x.Id == id); }
```

The name of the template parameter is important because that tells you that you need to have a parameter on your method with the same name. The following API method is a valid call from "api/Carts/5." The only problem is that cartId will be 0 and not 5 because the value 5 is in a parameter with the name "id":

```
[HttpGet]
[Route("{id}")]
public Cart GetById(int cartId){ return _carts.FirstOrDefault(x => x.Id == id); }
```

API route template parameter constraints help filter which calls get made in a particular method. These constraints are important because you don't want your method doing too much. Web API methods should be simple and bring back specific data, not a bunch of data that the client will then have to filter and dig out for information. By specifying the int constraint "{id: int}" on the template parameter, you specify that this method will only be called by a route that has an integer in this template position. Now, if you decide to create a GetByName method, you know which method will be called for each, and more important, you don't have to put logic into your method to determine if the parameter passed in was an integer or string value.

Route templates can be attached to a method in a few different ways. All of the following code examples will perform the same way:

```
[HttpGet]
[Route("{id: int}")]
public Cart GetById(int cartId){ return _carts.FirstOrDefault(x => x.Id == id); }

[HttpGet, Route("{id: int}")]
public Cart GetById(int cartId){ return _carts.FirstOrDefault(x => x.Id == id); }

[HttpGet("{id: int}")]
public Cart GetById(int cartId){ return _carts.FirstOrDefault(x => x.Id == id); }
```

The Return Type specified doesn't always have to be the actual class/value type you're returning. In many cases, your method will have more flexibility if you return IActionResult.

> Web API methods should be simple and bring back specific data, not a bunch of data that the client will then have to filter and dig out the information.

Your GetById method would actually benefit from this return type. When a cart can't be found, currently null is passed back to the client, but as a success. You don't want your client to have to perform that check, nor do you want your clients to have to determine in their ajax catch if this call had a data problem or a code problem, which could affect any messaging relayed to the user. You pass back built-in result objects, which automatically create the appropriate response message and status code:

```
[HttpGet, Route("{id: int}")]
public IActionResult GetById(int cartId){
  var cart = _carts.FirstOrDefault(x => x.Id == id);
  if (cart != null) return Ok(cart);
  return BadRequest();
}
```

Another added benefit of passing back IActionResult is that it doesn't care what data type you pass back within the result object. If you had a method GetItem, you could return Ok(new ObjectA()) or Ok(new ObjectB()). The type returned within the OkObjectResult here is of different types.

Waiting ... Waiting ... Waiting ...

Your current Web API method is now in good shape, but it isn't streamlined. It's synchronous and still holding onto threads. You need to not only use async/await, but make sure you take in a CancellationToken parameter, as well. Using an asynchronous call lets you have more throughput come into your application, but if a process starts taking a long time and a user refreshes the page, navigates away, or closes the browser, that process still runs and occupies resources that could be allocated for another user. That's why you always should implement a CancellationToken for every asynchronous GET method at a minimum. Adding in a

Figure 1 **Simple Angular Component: ProductsComponent**

```
import { Component } from '@AngularJS/core';
@Component({
  selector: 'products'
  template: `
    <h2 class="intro-header">List of Products</h2>
    <ul>
      <li>I am a product</li>
      <li>I am another product</li>
    </ul>
  `,
  styles: ['.intro-header { color: blue }', 'p {font-size: 10px;}'],
  providers: []
})
export class ProductsComponent {}
```

CancellationToken parameter to an API method won't affect your route template. It will give you a token that you can pass down to an appropriate async call to handle:

```
[HttpGet, Route("{id: int}")]
public async Task<IActionResult> GetByIdAsync(int cartId, CancellationToken token){
  var cart = _cartService.GetByIdAsync(cartId, token);
  if (await cart != null) return Ok(cart);
  return BadRequest();
}
```

Tools

Now that you have the Web API in place, you don't have to build out the UI or your own tool to test the Web API functionality. Along with your unit tests, two tools stand out as great utilities to support your API development: Postman (bit.ly/19MnN02) and Swagger (bit.ly/2p1GeYH).

Web API has other features to make Web API robust and help keep your API methods down to as few lines of code as possible. In your empty controller, you had only the need for some simple CRUD operations, but other APIs you've created have put a message on an Azure Queue, kicked off a back-end process, or fired and forgotten as part of CQRS. The application of using API Controllers is vast and extensive.

Angular

Today's Web APIs are utilized frequently by one of the many JavaScript frameworks that companies use. One of the more popular frameworks is AngularJS. AngularJS provides some great features and lets a development team get up and running quickly. AngularJS was completely rewritten from version 1.x (called AngularJS) to version 2.x (all versions starting from version 2 are now called Angular), and the Angular team recently released version 4.0, which is not a rewrite and is backward-compatible to version 2.x (for the most part it's a seamless upgrade). From a Visual Studio perspective, Angular can be intimidating because the setup isn't as easy as it once was with just JavaScript files. Other technologies it helps to be familiar with are Webpack (bit.ly/2p1Jfs5), TypeScript (bit.ly/2p1LBXN), npm (bit.ly/2pfw316) and Reactive Extensions for JavaScript (rxjs) (bit.ly/2piytxU).

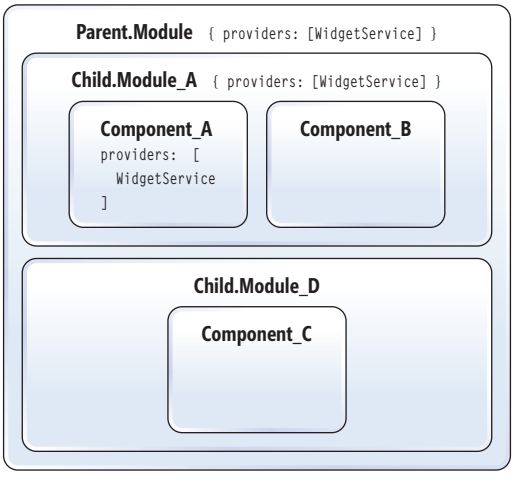


Figure 2 **Hypothetical Layout of Parent Modules, Child Modules and Their Components**

Modules

The building blocks of all Angular applications are Modules. You can compare Modules to an MVC Area. For the most part, it's segregated from the rest of the application, and different developers can work on different modules at the same time with barely any overlap. In the code provided on GitHub, there are three modules of importance: AppModule, ProductsModule and CartsModule. ProductsModule and CartsModule are the two that are isolated-like areas. AppModule is the connector as it puts everything together.

Components

A module itself is typically made up of other modules, components and services. The components declared in each module are where the data and HTML come together.

Through Visual Studio, you create a simple component (products.component.ts) under your Web project (MainSite/ClientApp/app). The initial creation of this component can be seen in **Figure 1**.

Component Decorator

Other than the @Component decorator, this TypeScript looks similar to a C# class with the references at the top and class being declared. The @Component decorator has a required object, which tells Angular how to use this class and lets the component be self-contained.

The selector informs Angular that in any HTML if this CSS selector is used, it should create this component and insert its generated HTML at that location. This also means that if you have HTML code like the following, you'll have three separate instances of each component and three separate classes will be created:

```
<products></products>
<products></products>
<products></products>
```

The template allows HTML to be inside a single file rather than always having to generate an HTML file for every view. As the current template wraps lines, single quotes aren't used. Instead, tick marks are created and it's a good habit to use even if the template is just a single line where single quotes could be used.

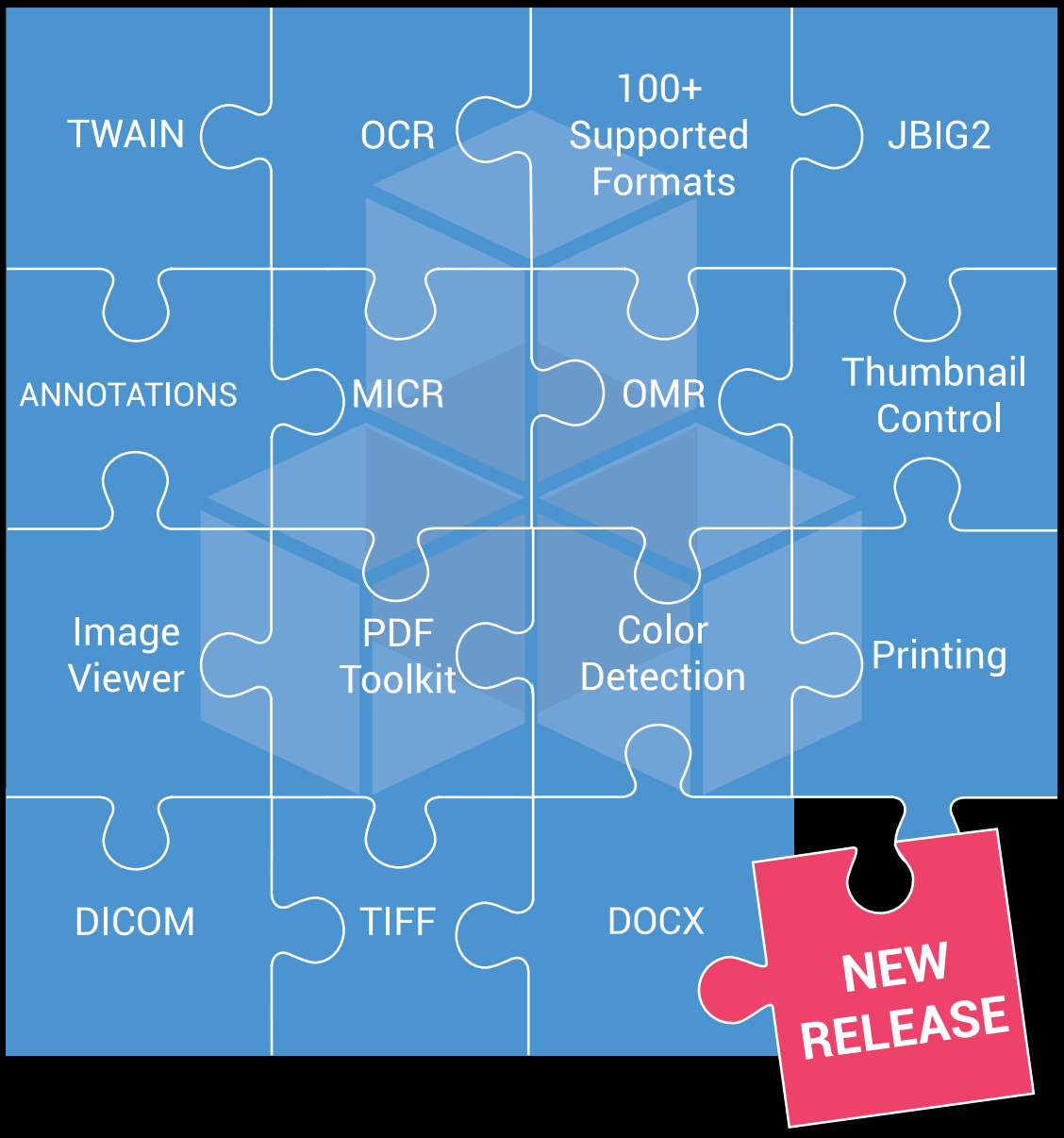
If your HTML gets a little messy, you can always pull your template out into its own file and reference it, but template now becomes templateUrl. The same can be said for styles, except that it's an array of styles or it's an array of references to styleUrls.

Providers is an array where Injectable Services for your component class can be declared. Singleton services would not be declared at the component level. Each time a component gets created, Angular does an inside-out search for the injected services. Looking at **Figure 2**, you'd never realistically create this

GdPicture.NET 14

100% ROYALTY FREE

Imaging SDK For **WinForms**, **WPF** And **Web** Development



Leverage your apps. with **GdPicture.NET** Imaging Toolkit

DOWNLOAD YOUR FREE TRIAL | **www.gdpicture.com**

GdPicture.NET is an ORPALIS imaging technologies product

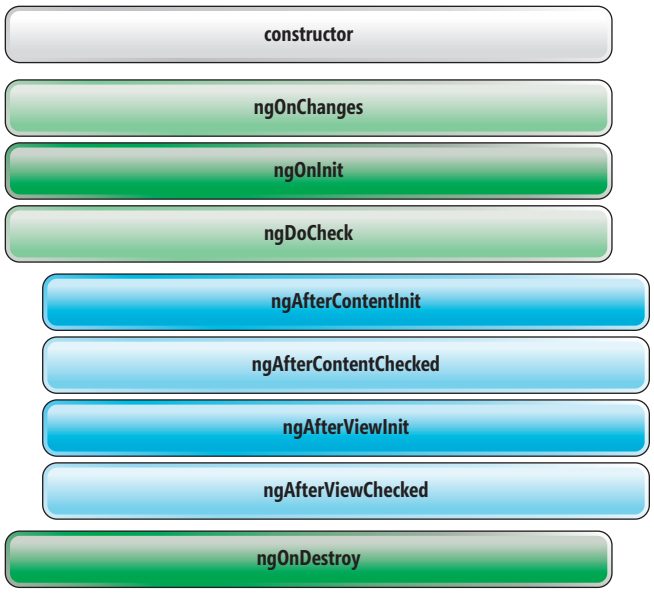


Figure 3 **Angular Lifecycle Hooks Provided by Angular Documentation**

scenario because re-registering providers is typically not a good thing, but this scenario illustrates the point. If Component_A has a WidgetService injected into it, Angular first looks inside Component_A. Identifying that the WidgetService is there, Component_A will get a new instance of WidgetService every time it's created. You could have two Component_A classes each with their own WidgetService in which those two services know nothing about each other. Moving on from there, Angular would find a WidgetService for Component_B at Child.Module_A. This service is a singleton for all components under Child.Module_A, which do not provide their own WidgetService. Last, a WidgetService for Component_C is found at the Parent.Module. A service placed out at the Parent.Module (or AppModule) level is traditionally used as a singleton for the entire Single Page Application.

> Singleton services wouldn't be declared at the component level. Each time a component gets created AngularJS does an inside-out search for the injected services.

Component Class

As mentioned, component classes are very similar to the C# classes with which most developers are familiar. They have constructors where services can be injected, as well as parameters declared inside and outside of methods. One added benefit that Angular provides is lifecycle hooks. **Figure 3** shows a list of all the lifecycle hooks that are able to be tied into. Two of the most popular are ngOnInit and ngOnDestroy.

ngOnInit tends to be the place where initial service calls are made to obtain initial data. This hook is only called once and, more important, any @Input variables have been set at this time.

ngOnDestroy is a great place to unsubscribe from subscriptions that don't self-terminate. If you don't unsubscribe from these subscriptions, it could potentially cause memory leaks.

Template Syntax

HTML Template Syntax provides a way for components to not only affect the look and feel of its own content, but also for components to talk to each other. Following is some of the most commonly used syntax.

Interpolation uses curly braces ({{}}) to extract the content of the object property:

