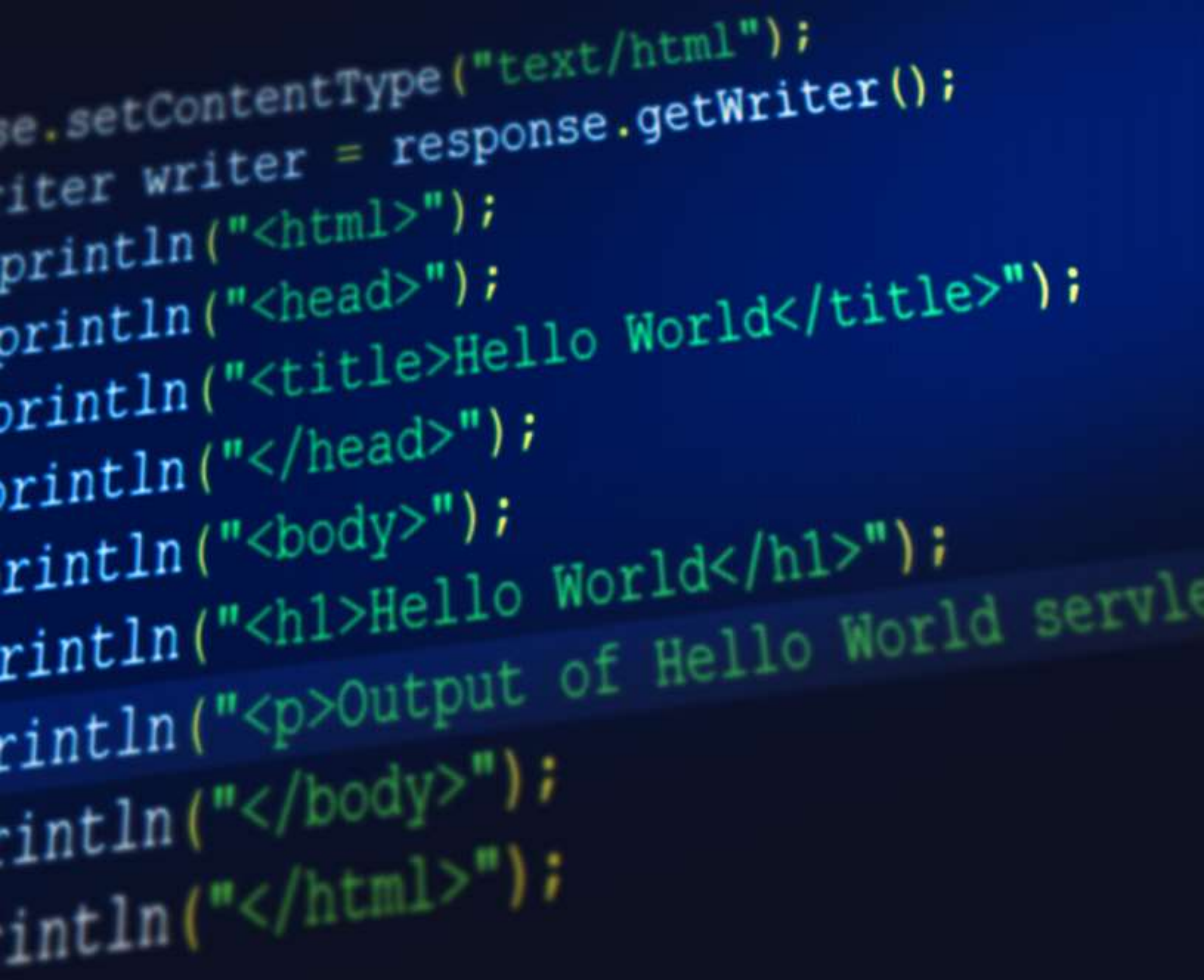


```
se.setContentType("text/html");
iter writer = response.getWriter();
println("<html>");
println("<head>");
println("<title>Hello World</title>");
rintln("</head>");
rintln("<body>");
rintln("<h1>Hello World</h1>");
rintln("<p>Output of Hello World servle
intln("</body>");
intln("</html>");
```

Cutting Right to the Code:

A web developer's guide to help eliminate non-coding tasks and get code done faster

Microsoft Azure

Microsoft

Abstract

This eBook helps web developers cut right to the code and get code done quickly by providing an overview of key services in the cloud, what services to use based on your needs, step by step guidance, sample code, sample applications, and a free account to get started.