```
@Component({
  template: `
    <div class="row">
      <div class="col-md-12 form-group">
        <label>Product Name:</label>
        <span class="form-control-static">{{product.name}}</span>
      </div>
    </div>
  `
})
export class ProductComponent { product: Product = new Product("Computer ABC"); }
```

One-way binding lets you set values from a component object/property as an attribute of an HTML element. Here, you designate the value as being one way bound by enclosing that attribute with brackets ([]):

```
@Component({
  template: `
    <div class="row">
      <div class="col-md-12 form-group">
        <label>Product Name:</label>
        <input type="text" class="form-control" disabled=""
          [value]="product.name" />
      </div>
    </div>
  `
})
export class ProductComponent { product: Product = new Product("Computer ABC"); }
```

Two-way binding is similar to one-way binding except that as the value of the HTML element changes, the value in the component changes, as well. In **Figure 4**, two-way binding by specifying (ngModel) is designated. In this case, ngModel is used because it's an internal Angular directive for form elements. Because all (boundName) does is comb both one-way binding [] and event binding (), you can use this syntax on any element as long as it supports both the ability to have a value set on boundName and a boundNameChange event.

Figure 4 **Example of Two-Way Binding**

```
@Component({
  template: `
    <div class="row">
      <div class="col-md-12 form-group">
        <label>Product Name:</label>
        <input type="text" class="form-control" [(ngModel)]="product.name" />
        <span class="form-control-static">{{product.name}}</span>
      </div>
    </div>
  `
})
export class ProductComponent { product: Product = new Product(); }
```

Event binding binds a particular action to an HTML element and designates the action upon this event happening. **Figure 5** shows an example of event binding where a (click) event calls handleButtonClick.

*ngIf lets you display content conditionally:

```
@Component({
  template: `
    <p *ngIf="product.name == 'Computer ABC'">Computer name is "Computer ABC"</p>
    <p *ngIf="product.name == 'Computer XYZ'">Computer name is "Computer XYZ"</p>
  `
})
export class ProductComponent { product: Product = new Product("Computer ABC"); }
```

*ngFor provides you the ability to look over an array of objects, as shown in **Figure 6**.

Pipes take your data and manipulates it. Some Angular pipes are | date, | currency, | uppercase. Custom pipes can be created, too. In **Figure 7**, | async is used. This pipe is useful when handling data from an Observable. Products is an Observable, and you don't have to call subscribe on getProducts because that's handled by Angular.

Communication between parent components and child components is interesting in Angular. To help ease that burden, Angular provides @Input and @Output decorators.

@Output decorator provides the opportunity to emit events up a level to the parent, as shown in **Figure 8**.

Likewise, @Input decorators are used to populate a child component from a parent component, as shown in **Figure 9**.

Figure 5 **Example of Event Binding**

```
@Component({
  template: `
    <div class="row">
      <div class="col-md-12 form-group">
        <label>Product Name:</label>
        <span class="form-control-static">{{product.name}}</span>
      </div>
    </div>
    <div class="row">
      <div class="col-md-12">
        <button type="button" class="btn btn-primary"
          (click)="handleButtonClick();">Click Me</button>
      </div>
    </div>
  `
})
export class ProductComponent {
  product: Product = new Product("Computer ABC");
  handleButtonClick(): void { this.product.name = "Comptuer XYZ"; }
}
```

Figure 6 **Example of *ngFor**

```
@Component({
  template: `
    <div class="row" *ngFor="let product of products; let i = index;">
      <div class="col-md-12 form-group">
        <label>Product Name:</label>
        <span class="form-control-static">{{product.name}}
          with Index: {{i}}</span>
      </div>
    </div>
  `
})
export class ProductsComponent {
  products: Product[] = [new Product("Computer ABC"),
    new Product("Computer XYX") ];
}
```

Figure 7 **Example of | async**

```
@Component({
  template: `
    <div class="row" *ngFor="let product of products | async; let i = index;">
      <div class="col-md-12 form-group">
        <label>Product Name:</label>
        <span class="form-control-static">{{product.name}} with Index: {{i}}</span>
      </div>
    </div>
  `
})
export class ProductsComponent implements OnInit {
  products: Observable<Product[]>;
  constructor(private productService: ProductService) { }
  ngOnInit() { this.products = this.productService.getProducts(); }
}
```

Communication between parent components and child components is interesting in Angular.

If a more robust notification is needed, you can build an internal notification system where other components can subscribe to Observables defined by this internal system.

Other IDEs

Angular completely separated from the Visual Studio build process is advantageous because the code itself can be pulled out and used in other editors such as WebStorm (bit.ly/2oxkUeX) and Sublime (bit.ly/1SuiMgd).

Figure 8 **Example of @Output Decorator**

```
@Component({
  template: `
    <div class="row">
      <div class="col-md-12 form-group">
        <label>Button Action:</label>
        <span class="form-control-static">{{buttonAction}}</span>
      </div>
    </div>
    <buttons (buttonClicked)="handleButtonClicked($event)"></buttons>
  `
})
export class ProductsComponent {
  buttonAction: string = "No Button Pressed";
  handleButtonClicked(actionTaken: string): void { this.buttonAction =
    actionTaken; }
}


@Component({
  selector: 'buttons',
  template: `
    <div class="row">
      <div class="col-md-12 form-group">
        <button type="button" class="btn btn-primary"
          (click)="save()">Save</button>
        <button type="button" class="btn btn-default"
          (click)="remove()">Remove</button>
      </div>
    </div>
  `
})
export class ButtonsComponent {
  @Output() buttonClicked = new EventEmitter();
  save() { this.buttonClicked.emit('Save Clicked'); }
  remove() { this.buttonClicked.emit('Delete Clicked'); }
}
```

Figure 9 **Example of @Input Decorator**

```
@Component({
  template: '<product-info [productDetails]="product"></product-info>'
})
export class ProductsComponent { product: Product = new Product("Computer ABC"); }