Developers get out of bed wanting to write code. Unfortunately, many non-coding tasks need to be wrangled before coding can begin. We don't mean coffee; we mean things like waiting for servers to be purchased, installing your database engine, configuring your firewall ports, reinstalling the right version of the database engine, figuring out your deployment toolchain, creating a reasonable authentication approach, and getting the testing frameworks lined up. Those are just the things in the way, not the things that aren't in the way but can't be forgotten, such as backups, scale planning, and logging. These tasks cost you precious time that could be better spent coding, brainstorming, standing around the whiteboard, playing foosball, etc. And once you jump through the necessary non-coding hoops and finally get to coding, the question quickly becomes, "How do you get code done faster?"

As developers, our job is to cut through any barrier that stops us from being creative and get our code done faster. Taking advantage of the right services running in the cloud helps you cut right to the code by eliminating a number of non-coding tasks. Examples include environment management, spinning up a SQL Database in Azure to skip version problems, automating backups, and enabling advanced encryption at rest features, using repeatable scripting of environments to make all the firewall ports open automatically, adopting the CI toolset of Azure Websites to publish straight from GitHub (oh, hello again, three otherwise lost weeks), and provisioning Azure AD for authentication and authorization.

Then there's "getting code done fast." As one developer put it, there are two types of code:

1. The fun stuff: Code that makes your app unique and valuable, and helps drive the business.

2. The not fun stuff: Glue code which doesn't add a lot of unique value but makes stuff work. An example would be building framework code to compensate for how different browsers or devices work.

Powerful finished services in the cloud can help web developers get from 0-60 in their app building process. How about easily handling logins from your company's directory and popular social media logins? How about getting global scale with a world-wide CDN? Other services like Push notifications, analytics, and caching are ready for you. All of it can be accessed with a few lines of code.

The time is now. We invite you to cut right to the code.

PUBLISHED BY
Microsoft Corporation
One Microsoft Way
Redmond, Washington 98052-6399

Table of Contents

Overview

Introduction

With over sixty services, Microsoft Azure provides a modern platform for any app, written in any language. The lifeblood of the private and public cloud is web applications (web apps) that expose rich, interactive sites and services to everyone on the planet. Azure App Service helps you build modern web apps that scale. It is a cloud platform to build powerful web and mobile apps, for any platform and any device, that connect to data anywhere, in the cloud or on-premises. Built for developers, App Service is a fully-managed platform with powerful capabilities that make it easy to stage and then deploy to production with support for automatic patching. You can code in your language of choice, use your favorite development tools, and easily scale up and out to meet the demands of your business and customers.

Azure App Service provides an abundance of services and features in four categories:

- Web Apps
- Mobile Apps
- API Apps
- Logic Apps

This guide will cover how you can build a new web site or move an existing web site to Azure App Service and then take advantage of the platform services to build a modern web app that scales when you need it. Other guides will cover other parts of Azure App Service.

Pro Tip

One thing to watch for in this guide is the *Additional resources* sections. Each one will provide additional links to more detailed documentation about the subject you've just read about. In addition, many will have a link to the hands-on labs, sample code, and more. These labs contain tons of sample code, step-by-step walkthroughs, and ARM templates to help you explore and get to know Azure App Service and web apps.

Microsoft Azure App Service is a platform-as-a-service (PaaS) offering. You use it to create web and mobile apps for any platform or device. You can use it to integrate your apps with SaaS solutions, connect with on-premises applications, and automate your business processes. Azure runs your apps on fully-managed virtual machines (VMs), with your choice of shared VM resources or dedicated VMs.

App Service combines the web and mobile capabilities that were previously delivered separately as Azure Websites and Azure Mobile Services. In addition, there are new capabilities for automating business processes and hosting cloud APIs. As a single integrated service, App Service lets you compose various components into a single solution.

Overview

Why App Service for your Web Apps

Azure App Service provides a number of great benefits to you and your application. When looking at App Service over other options, a key foundational tenet is that App Service manages the underlying infrastructure. As a developer, you're not responsible for patching and maintenance. This is a major non-coding task that is eliminated. This means you can focus on coding tasks instead of software updates and maintenance. Speaking of code, App Service provides choice with first-class support for ASP.NET, Node.js, Java, PHP, and Python.