@Component({
  selector: 'product-info',
  template: `
    <div class='row'>
      <div class="col-md-12 form-group">
        <label>Product Name:</label>
        <span class="form-control-static">{{productDetails.name}}</span>
      </div>
    </div>
  `
})
export class ProductComponent { @Input() productDetails: Product; }
```

DocumentDB

DocumentDB is a fully managed NoSQL database service built for fast and predictable performance, high availability, elastic scaling, global distribution and ease of development. As a schema-free NoSQL database, DocumentDB provides rich and familiar SQL query capabilities with consistent low latencies on JSON data, ensuring that 99 percent of your reads are served in less than 10 ms and 99 percent of your writes are served in less than 15 ms.

DocumentDB transparently replicates your data to all regions you've associated with your DocumentDB account, enabling you to develop applications that require global access to data while providing tradeoffs between consistency, availability and performance, all with corresponding guarantees. DocumentDB provides transparent regional failover with multi-homing APIs, and the ability to elastically scale throughput and storage across the globe.

The core promise of DocumentDB is its service-level agreements (SLAs), which are highly rated in the industry for throughput, latency, availability and consistency. This is a unique offering in the database industry. DocumentDB provides these guarantees to help alleviate the challenges faced by developers who need to develop low latency, distributed, highly available applications.

Even though it may not be related to the application you've built, it's good to know that DocumentDB provides four consistency levels:

Strong: This is an RDBMS-like consistency. With every request, the client is always guaranteed to read the latest acknowledge write. However, this is slow and in order to use this, the DocumentDB account cannot be associated with more than one region.

> DocumentDB provides rich and familiar SQL query capabilities with consistent low latencies on JSON data, ensuring that 99 percent of your reads are served in less than 10 ms and 99 percent of your writes are served in less than 15 ms.

Bounded staleness: This level guarantees that the reads may lag behind by at most x versions of the document or a certain time interval by the client. So, if the client sets x=2, the user will be guaranteed to get a document no later than the last two versions. This functions the same with time. If the time is set to five seconds, every five seconds the resource will be guaranteed to have been

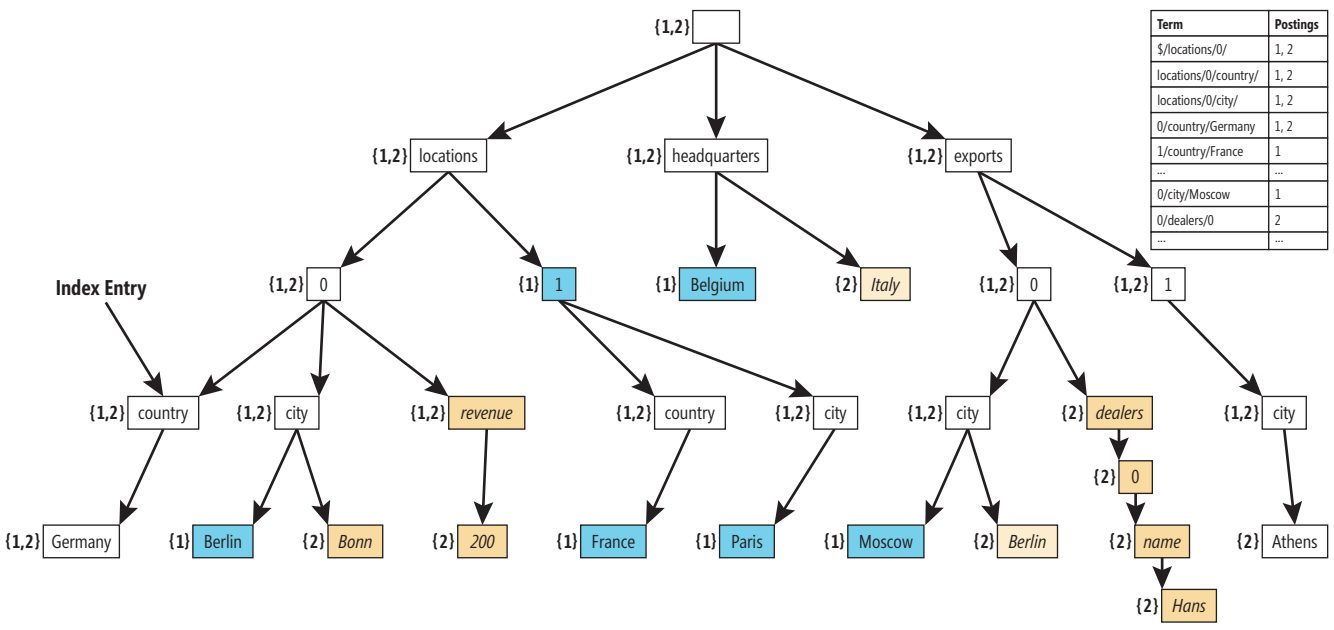


Figure 10 **Representation of the Index for JSON Documents**

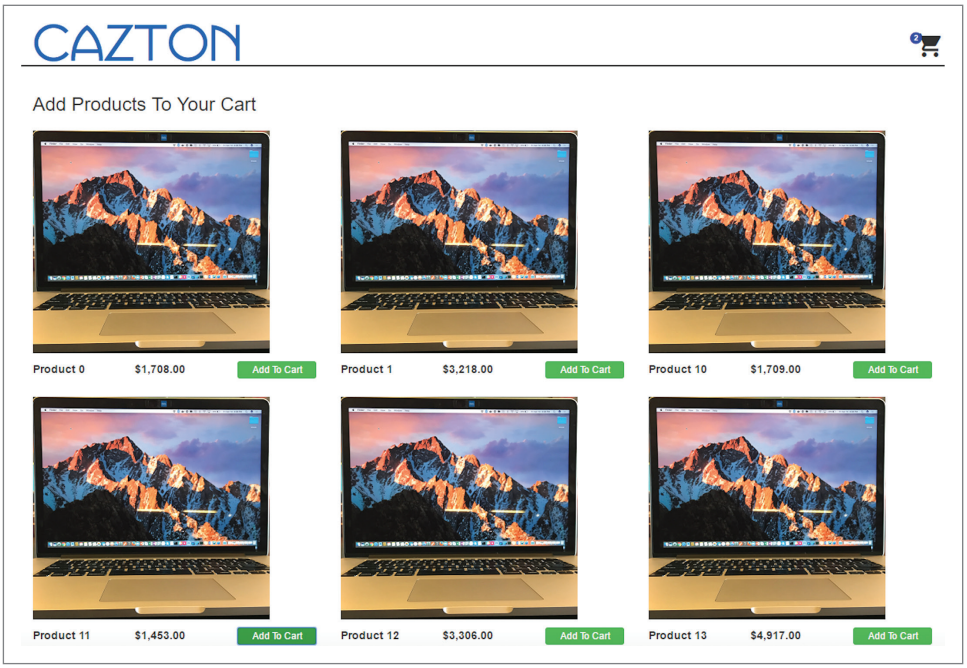


Figure 11 **Output of the GitHub Code, Built Using ASP.NET Core, Angular and DocumentDB**

written to all replicas to make sure that subsequent requests can see the latest version.

Session: This is the most popular of all, and as the name suggests, is scoped to a client session. Imagine someone added a comment on a product on an eCommerce Web site. The user who commented should be able to see it; however, it will take some time before other users on the Web site can see it, too.

Eventual: As the name suggests, the replicas will eventually converge in absence of any additional writes. Eventual consistency provides the weakest read consistency, but offers the lowest latency for both reads and writes.

Architecture

DocumentDB Account has multiple databases. The database can be reached via the Uri [AccountUri]/ dbs/{id}, where AccountUri is of the pattern https://[account].documents.azure.net and whose database has the following collections (collections can be reached with the Uri [AccountUri]/dbs/{id}/colls/{id}):

- Documents—can be reached with the Uri [AccountUri]/ dbs/{id}/colls/{id}/docs/{id}
- Attachments—can be reached with the Uri [AccountUri]/ dbs/{id}/colls/{id}/docs/{id}/attachments/{id}
- Stored Procedures—can be reached with the Uri [AccountUri]/ dbs/{id}/colls/{id}/sprocs/{id}
- Triggers—can be reached with the Uri [AccountUri]/ dbs/{id}/colls/{id}/triggers/{id}
- User Defined Functions—can be reached with the Uri [AccountUri]/ dbs/{id}/colls/{id}/functions/{id}
- Users—can be reached with the Uri [AccountUri]/ dbs/{id}/users/{id}. Users have permissions that can be reached with the Uri [AccountUri]/ dbs/{id}/users/{id}/permissions/{id}

The unit of record is a Document, and a collection is just as the name sounds: a collection of documents. Because documents are flat, it's better to think of them as flat objects and not like rows in a table. Coming from the SQL world, there's a tendency to think of a collection as a table and documents like rows. However, that analogy has more problems than you might realize, especially when it comes to designing the architecture and later implementing it.

DocumentDB automatically handles all aspects of indexing. DocumentDB also supports specifying a custom indexing policy for collections during creation. Indexing policies in DocumentDB are more flexible and powerful than secondary indexes offered in other database platforms because they let you design and customize the shape of the index without sacrificing schema flexibility.

DocumentDB models JSON documents and the indexes as trees, and lets you tune to policies for paths within the tree. You can find more details in an introduction to DocumentDB indexing at bit.ly/2qg2Nqa. Within documents, you can choose which paths must be included or excluded from indexing. This results in improved write performance and lower index storage for scenarios when the query patterns are known beforehand.

In **Figure 10**, you see logical representation of the index for JSON documents.

Additional Info

Testing your query in DocumentDB is sometimes the hardest part of implementing it. Azure provides some ability to run queries through its portal, but I particularly like using DocumentDB Studio (studiodocumentdb.codeplex.com). Also, if you're interested in checking out DocumentDB in more depth, checkout my Channel 9 Video on DocumentDB (bit.ly/2pJ6A2c).

Overall, Angular, Web API and Azure DocumentDB are each a great technology to have in your development arsenal, but combining them together, applications from as simple as hosting blog entries to as complex as an eCommerce Web site can be created. **Figure 11** shows what can quickly be constructed when these three flexible and easy-to-implement technologies are used together. ■

Chander Dhall *is a Microsoft MVP, Azure Advisor, ASP.NET Insider and Web API Advisor, as well as CEO of Cazton Inc. (cazton.com). He's been active in speaking at top technical conferences around the world. Dhall also conducts workshops internationally to train top developers on ASP.NET Core, Angular, TypeScript, databases (both NoSQL and SQL) and many other technologies.*

Thanks to the following Microsoft technical experts who reviewed this article: *Govind Kanshi, Daniel Roth and Steve Sanderson*

TED NEWARD

How To Be MEAN: Angular Ins and Outs

Welcome back, MEANers.

Last month I talked briefly about how to create components that can synthesize data (such as the current year) and display it as part of the component's view. I also briefly experimented with the Angular Command-Line Interface (CLI) tool, "ng," and used that to generate a new application. In this month's column, I'll go further down the Web component path and talk about how to pass information into—and out of—a given Angular component.

In this particular case, as part of my ongoing loose example around tracking speakers and their talks, I'm going to build a small component to track the "upvotes" from users/attendees so that people can rate speakers' talks and offer feedback. I'll call this component the UpvoteComponent; you want the component to obey the following (highly simplified) set of specs:

- The Upvote should initialize the "Upvote" property from the "votes" attribute. Default to 0 if no votes attribute is provided.
- The Upvote should let text between the tags act as the label.
- The Upvote should display an up-arrow character and when the user clicks on the up-arrow, it should increment the vote count.
- The Upvote should, when clicked, also notify interested parties (such as other Angular components) of the new vote count.

If the UpvoteComponent is actually going to use a full MVC approach, then you also need a "model" class, which we'll call Upvote.

In addition, you're going to deliberately structure this UpvoteComponent to use a simplified, miniaturized Model-View-Controller (MVC) approach because this will help structure the application as a whole later.

UpvoteComponent

The first step in any component is to generate the basic scaffolding of the component and the Angular CLI gives you that with a single command-line command: "ng generate component upvote." This will create a series of four files (the same four-part .ts/.cs"/.html/.spec.ts split you got before) in an upvote subdirectory.

Figure 1 **Verifying the Upvote Class**

```
import {Upvote} from './upvote';

describe('Upvote', () => {
  it('should create an instance', () => {
    expect(new Upvote(0)).toBeTruthy();
  });

  it('should remember the votes passed in via the constructor', () => {
    let v = new Upvote(12);
    expect(v.count).toEqual(12);
  });

  it('should increment the vote count by one when incremented', () => {
    let v = new Upvote(12);
    v.increment();
    expect(v.count).toBe(13);
  });
});
```

Upvote

If the UpvoteComponent is actually going to use a full MVC approach, then you also need a "model" class, which we'll call Upvote. This can be scaffolded out using the Angular CLI, as well, by running "ng generate class upvote --spec true." (The parameter at the end asks the CLI to generate a unit-test file for the Upvote class, and we all like to write unit tests, right?) This will create src/app/upvote.ts and src/app/upvote.spec.ts files. The first will be effectively an empty class, waiting for a more intelligent/feature-driven class definition, which you'll promptly provide:

```
export class Upvote {
  constructor(private votes : number) { }

  public get count() { return this.votes; }
  public increment() { this.votes++; }
}
```

(**Note:** We could call this a "Vote" model, because that's what it's really tracking, but it's common in Angular components for the model to directly reflect the component itself, so if we're calling this an "UpvoteComponent," consistency suggests we call it an "Upvote." This is, of course, entirely aesthetic, and teams can—and should—come up with their own naming conventions that make sense to them, at least until the Angular community as a whole has locked some in stone.)

And again, because you're a good, unit-test-loving developer, you'll also set up a quick set of unit tests in the upvote.spec.ts file

Extreme Performance Linear Scalability

NCache™

Distributed Cache

- In-Memory App Data Caching
- ASP.NET Sessions & View State
- Runtime Data Sharing

NosDB

NoSQL Database

- Schema-Free JSON Documents
- Multiple Shards & Data Replication
- Powerful SQL, LINQ, ADO.NET

100% Native .NET

Open Source

sales@alachisoft.com

alachisoft.com

US: +1 (925) 236 3830

to verify that the Upvote class behaves the way it's supposed to, as shown in **Figure 1**.

There's obviously more you can do to test this (Can an Upvote have negative values? Can an Upvote be initialized with non-integer values?), but this covers the basics.

By the way, if you're not already, you can run the Angular test-runner by (again) using the CLI to kick it off in a long-running process by running "ng test." This will fire up a browser window, execute the tests and provide interactive feedback by running the tests every time the source code files change. In many respects, it's even better than a compiler, so long as you're good about writing tests.

Now that you have a model to work with, let's work on the UpvoteComponent itself.

UpvoteComponent Use

It often helps to start from the perspective of how you want to use the component, so let's take a moment and rewrite the AppComponent's view to think about how we'll use the UpvoteComponent:

```
<upvote votes="10" (onIncrement)="upvoted($event)">Current upvotes:</upvote>
```

You have a little bit of everything here: some content between the tags you want to use as part of the UpvoteComponent rendering, an attribute (votes) to initialize the UpvoteComponent internal state and an event that you want to expose to the caller/user of the UpvoteComponent so that they can bind in a local (to the calling component) function that will receive an update every time the user clicks on the UpvoteComponent up-arrow.

It's easiest first to provide the votes attribute syntax; this simply requires that you provide a field in the UpvoteComponent and decorate it using the @Input decorator, so that Angular can discover it and wire up the necessary code to pull the value of votes and store it in that field. However, you're going to do something simultaneously tricky and cool with TypeScript along the way, as shown in **Figure 2**.

Notice that "model" is declared as a private field that uses the TypeScript "intersection type" syntax; this means that the field model can be either a number or an Upvote object. You then test to see if it's an Upvote. If it isn't, then you test whether it's a number, so that regardless of what was passed in, you ultimately end up with an Upvote object holding the data in question. The truly mind-bending part of this code is in the "else" block—we assign "votes" a new Upvote object, initialized with votes. Because TypeScript is doing some deep code analysis on this block, it knows—thanks to the earlier "if" test that determines that votes is not an Upvote type—that it must be a number and, therefore, "this.votes" (temporarily) satisfies the condition that the Upvote constructor requires a number.

Figure 2 **Initializing UpvoteComponent Using TypeScript "Intersection Type" Syntax**

```
export class UpvoteComponent implements OnInit {
  @Input() votes : number | Upvote;

  ngOnInit() {
    if (this.votes instanceof Upvote) {
      // It's already an Upvote
    }
    else if (this.votes == undefined) {
      this.votes = new Upvote(0);
    }
    else {
      this.votes = new Upvote(this.votes);
    }
  }
}
```

Next, notice how we would like to provide a label to the content of the UpvoteComponent, so that it can display the text along with a number (and an up-arrow). To do that, Angular lets you *project* content using the ng-content tag, essentially picking up the content inside the app-upvote tag and dropping it somewhere in the view (defined, remember, in upvote.component.html), like so:

```
<p><ng-content></ng-content> {{votes.count}}  &#x25B2;</p>
```

The Unicode character at the end is the Unicode code point for the up-arrow glyph, and recall that the double-curly-bracket syntax is string interpolation of the enclosed expression. The ng-content tag will then be replaced by whatever the user of the UpvoteComponent specified in the body of the tag, which in this case will be the text Current upvotes.

> The truly mind-bending part of this code is in the "else" block—we assign "votes" a new Upvote object, initialized with votes.

Last, you want to provide a "push" sort of notification system, so that when the user clicks on the up-arrow, the count will increment and then notify interested parties. This is going to require three steps: knowing when the user clicks on the up-arrow (which will require trapping the event in the UpvoteComponent's view), doing the Upvote increment and then pushing that out to the world.

In the first step, you need to trap when the user clicks on the up-arrow, so in the UpvoteComponent view, surround the up-arrow character in a span tag and attach a local method, clicked, to the click event using the Angular event-binding syntax, like so:

```
<p><!-- as before -->  <span (click)="clicked()">&#x25B2;</span></p>
```

This will call a method, clicked, that needs to be defined on the UpvoteComponent class, like so:

```
clicked() {
  (this.votes as Upvote).increment();
  this.onIncrement.emit((this.votes as Upvote).count);
}
```

Notice that because the "votes" field is an intersection type of Upvote or number, you need to explicitly cast it as an Upvote object in order to call the Upvote object's increment method.

The third part is already wired in the clicked method—you use an EventEmitter object to send out a message to any interested parties. You pass the current vote count as the sole event parameter to any parties that subscribe to this event. The EventEmitter is also declared as a field of the UpvoteComponent, but marked with the @Output decorator, like so:

```
export class UpvoteComponent implements OnInit {

  @Output() onIncrement = new EventEmitter<number>();