App Service was built with DevOps in mind. You can set up continuous integration and deployment with Visual Studio Team Services, GitHub, or BitBucket. You can shepherd updates through test and staging environments as well as perform A/B testing. Because App Service has a rich API surface, you can manage your apps using Azure PowerShell or the cross-platform command-line interface (CLI) tooling.

Once you've got your web app running in Azure, you're supported by the global scale and reach of twenty-four world-wide regions so you can host your app where you need it most. Additionally, you'll have the peace of mind that comes with App Service's high availability SLA.

With all of these benefits, you'll want to get started quickly. In the rest of this guide, you'll see how to use dedicated tools in Visual Studio to streamline the work of creating, deploying, and debugging your web app.

Getting Started with Microsoft Azure

In order to build and deploy your web app to Microsoft Azure, you need a subscription. You can get a free Azure account or it's possible you already have access and might not even realize it. You could have access via:

- Your organization's subscription
- Your own subscription provided via your MSDN Subscription
- Your own subscription provided via Visual Studio Dev Essentials

Regardless of how you have access, you'll need enough permissions to create and manage new objects in the subscription.

Common Web App Scenarios

You're ready to move your web app to the cloud. You want the benefits, but are unsure where to start. The following scenarios are designed to give you a jumping-off point covering common tasks that many customers have.

When to Use It

App Service is the preferred option for building web sites because it provides the productivity, scale, performance, and deployment options for most requirements. With or without a database, you use App Service when you want to focus on building unique features for your app and need the infrastructure to just work. You can use App Service in conjunction with the many other Azure services like Azure AD, Redis Cache, Azure CDN, and Application Insights to quickly build and deploy powerful apps.

As with most technologies, Azure continues to grow and evolve at a rapid pace. Steps to complete certain tasks get optimized out, tools get updated, and generally things just get better. This document reflects the state of the art as of June 2016, using versions with the latest updates and patches. As with many things, there is more than one way to create something.

The tools that enable you to get the most done in as few steps as possible include:

- Visual Studio 2015
- Azure SDK and Tools for Visual Studio 2015

Moving an Existing ASP.NET Website to App Service

Overview

Moving an existing ASP.NET website backed by a SQL Server database to Azure App Service only requires a few steps once you have the right tools installed. Most of the steps can be completed from within Visual Studio with a few items requiring the use of the web based Azure Portal.

The problem

When you're new to a platform, it can be hard to get started. How should you move things? What's the right order of operations? How do I get my app to talk to its data sources? If you've never done it, it's not always obvious where to begin.

The solution

For this type of application, you'll move your database catalog and sample data to the cloud. You'll then migrate your website and your data. Once you've done that, you can tweak additional settings like custom domains and implement continuous delivery via tools like Visual Studio Team Services.

Services used

- Web Apps
- SQL Database

Publishing the Database

Azure's SQL Database (SQL Database) is a highly compatible version of SQL Server optimized for the Azure cloud. As a platform-as-a-service (PaaS) offering (as opposed to just running SQL Server in virtual machines), there are some differences and incompatibilities between the two versions. You need to validate your database for compatibility. If your database uses partially or completely unsupported features, you need to re-engineer it to remove these incompatibilities before migrating. You should read Azure SQL Database Transact-SQL differences for the small set of features not yet supported by SQL Database. Once you've addressed any issues, you often migrate your local SQL Server database up into SQL Database twice. First, you move just the database catalog (and possibly some sample or test data), and then you do it again if you already have the database in production to move your real data.

There is more than one way to accomplish the validation and deployment of your database. If it has been in use for a while, you should review the Azure SQL Database General Limitations and Guidelines. As Microsoft enhances SQL Database, new features are added. Microsoft has made it a key goal for SQL Database V12 to improve the compatibility with Microsoft SQL Server 2014, and to maintain the compatibility as they release new versions of SQL Server.