  // ... as before
}
```

Marking it with @Output means that it's now accessible to interested parties, just as @Input does, but as the names imply, one is for incoming values declared as attributes ("property-binding") and the other as events ("event-binding"). Notice that the field name ("onIncrement") is the name used for the event in the <app-upvote> tag when used. Assuming the AppComponent (which is the root component, the one using the <app-upvote> component) has a method on its class called "upvoted," every time the user clicks the up-arrow, the vote count will increment, and the AppComponent will have its upvoted method called with the parameter value set to the current vote count (after incrementing).

Wrapping Up

Now, permit yourself a moment of congratulations; I've covered a fair amount of ground in this one, so it might take a while to settle in. Take a moment to experiment with the code and make sure the property-binding and event-binding syntaxes make sense, both from the "outside" as well as from the "inside."

It might seem odd that you didn't just use a number for the votes field in the UpvoteComponent; it certainly would be simpler to just track a number, but then a crucial part of the "model" concept gets lost, because now data (the vote count) would be duplicated, once in the UpvoteComponent and once wherever the vote count is tracked outside of the UpvoteComponent. Using the slightly more complicated syntax you saw previously, a user can provide an Upvote object as the parameter to the votes attribute, and the UpvoteComponent can increment it directly without requiring any additional work. In exchange for that bit of complexity buried away inside the component, clients will get a better degree of simplicity (which you'll see as you build out the example further).

> There's still some more to do, but hopefully the component concept is starting to take clearer shape now.

There's still some more to do, but hopefully the component concept is starting to take clearer shape now. You still need to define a SpeakerComponent (and a Speaker model that holds the data for a given speaker), but you can well imagine what it'll look like already, given what's shown here. (I encourage you to take a shot at defining one before my next column comes out, to test yourself.) For now, however, happy coding! ■

Ted Neward *is a Seattle-based polytechnology consultant, speaker and mentor, currently working as the director of developer relations at Smartsheet.com. He has written more than 100 articles, authored and coauthored a dozen books, and works all over the world. Reach him at ted@tedneward.com or read his blog at blogs.tedneward.com.*

Thanks to the following technical expert for reviewing this article: *Ward Bell*

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

CHICAGO
SEPTEMBER 18 - 21, 2017
DOWNTOWN MARRIOTT

INTENSE TRAINING FOR DEVELOPERS, ENGINEERS, PROGRAMMERS, ARCHITECTS AND MORE!

Development Topics Include:

- Visual Studio / .NET Framework
- JavaScript / HTML5 Client
- Native Client
- Software Practices
- Database and Analytics
- Angular JS
- ASP.NET / Web Server
- Agile
- ALM / DevOps
- Cloud Computing
- Windows Client
- Xamarin

Register By July 21 and Save $300!*

Use promo code VSLCH12

REGISTER NOW

GOLD SPONSOR

Suncfusion

SUPPORTED BY

Visual Studio msdn magazine Visual Studio MAGAZINE

PRODUCED BY

1105 MEDIA

CHICAGO AGENDA AT-A-GLANCE

ALM / DevOps	Cloud Computing	Database and Analytics	Native Client	Software Practices	Visual Studio / .NET Framework	Web Client	Web Server

START TIME	END TIME				
		Visual Studio Live! Pre-Conference Workshops: Monday, September 18, 2017 *(Separate entry fee required)*			
7:30 AM	9:00 AM	Pre-Conference Workshop Registration - Coffee and Morning Pastries			
9:00 AM	6:00 PM	**M01** Workshop: Distributed Cross-Platform Application Architecture - *Jason Bock & Rockford Lhotka*	**M02** Workshop: Practical ASP.NET DevOps with VSTS or TFS - *Brian Randell*	**M03** Workshop: SQL Server 2016 for Developers - *Andrew Brust & Leonard Lobel*	
6:45 PM	9:00 PM	Dine-A-Round			
START TIME	**END TIME**	**Visual Studio Live! Day 1: Tuesday, September 19, 2017**			
7:00 AM	8:00 AM	Registration - Coffee and Morning Pastries			
8:00 AM	9:00 AM	KEYNOTE: To Be Announced			
9:15 AM	10:30 AM	**T01** JavaScript for the C# Developer - *Philip Japikse*	**T02** Storyboarding 101 - *Billy Hollis*	**T03** Build Cross-Platform Apps in C# using CSLA .NET - *Rockford Lhotka*	**T04** What's New in Visual Studio 2017 - *Robert Green*
10:45 AM	12:00 PM	**T05** TypeScript: The Future of Front End Web Development - *Ben Hoelting*	**T06** Better, Faster, Automated! Windows App Deployment in Visual Studio Mobile Center - *Ela Malani & Piyush Joshi*	**T07** Roll Your Own Dashboard in XAML - *Billy Hollis*	**T08** What's New in C#7 - *Jason Bock*
12:00 PM	1:00 PM	Lunch - Visit Exhibitors			
1:00 PM	1:30 PM	Dessert Break - Visit Exhibitors			
1:30 PM	2:45 PM	**T09** ASP.NET Core MVC - What You Need to Know - *Philip Japikse*	**T10** Professional Scrum Development Using Visual Studio 2017 - *Richard Hundhausen*	**T11** What's New for Developers in SQL Server 2016 - *Leonard Lobel*	**T12** To Be Announced
3:00 PM	4:15 PM	**T13** User Authentication for ASP.NET Core MVC Applications - *Brock Allen*	**T14** PowerShell for Developers - *Brian Randel*	**T15** What's New in Azure IaaS v2 - *Eric D. Boyd*	**T16** To Be Announced
4:15 PM	5:30 PM	Welcome Reception			
START TIME	**END TIME**	**Visual Studio Live! Day 2: Wednesday, September 20, 2017**			
7:30 AM	8:00 AM	Registration - Coffee and Morning Pastries			
8:00 AM	9:15 AM	**W01** Securing Web APIs in ASP.NET Core - *Brock Allen*	**W02** Professional Software Testing Using Visual Studio 2017 - *Richard Hundhausen*	**W03** Cloud Oriented Programming - *Vishwas Lele*	**W04** Database Development with SQL Server Data Tools - *Leonard Lobel*
9:30 AM	10:45 AM	**W05** Assembling the Web - A Tour of WebAssembly - *Jason Bock*	**W06** Get Started with Git and GitHub - *Robert Green*	**W07** Building Modern Web Apps with Azure - *Eric D. Boyd*	**W08** Tactical DevOps for SQL Server - *Brian Randell*
11:00 AM	12:00 PM	General Session: To Be Announced			
12:00 PM	1:00 PM	Birds-of-a-Feather Lunch - Visit Exhibitors			
1:00 PM	1:30 PM	Dessert Break - Visit Exhibitors - Exhibitor Raffle @ 1:15pm (Must be present to win)			
1:30 PM	2:45 PM	**W09** Tools for Modern Web Development Dev Ops - *Ben Hoelting*	**W10** Go Mobile with C#, Visual Studio, and Xamarin - *James Montemagno*	**W11** Build Awesome AF Apps! - *Rachel Appel*	**W12** Power BI: Analytics for Desktop, Mobile and Cloud - *Andrew Brust*
3:00 PM	4:15 PM	**W13** Get Rid of HTML Tables for Better Mobile Web Applications - *Paul Sheriff*	**W14** Continuous Integration & Deployment for Mobile Apps - *James Montemagno*	**W15** Microservices with Azure Container Service & Service Fabric - *Vishwas Lele*	**W16** Power BI: Beyond the Basics - *Andrew Brust*
4:30 PM	5:45 PM	**W17** Use HTML5/Bootstrap to Build Business UI - *Paul Sheriff*	**W18** Mobilizing your Existing Enterprise Applications - *Nick Landry*	**W19** Busy Developer's Guide to the Google Cloud Platform - *Ted Neward*	**W20** Big Data Solutions in Azure - *David Giard*
7:00 PM	9:00 PM	Visual Studio Live! Evening Event			
START TIME	**END TIME**	**Visual Studio Live! Day 3: Thursday, September 21, 2017**			
7:30 AM	8:00 AM	Web Client			
8:00 AM	9:15 AM	**TH01** I Just Met You, and "This" is Crazy, But Here's My NaN, So Call(Me), Maybe? - *Rachel Appel*	**TH02** PowerApps, Flow, and Common Data Service: Empowering Businesses with the Microsoft Business Application Platform - *Archana Nair*	**TH03** Lessons Learned from Real World Xamarin.Forms Projects - *Nick Landry*	**TH04** Busy Developer's Guide to NoSQL - *Ted Neward*
9:30 AM	10:45 AM	**TH05** Build Object-Oriented Enterprise Apps in JavaScript with TypeScript - *Rachel Appel*	**TH06** PowerApps and Flow Part II: Package, Embed, and Extend Your Applications - *Anousha Mesbah & Pratap Ladhani*	**TH07** Improve Your Retrospective Outcomes with Agile Kaizen - *Angela Dugan*	**TH08** Building Applications with DocumentDb - New Features and Best Practices - *Raj Krishnan*
11:00 AM	12:15 PM	**TH09** What's New in TypeScript? - *Doris Chen*	**TH10** Getting Started with Entity Framework Core - *Jim Wooley*	**TH11** Open Source for Microsoft Developers - *Rockford Lhotka*	**TH12** Introduction to Machine Learning with R - *Raj Krishnan*
12:15 PM	1:15 PM	Lunch			
1:15 PM	2:30 PM	**TH13** Practical Performance Tips and Tricks to Make Your HTML/ JavaScript Faster - *Doris Chen*	**TH14** Getting Your Agile Team Unstuck! Tips and Tricks for Blasting Through Common Setbacks - *Angela Dugan*	**TH15** DI Why? Getting a Grip on Dependency Injection - *Jeremy Clark*	**TH16** The Rise of the Machines - Machine Learning for Developers - *Adam Tuliper*
2:45 PM	4:00 PM	**TH17** Building Powerful Applications with AngularJS 2 and TypeScript - *David Giard*	**TH18** Improving Code Quality with Roslyn Code Analyzers - *Jim Wooley*	**TH19** Unit Testing Makes Me Faster: Convincing Your Boss, Your Co-Workers, and Yourself - *Jeremy Clark*	**TH20** I'm Emotional - Using Microsoft Cognitive Services to Understand the World Around You - *Adam Tuliper*

Speakers and sessions subject to change

vslive.com/chicagomsdn

CONNECT WITH US

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

linkedin.com – Join the "Visual Studio Live" group!

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

ANAHEIM, CA
OCT 16-19, 2017
HYATT REGENCY
A Disneyland® Good Neighbor Hotel

INTENSE TRAINING FOR DEVELOPERS, ENGINEERS, PROGRAMMERS, ARCHITECTS AND MORE!

Development Topics include:

- Visual Studio / .NET
- JavaScript / HTML5
- Angular
- Native Mobile & Xamarin
- Software Practices
- Database and Analytics
- ASP.NET Core
- Web API
- ALM / DevOps
- Cloud Computing
- UWP
- Unit Testing

Register Now and Save $300!*

Use promo code VSLAN1 *SAVINGS BASED ON 4-DAY PACKAGES ONLY.

REGISTER NOW

EVENT PARTNER
Microsoft

SUPPORTED BY
Visual Studio msdn magazine Visual Studio MAGAZINE

PRODUCED BY
1105 MEDIA
YOUR GROWTH. OUR BUSINESS.

ANAHEIM AGENDA AT-A-GLANCE

ALM / DevOps | Cloud Computing | Database and Analytics | Native Client | Software Practices | Visual Studio / .NET Framework | Web Client | Web Server

START TIME	END TIME				
Visual Studio Live! Pre-Conference Workshops: Monday, October 16, 2017 *(Separate entry fee required)*					
7:30 AM	9:00 AM	Pre-Conference Workshop Registration - Coffee and Morning Pastries			
9:00 AM	6:00 PM	**M01** Workshop: Distributed Cross-Platform Application Architecture - *Jason Bock & Rockford Lhotka*	**M02** Workshop: Practical ASP.NET DevOps with VSTS or TFS - *Brian Randell*	**M03** Workshop: Developer Dive into SQL Server 2016 - *Leonard Lobel*	
6:45 PM	9:00 PM	Dine-A-Round			
START TIME	**END TIME**	**Visual Studio Live! Day 1: Tuesday, October 17, 2017**			
7:00 AM	8:00 AM	Registration - Coffee and Morning Pastries			
8:00 AM	9:00 AM	KEYNOTE: To Be Announced			
9:15 AM	10:30 AM	**T01** What's New in TypeScript? - *Doris Chen*	**T02** Go Mobile with C#, Visual Studio, and Xamarin - *James Montemagno*	**T03** To Be Announced	**T04** What's New in Visual Studio 2017 - *Robert Green*
10:45 AM	12:00 PM	**T05** Build Object-Oriented Enterprise Apps in JavaScript with TypeScript - *Rachel Appel*	**T06** Optimizing and Extending Xamarin.Forms Mobile Apps - *James Montemagno*	**T07** What's New for Developers in SQL Server 2016 - *Leonard Lobel*	**T08** To Be Announced
12:00 PM	1:30 PM	Lunch			
1:30 PM	2:45 PM	**T09** Angular 101: Part 1 - *Deborah Kurata*	**T10** Get Started with Git and GitHub - *Robert Green*	**T11** Exploring T-SQL Enhancements: Windowing and More - *Leonard Lobel*	**T12** Microsoft Teams - More Than Just Chat! - *Nedra Allmond*
3:00 PM	4:15 PM	**T13** Angular 101: Part 2 - *Deborah Kurata*	**T14** Do It Again, Faster! Automate Your Windows Deployment Pipeline - *Ela Malani*	**T15** What's New in Azure IaaS v2 - *Eric D. Boyd*	**T16** Open Source for the Microsoft Developer - *Rockford Lhotka*
4:15 PM	5:30 PM	Welcome Reception			
START TIME	**END TIME**	**Visual Studio Live! Day 2: Wednesday, October 18, 2017**			
7:00 AM	8:00 AM	Registration - Coffee and Morning Pastries			
8:00 AM	9:15 AM	**W01** I Just Met You, and "This" is Crazy, But Here's My NaN, So Call(Me), Maybe? - *Rachel Appel*	**W02** Tactical DevOps with VSTS - *Brian Randell*	**W03** Go Serverless with Azure Functions - *Eric D. Boyd*	**W04** What's New in C#7 - *Jason Bock*
9:30 AM	10:45 AM	**W05** Practical Performance Tips and Tricks to Make Your HTML/ JavaScript Faster - *Doris Chen*	**W06** Real World VSTS Usage for the Enterprise - *Jim Szubryt*	**W07** Cloud Oriented Programming - *Vishwas Lele*	**W08** I'll Get Back to You: Understanding Task, Await, and Asynchronous Methods - *Jeremy Clark*
11:00 AM	12:00 PM	General Session: To Be Announced			
12:00 PM	1:30 PM	Birds-of-a-Feather Lunch			
1:30 PM	2:45 PM	**W09** Assembling the Web - A Tour of WebAssembly - *Jason Bock*	**W10** Database Lifecycle Management and the SQL Server Database - *Brian Randell*	**W11** Microservices with Azure Container Service & Service Fabric - *Vishwas Lele*	**W12** Getting Started with Entity Framework Core - *Jim Wooley*
3:00 PM	4:15 PM	**W13** Building Single Page Web Applications Using Aurelia.js and the MVVM Pattern - *Ben Hoelting*	**W14** Getting to SAFe in the Enterprise - *Jim Szubryt*	**W15** Busy Developer's Guide to the Clouds - *Ted Neward*	**W16** Improving Code Quality with Roslyn Code Analyzers - *Jim Wooley*
4:30 PM	05:45 PM	**W17** Securing Angular Apps - *Brian Noyes*	**W17** Building Applications with DocumentDb - New Features and Best Practices - *Raj Krishnan*	**W18** Busy Developer's Guide to the Google Cloud Platform - *Ted Neward*	**W19** Learn to Love Lambdas (and LINQ, Too) - *Jeremy Clark*
7:00 PM	9:00 PM	Visual Studio Live! Evening Event			
START TIME	**END TIME**	**Visual Studio Live! Day 3: Thursday, October 19, 2017**			
7:30 AM	8:00 AM	Registration - Coffee and Morning Pastries			
8:00 AM	9:15 AM	**TH01** ASP.NET Core MVC - What You Need to Know - *Philip Japikse*	**TH02** Tools for Modern Web Development Dev Ops - *Ben Hoelting*	**TH03** The Rise of the Machines - Machine Learning for developers - *Adam Tuliper*	**TH04** Storyboarding 101 - *Billy Hollis*
9:30 AM	10:45 AM	**TH05** Role-Based Security Stinks: How to Implement Better Authorization in ASP.NET and ASP. NET Core - *Benjamin Day*	**TH06** Building Cross-Platform Apps in C# using CSLA .NET - *Rockford Lhotka*	**TH07** Introduction to Machine Learning with R - *Raj Krishnan*	**TH08** Agile: You Keep Using That Word... - *Philip Japikse*
11:00 AM	12:15 PM	**TH09** From Zero to the Web API - *Paul Sheriff*	**TH10** Programming with the Model-View-ViewModel Pattern - *Miguel Castro*	**TH11** I'm Emotional - Using Microsoft Cognitive Services to Understand the World Around You - *Adam Tuliper*	**TH12** Top 10 Ways to Go from Good to Great Scrum Master - *Benjamin Day*
12:15 PM	1:15 PM	Lunch			
1:15 PM	2:30 PM	**TH13** Cortana Everywhere: Speech, Conversation & Skills Development - *Nick Landry*	**TH14** Roll Your Own Dashboard in XAML - *Billy Hollis*	**TH15** Unit Testing T-SQL Code - *Steve Jones*	**TH16** Exposing an Extensibility API for your Applications - *Miguel Castro*
2:45 PM	4:00 PM	**TH17** Securing Web Apps and APIs with IdentityServer - *Brian Noyes*	**TH18** Windows 10 for Developers: Building Universal Apps for 1B Devices - *Nick Landry*	**TH19** A Tour of SQL Server Security Features - *Steve Jones*	**TH20** Real World Applications for Dependency Injection - *Paul Sheriff*

Speakers and sessions subject to change

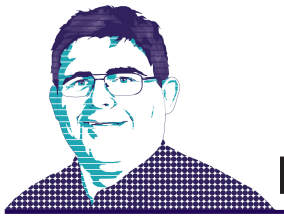
CONNECT WITH US

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

linkedin.com – Join the "Visual Studio Live" group!

vslive.com/anaheimmsdn



Launch Other Applications from Your UWP App

Windows has long provided the capability to programmatically launch applications from within another application. Indeed, it's a feature common to practically any OS. The Universal Windows Platform (UWP) is no exception. In fact, the UWP adds a few new features. In this month's column, I'll explore the Launcher class and how to put it to use in your UWP apps.

In my last two columns, I wrote about exploring the rich functionality of the Bing Maps Control for UWP apps. There might be cases where you don't need or wish to embed a rich mapping solution inside your UWP app. For those scenarios, you can leverage the built-in Maps app. This solution is ideal if you want to provide your users with mapping tools, but don't wish to add complexity to your code base. Accomplishing this is easy with the Launcher class.

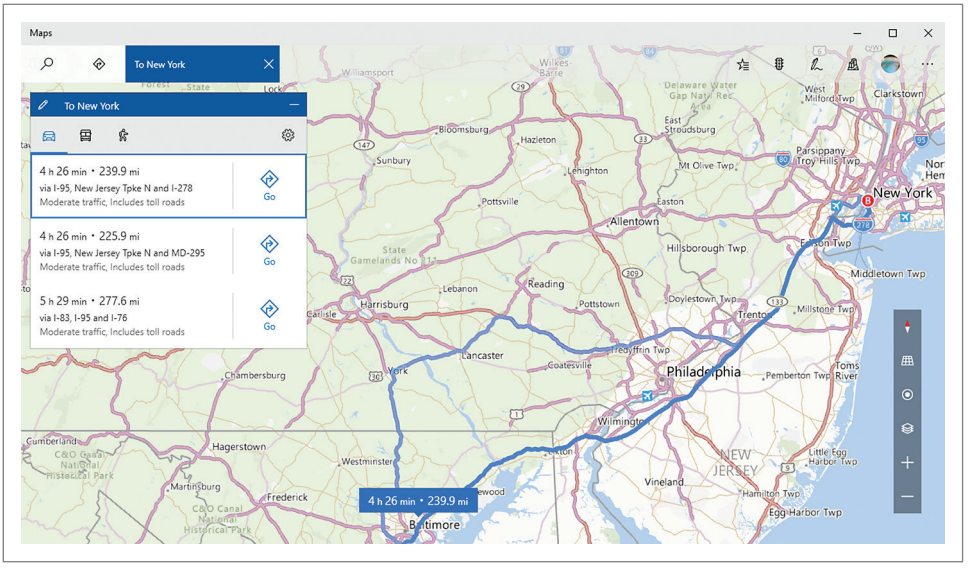


Figure 1 **Driving Directions Between New York City and Washington, D.C., in the Maps App**

Use a Map Without Using a Map Control

Create a new blank UWP project in Visual Studio by choosing New Project from the File menu. Expand the Installed Templates | Windows | Blank App (Universal Windows). Name the project MapLauncher and then click OK. Immediately afterward, a dialog box will appear asking you which version of Windows the app should target. For this project, the default options will be fine, so you can just click OK. In the MainPage.xaml file, add the following XAML to create a button control:

```
<Button Name="btnLaunchMap" Click="btnLaunchMap_Click">Launch Map</Button>
```

In the MainPage.xaml.cs file, add the following code for the event handler:

```
private async void btnLaunchMap_Click(object sender, RoutedEventArgs e)
{
  Uri uri = new Uri("bingmaps:?rtp=adr.Washington,%20DC~adr.New%20York,
    %20NY&amp;mode=d&amp;trfc=1");
  await Launcher.LaunchUriAsync(uri);
}
```

Code download available at bit.ly/2qkp2hz.

Run the solution and click on the button. The Maps app should launch and display driving directions between Washington, D.C., and New York City along with traffic information, as shown in **Figure 1**.

It's quite impressive what can be done in so few lines of code. What, exactly, is going on here that made this so simple and why did the default Maps app automatically launch?

Protocol Launching

Protocol launching enabled launching the Maps app prepopulated with a route. Windows has always relied on file extensions and file-type associations to determine what applications should launch when a file is run. Starting with Windows 8, protocol launching was added to help facilitate launching apps with the option of passing along parameters.

Launching Apps via URI

Taking a closer look at the code in the previous sample, there's something familiar about the line of code that creates a URI. The string passed to the URI constructor method follows a pattern familiar to Web developers, yet with a few differences.

Web addresses or, specifically, URIs, follow a pattern of [protocol]:[host address]?[parameter1]=[value1]&[parameter2]=[value2].

Spreadsheets Made Easy.

Fastest Calculations

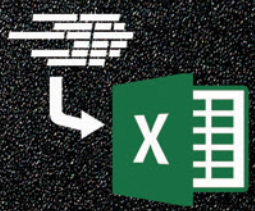
Evaluate complex Excel-based models and business rules with the fastest and most complete Excel-compatible calculation engine available.

Powerful Controls

Add powerful Excel-compatible viewing, editing, formatting, calculating, filtering, sorting, charting, printing and more to your WinForms, WPF and Silverlight applications.

Comprehensive Charting

Enable users to visualize data with comprehensive Excel-compatible charting which makes creating, modifying, rendering and interacting with complex charts easier than ever before.



Scalable Reporting

Easily create richly formatted Excel reports without Excel from any ASP.NET, Windows Forms, WPF or Silverlight application.

Download your free fully functional evaluation at SpreadsheetGear.com

SpreadsheetGear

Toll Free USA (888) 774-3273 | Phone (913) 390-4797 | sales@spreadsheetgear.com

Figure 2 **Default Protocols in Windows**

URI Scheme	Launches
bingmaps:, ms-drive-to:, and ms-walk-to:	Maps app
http:	Default Web browser
mailto:	Default e-mail app
ms-call:	Call app
ms-chat:	Messaging app
ms-people:	People app
ms-settings:	Settings app
ms-store:	Store app
ms-tonepicker:	Tone picker
ms-yellowpage:	Nearby Numbers app

For instance, for the URI http://bing.com/search?q=franksworld, the protocol is HTTP, the host address is bing.com, and the value of "franksworld" is passed to the parameter "q." The same holds true for the following URI, but with a few differences:

```
bingmaps:?rtp=adr.Washington,%20DC~adr.New%20York,%20NY&amp;mode=d&amp;trfc=1
```

Unlike HTTP or HTTPS, the protocol "bingmaps" is probably unfamiliar to most readers. Also, there's no address for the resource. Interestingly, typing the this URI into the Edge browser will have the same effect, launching the Bing Maps app with the route between Washington, D.C., and New York City pre-populated. Additionally, the Run dialog, accessed by Windows key+R, will do the same if that URI is entered. Protocol activation is part of Windows 10 and may be leveraged outside of the UWP. In fact, Windows comes with a series of protocols already defined. The entire list can be found in **Figure 2**. Note that some of the protocols, like ms-call, may only be available on Windows Mobile.

Figure 3 **XAML Code to Create the UI**

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="43*"/>
    <ColumnDefinition Width="137*"/>
  </Grid.ColumnDefinitions>
  <Grid.RowDefinitions>
    <RowDefinition Height="45*"/>
    <RowDefinition Height="44*"/>
    <RowDefinition Height="45*"/>
    <RowDefinition Height="45*"/>
    <RowDefinition Height="461*"/>
  </Grid.RowDefinitions>

  <TextBlock Margin="10" FontSize="24" Grid.RowSpan="5"
    Grid.ColumnSpan="2">Food Finder</TextBlock>
  <TextBlock Grid.Row="1" HorizontalAlignment="Right"
    VerticalAlignment="Center">Genre</TextBlock>
  <TextBlock Grid.Row="2" HorizontalAlignment="Right"
    VerticalAlignment="Center">Location</TextBlock>
  <ComboBox Name="cbxGenre" Grid.Row="1" Grid.Column="1"
    VerticalAlignment="Center" Margin="5" Width="212" SelectedIndex="0">
    <ComboBoxItem>Pizza</ComboBoxItem>
    <ComboBoxItem>BBQ</ComboBoxItem>
    <ComboBoxItem>Coffee</ComboBoxItem>
  </ComboBox>
  <TextBox Name="txtLocation" Grid.Row="2" Grid.Column="1"
    Margin="5"></TextBox>
  <CheckBox Name="ckbTraffic" Grid.Row="3" Grid.Column="1"
    Margin="5">Show Traffic</CheckBox>
  <StackPanel Orientation="Horizontal" Grid.Row="4"
    Grid.ColumnSpan="2" HorizontalAlignment="Center"
    VerticalAlignment="Top">
    <Button Name="btnLaunchMap" Click="btnLaunchMap_Click"
      Margin="5">Launch Map</Button>
    <Button Name="btnSearchMusic" Click="btnSearchMusic_Click"
      Margin="5">Search Music</Button>
  </StackPanel>
</Grid>
```

Passing Parameters

As for the Maps app, there's any number of parameters to pass and the Maps app can be controlled to fit any app's specific scenario. For instance, if the app needs to find pizza places in the Bronx, you'd simply use the following URI:

```
bingmaps:?q=pizza&amp;where=Bronx,NY
```

Food Finder App

To demonstrate how to put parameters and protocol launching to practical use, create a new blank UWP project in Visual Studio. Name the project FoodFinder and click OK to create the app.

Add the XAML shown in **Figure 3** to the MainPage.xaml file.

> As for the Maps app, there's any number of parameters to pass and the Map app can be controlled to fit any app's specific scenario.

In the MainPage.xaml.cs file add the following code for the btnLaunchMap_Click event handler:

```
private async void btnLaunchMap_Click(object sender, RoutedEventArgs e)
{
  string genre = (cbxGenre.SelectedItem as ComboBoxItem).Content.ToString();
  string trafficValue = (ckbTraffic.IsChecked.Value) ? "1" : "0";

  string uriString = string.Format(
    $"bingmaps:?q={genre}&where={txtLocation.Text}&lvl=15&trfc={trafficValue}");
  Uri uri = new Uri(uriString);
  await Launcher.LaunchUriAsync(uri);
}
```

Run the project now. The interface should look similar to what's shown in **Figure 4**. Pick a genre of food, enter a city into the Location textbox, and click Launch Map. The Maps app will launch and display a result set based on the inputs. Naturally, search results will vary based on the genre chosen and location entered. The final result should look something like **Figure 5**.

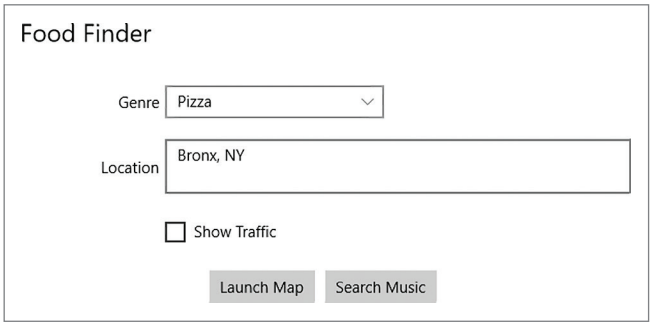


Figure 4 **FoodFinder Interface**

TECHMENTOR

IN-DEPTH TRAINING FOR IT PROS

AUGUST 7 - 11, 2017

MICROSOFT HEADQUARTERS

REDMOND, WA

PLUG IN TO NEW KNOWLEDGE @ THE SOURCE

WHAT SETS TECHMENTOR APART?

+ Immediately usable IT education
+ Training you need today, while preparing you for tomorrow
+ Zero marketing-speak, a strong emphasis on doing more with the technology you already own, and solid coverage of what's just around the corner
+ Intimate setting, where your voice is heard, making it a viable alternative to huge, first-party conferences
+ Experience life @ Microsoft Headquarters for a full week

YOU OWE IT TO YOURSELF, YOUR COMPANY AND YOUR CAREER TO BE AT TECHMENTOR REDMOND 2017!

HOT TRAINING TOPICS INCLUDE:

+ Windows Server + Hyper-V + Windows PowerShell + DSC + DevOps + Azure + Security + And More! +

++

REGISTER NOW

SAVE $300 THROUGH AUGUST 7
MUST USE DISCOUNT CODE TMEB01

[TECHMENTOREVENTS.COM/REDMOND]

EVENT SPONSOR: Microsoft

SUPPORTED BY: Redmond | Redmond Channel Partner | Virtualization Review

GOLD SPONSOR: Goverlan Reach

PRODUCED BY: 1105 Media

Close the Maps app and go back to the FoodFinder app. This time, make sure that the Show Traffic checkbox is checked. Click Launch Map again to see that traffic data is rendered onto the map. If not, then Bing might not have traffic data for that particular location.

Much more can be done with the Maps app. In fact, the full functionality exposed by the Web version of Bing Maps is also accessible to the Maps app and, as a rule, follows the same parameter name and value format. For more information on how to build a Bing Maps URL, see bit.ly/1MIoJ5K.

More Than Maps

While mapping adds significant value to this app, there are many food-finding apps available in the Windows Store. It would be beneficial to differentiate this app by adding a unique feature: music search. For example, if St. Louis were in the Location textbox, users could search for songs about St. Louis and listen to music about their destination. The app will offer users a chance to search for songs with the name of the location in the title. Protocol activation makes it easy to incorporate searches from the Windows Store into apps.

To accomplish this, add the following code to the event handler for the btnSearchMusic_Click event:

```
private async void btnSearchMusic_
Click(object sender, RoutedEventArgs e)
{
  string uriString = string.Format(
    $"ms-windows-store://
    search/?query={txtLocation.
    Text}&type=Songs");
  Uri uri = new Uri(uriString);
  await Launcher.LaunchUriAsync(uri);
}
```

Run the app once again, enter a city name in the Location textbox, and click Search Music. The Windows Store app should launch and display a search of songs with the city name in the title. For instance, if a user entered "St. Louis" as the location, the results would look like what's shown in **Figure 6**.

For more information about the parameters that can be passed to the Windows Store app, refer to the documentation on Windows Dev Center at bit.ly/2pPJvaA.

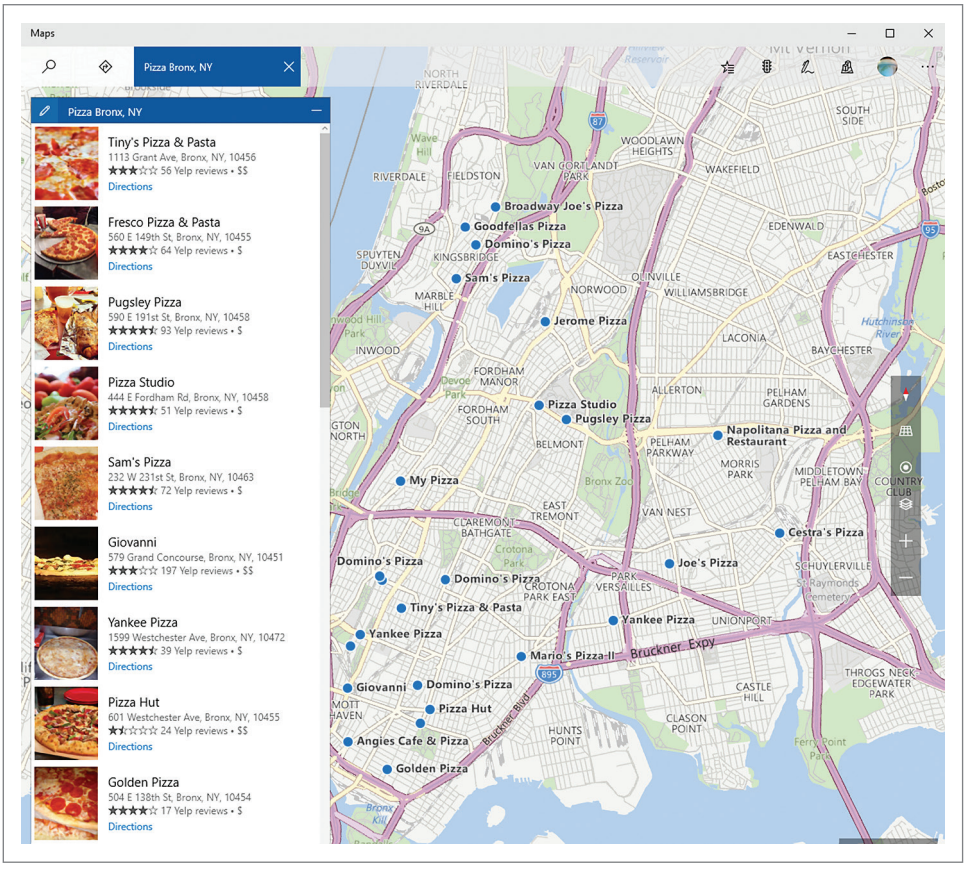
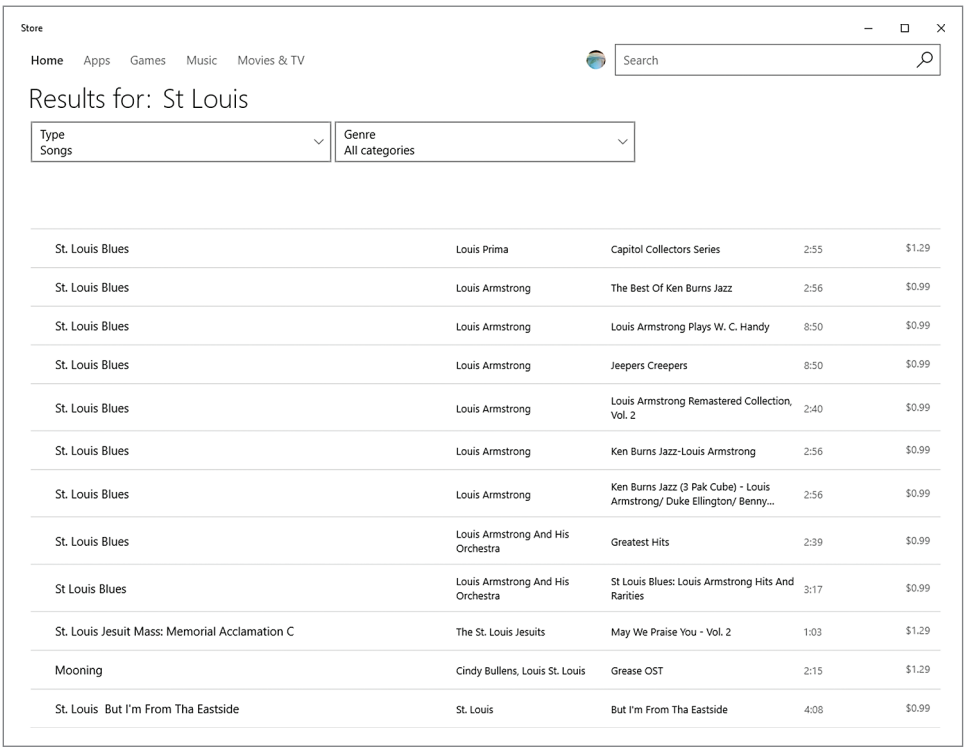


Figure 5 **Results for Pizza in Bronx, N.Y.**



St. Louis Blues	Louis Prima	Capitol Collectors Series	2:55	$1.29
St. Louis Blues	Louis Armstrong	The Best Of Ken Burns Jazz	2:56	$0.99
St. Louis Blues	Louis Armstrong	Louis Armstrong Plays W. C. Handy	8:50	$0.99
St. Louis Blues	Louis Armstrong	Jeepers Creepers	8:50	$0.99
St. Louis Blues	Louis Armstrong	Louis Armstrong Remastered Collection, Vol. 2	2:40	$0.99
St. Louis Blues	Louis Armstrong	Ken Burns Jazz-Louis Armstrong	2:56	$0.99
St. Louis Blues	Louis Armstrong	Ken Burns Jazz (3 Pak Cube) - Louis Armstrong/ Duke Ellington/ Benny...	2:56	$0.99
St. Louis Blues	Louis Armstrong And His Orchestra	Greatest Hits	2:39	$0.99
St Louis Blues	Louis Armstrong And His Orchestra	St Louis Blues: Louis Armstrong Hits And Rarities	3:17	$0.99
St. Louis Jesuit Mass: Memorial Acclamation C	The St. Louis Jesuits	May We Praise You - Vol. 2	1:03	$1.29
Mooning	Cindy Bullens, Louis St. Louis	Grease OST	2:15	$1.29
St. Louis But I'm From Tha Eastside	St. Louis	But I'm From Tha Eastside	4:08	$0.99

Figure 6 **Windows Store App Music Section with St. Louis in Titles of Songs**

Advanced Launch Features

Thus far, the examples in the FoodFinder app had specific outcomes in mind. I had intended to launch the default Maps app and the Windows Store app. In cases where developers want to customize the launching experience or give the user a choice in which app launches, the Launcher class exposes additional features through the use of the LauncherOptions class.

Choosing an Application to Launch

From time to time, users will be presented an option to pick which app to launch when there are multiple apps registered to handle a specific file extension or protocol. For example, it's very common for users to have multiple browsers installed on their systems. In fact, Windows 10 comes with Edge and Internet Explorer. Giving users a choice in which application to launch would be ideal in particular use cases. Fortunately, the LauncherOptions class in the Windows.System namespace makes it easy to customize how the Launcher class acts.

In the MainPage.xaml file for the FoodFinder app, add the following XAML in the StackPanel to add a new button to perform Web searches:

```
<Button Name="btnSearchWeb" Click="btnSearchWeb_Click" Margin="5">Search Web</Button>
```

Now, add the following event handler code in the MainPage.xaml.cs file:

```
private async void btnSearchWeb_Click(object sender, RoutedEventArgs e)
{
  string genre = (cbxGenre.SelectedItem as ComboBoxItem).Content.ToString();
  string queryString = Uri.EscapeDataString($"{genre} in {txtLocation.Text}");
  var searchUri = $"https://www.bing.com/search?q={queryString}";
  var uriBing = new Uri(searchUri);
  var promptOptions = new Windows.System.LauncherOptions();
  promptOptions.DisplayApplicationPicker = true;
  var success = await Windows.System.Launcher.LaunchUriAsync(uriBing, promptOptions);

}
```

Just as before, this code constructs a URI based on the choice in the combo box and value entered into the textbox. The addition of the LauncherOptions class and setting the DisplayApplicationPicker property to true will trigger a prompt similar to **Figure 7**. Run the app now, enter some values and click Search Web to see the Application Picker Prompt.

By default, the DisplayApplicationPicker is set to false, which launches the default browser on the system.

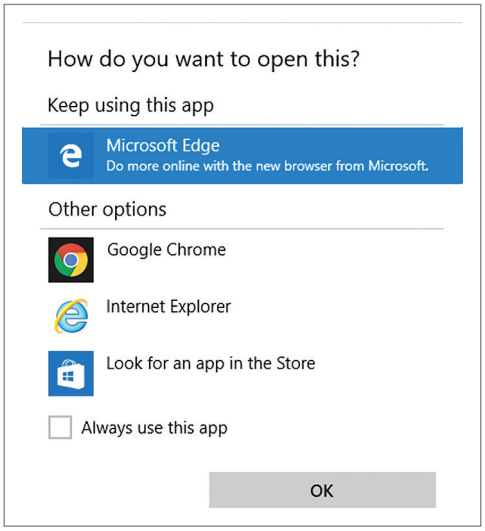


Figure 7 **Application Picker Prompt**

Confirming Launch

It's important to point out that, in certain situations, Windows will display a prompt to confirm if the user actually wishes to switch apps, as shown in **Figure 8**. However, UWP developers cannot suppress this dialog. This is an important security precaution to prevent users from running malicious code.

There might be times when it's advantageous to confirm with the user that the app intends to switch apps. For these cases, set the TreatAsUntrusted property to "true."

Remove the line where the DisplayApplicationPicker property is set to true and add this line to the btnSearchWeb_Click event handler immediately after the declaration of the promptOptions variable:

```
promptOptions.TreatAsUntrusted = true;
```

Run the solution again, enter a value in the Location textbox, and click the Search Web button. You'll see a prompt prior to the browser launching. If the DisplayApplicationPicker and the TreatAsUnstrusted properties are both set to true, then only the application picker prompt will display. This streamlines the process for the user, who would likely get frustrated at the process of clicking through multiple dialog boxes to perform a task.

> The LauncherOptions class exposes much more functionality and provides flexibility for any number of use-case scenarios.

The LauncherOptions class exposes much more functionality and provides flexibility for any number of use-case scenarios. You can find out more at bit.ly/2qoW8LN.

Wrapping Up

In this column, I demonstrated how to use protocol activation in UWP apps to leverage the power of other apps available on the user's device. This lets you add rich features and conveniences without inserting complexity to its code base. The UWP offers great flexibility via the LauncherOptions class, which lets developers customize the UX. ■

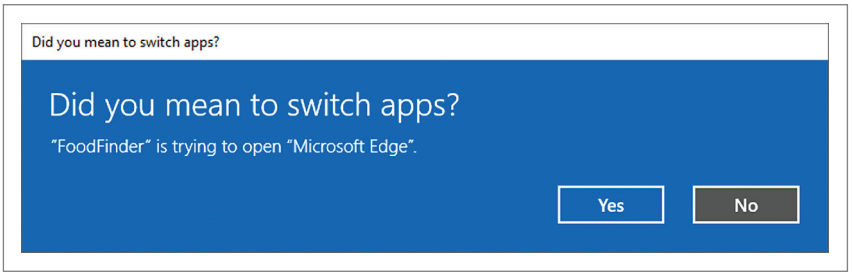


Figure 8 **Prompt Confirming Desire to Switch Apps**

Frank La Vigne *is chief evangelist at DataLeader.io, where he helps customers leverage technology in order to create a better world. He is co-host of the DataDriven podcast and blogs regularly at FranksWorld.com. He has a YouTube channel called Frank's World TV (FranksWorld.TV).*

Thanks to the following technical expert for reviewing this article: *Jose Luis Manners*

The Ultimate Education Destination

2017 Orlando

ROYAL PACIFIC RESORT AT UNIVERSAL ORLANDO
NOVEMBER 12-17

Live! 360: A Unique Conference for the IT and Developer Community

- 6 FULL Days of Training
- 5 Co-Located Conferences
- 1 Low Price
- Create Your Own Agenda from Hundreds of Sessions
- Expert Education and Training
- Knowledge Share and Networking

CONNECT WITH LIVE! 360

twitter.com/live360
@live360

facebook.com
Search "Live 360"

linkedin.com
Join the "Live! 360" group!

EVENT PARTNERS

Microsoft Magenic

PLATINUM SPONSOR

SMARTBEAR

SUPPORTED BY

Visual Studio msdn magazine Redmond Channel Partner

LIVE! 360

TECH EVENTS WITH PERSPECTIVE

NEW: HANDS-ON LABS

Join us for full-day, pre-conference hands-on labs Sunday, November 12.

Only $595 through August 11

5 GREAT CONFERENCES, 1 GREAT PRICE

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

Visual Studio Live!: Code in Paradise at VSLive!™, featuring unbiased and practical development training on the Microsoft Platform.

SQL Server LIVE!

TRAINING FOR DBAs AND IT PROS

SQL Server Live! will leave you with the skills needed to drive your data to succeed, whether you are a DBA, developer, IT Pro, or Analyst.

TECHMENTOR

IN-DEPTH TRAINING FOR IT PROS

TechMentor: This is where IT training meets sunshine, with zero marketing speak on topics you need training on now, and solid coverage on what's around the corner.

Office & SharePoint LIVE!

ON-PREMISE, CLOUD & CROSS-PLATFORM TRAINING

Office & SharePoint Live!: Today, organizations expect people to work from anywhere at any time. Office & SharePoint Live! provides leading-edge knowledge and training to work through your most pressing projects.

Modern Apps LIVE!

MOBILE, CROSS-DEVICE & CLOUD DEVELOPMENT

Modern Apps Live!: Presented in partnership with Magenic, this unique conference delivers how to architect, design and build a complete Modern App.

REGISTER NOW

REGISTER BY AUGUST 11 AND SAVE $500!*

Use promo code L360MAY2

*Savings based on 5-day packages only. See website for details.

PRODUCED BY

Redmond MAGAZINE | VIRTUALIZATION & Cloud Review | Visual Studio MAGAZINE | 1105 MEDIA

LIVE360EVENTS.COM

Live and in Concert

I just got back from Build 2017 in Seattle. The live convention business took some serious knocks during the recession, but Build sells out within minutes, even though all of its content is available online. There's just something special about direct human-to-human interaction with the people who build the software that drives our professional lives, something that no telepresence can ever duplicate. It's the ultimate geek contact high. In no particular order, here are my impressions:

Microsoft should build an automated buzzword bingo game into the conference's smartphone app. For example:

Keynote speaker (droning): "We proactively empower your strategic cyber-aggregation …"

Attendee (leaping to feet, waving phone): "BINGO!"

That would be an impressive demonstration of Microsoft's new Cognitive Services, especially discerning among the differing voices and accents of the various speakers. Is it smart enough to deduce new buzzwords from context? Just make sure you give away a killer prize like a HoloLens, instead of a leftover Lumia phone.

Speaking of which, I was glad not to see Microsoft wasting time on another me-too phone. The conference's scheduling app ran on Android and iPhone, but not on whatever Windows phones still survive, which shows that Microsoft has accepted the market's judgment. To paraphrase John Prine's classic song: "Microsoft, Microsoft, you have no complaint. You are what you are and you ain't what you ain't." To which I add, the company is wise to finally realize it.

I always enjoy hearing Harry Shum, director of Microsoft Research (MSR). MSR has now developed AI to the point where it's easier to use than not.

Microsoft highlighted its augmented/mixed/virtual reality (A/M/VR) solutions, which I found fascinating. The price of hardware is falling, and support is now built into the OS. I think A/M/VR is about to break out of its gaming niche market. But after I tried a few demos, I can confidently report that it hasn't yet reached the level of improving the looks of an ugly blind date (of any gender or preference). You still need alcohol for that.

Microsoft is once again trying to launch pen computing with gestures, this time with the Ink Interface in the Windows 10 Fall Creators Update. The company has been trying this since at least the 1991 Professional Developers Conference (held just down the hill at the 5th Avenue Theatre), at which it released the beta of Windows 3.1. I remember drinking with some programmers from the United States Navy, who said, "Does Microsoft seriously think the admiral is going to learn gestures? The admiral already knows the only gesture he needs, which is (crooking finger), 'Come here, ensign, and fix this crap.'"

One Microsoft booth featured a whiteboard asking for attendees' comments: "If we could fix just one thing immediately, what would it be?" I wrote, "My hemorrhoids." I wonder how that did in triage.

As always, I interviewed as many attendees as I could, getting their take on today's industry. A few recognized my name from my badge. Many recognized this column when I mentioned it: "Oh, you're that genius/idiot?" But not one attendee recognized me by my picture on the page. I have less hair in person, and more of it's gray. I think I need a new one.

I always enjoy hearing Harry Shum, director of Microsoft Research (MSR). MSR has now developed artificial intelligence (AI) to the point where it's easier to use than not. You don't have to be an AI programmer. Just hand it some data sets.

I find this more than a little scary. One keynote demonstration showed cameras surveying a mock construction site, automatically recognizing people and tools, and objecting when the wrong guy picked up a jackhammer. Satya said in his keynote, "We have to use this power for good and not evil." True that. But even stipulating that Microsoft will be universally good, will everyone to whom it gives the toolset be good? I have a hard time imagining that. As I wrote in my January column (msdn.com/magazine/mt793276), "… men turned their thinking over to machines in the hope that this would set them free. But that only permitted other men with machines to enslave them." Skynet, anyone?

Our new AI will keep getting better and better, without any additional coding, as it gains experience. That means that we've crossed a fundamental watershed. I'm not sure we recognize this, as an industry or as a society. Where is this going to take us? I can only defer to Arthur C. Clarke, who wrote in his introduction to "2001: A Space Odyssey": "It is important to remember that this is a work of fiction. The truth, as always, will be far stranger." ■

DAVID S. PLATT *teaches programming .NET at Harvard University Extension School and at companies all over the world. He's the author of 11 programming books, including "Why Software Sucks" (Addison-Wesley Professional, 2006) and "Introducing Microsoft .NET" (Microsoft Press, 2002). Microsoft named him a Software Legend in 2002. He wonders whether he should tape down two of his daughter's fingers so she learns how to count in octal. You can contact him at rollthunder.com.*