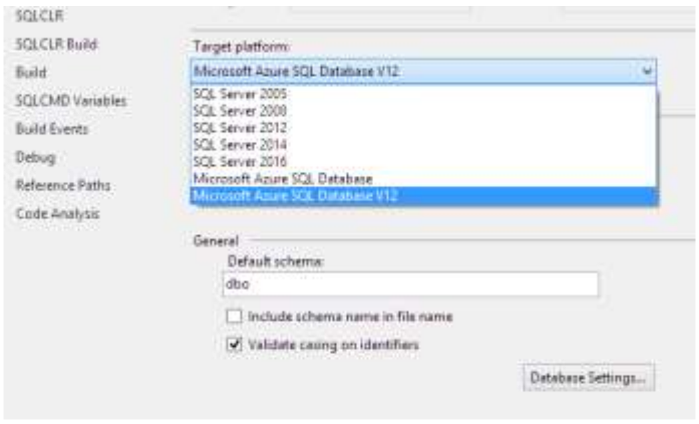
Migrate and Test SQL Server database compatibility

There are several different tools you can use to test for compatibility. For developers using Visual Studio, the SQL Server Developer Tools (SSDT) provide a familiar experience to work with the database catalog, check for compatibility, and ultimately migrate to Azure. SSDT uses the most recent set of compatibility rules and, as of June 2016, supports SQL Server versions 2005 to 2016.

You can get the SSDT via the Download SQL Server Data Tools (SSDT) page.

With SSDT, you can import your database catalog three different ways. You can live connect to a running SQL Server instance. You can import the catalog from a Data-tier Application (dacpac) file. Finally, you can import a SQL Script file that describes your database's objects. Once you've completed the import process, you'll have an offline version of your database catalog that you can then test for Azure compatibility. The MSDN docs cover Project-Oriented Offline Database Development in more detail.

When you have your database catalog imported (and hopefully after you've put it under version control), you open the **Project Settings** page. You change the **Target platform** from the imported setting and change it to **Microsoft Azure SQL Database V12**.



Target platform: Microsoft Azure SQL Database V12

Default schema: dbo

☐ Include schema name in file name

☑ Validate casing on identifiers

Once you've made the change and saved it, you build the project. SSDT will then evaluate your database catalog against the supported features of Azure. If there any compatibility issues, SSDT will put them in the Visual Studio **Error List** window. You can then double-click on the error to open the offending object's script. Use the error code to get additional help online.

Once you've fixed all the errors, you can use the **Publish** command in SSDT to upload your new compatible database up to Microsoft Azure. However, this will only migrate the catalog. This is fine if you don't need to move any data. If you have data to move, you'll want to synchronize your project with SQL Server and fix up any conflicts caused by your adjustments. You can use the Data Comparison tools in SSDT to compare your catalog.

Once you've got your database catalog in a database with your data, you can use SQL Server Management Studio to export your database and its data using the **Export Data-tier Application** command. This creates a file that you can then import into your SQL Database in Azure. In order to do this, you'll need to have a few things ready.

First, you'll need an appropriate Azure SQL Database logical server defined with an empty, temporary database in the SQL Database instance. Azure will not let you create a new server unless it has a database in it, so you will need to create a server with an empty database. After you've imported your real database, you can delete the temporary one. When you create your SQL instance, you'll want to consider if you want the logical server to be shared by multiple apps or be dedicated to your app. To make it easier to manage resources in Azure, you put them into logical groups known as Resource Groups. You use the Azure Resource Manager to deploy, manage, and monitor the resources. You can read more about ARM and Resource Groups in the Azure Resource Manager overview. You'll need to put your logical server in a Resource Group at creation time. One other important point is where you want your database server located. Azure has over twenty regions around the world. To avoid unnecessary charges and for better performance, you could put your database and the web app in the same region. Naturally, before you do any deployments containing data, you should verify which region meets your company's data sovereignty requirements, etc. You'll find the SQL Database tutorial useful, as well as the Azure Developer Immersion hands-on labs.

Second, once you've got the logical server instance created, if you're working from a machine outside of Azure, you'll need to open up the Azure firewall in order for the SQL Server tools to reach the server and your database. You'll find the access link in the **Essentials** section of the SQL Server blade in the Azure portal.

Finally, once you've done this, you can connect to your instance in Azure. From the connection, you use the **Import Data-tier Application** option to load both the database catalog and the point-in-time copy of your data to your new server in Azure. At this point, you can remove the temporary database you defined when you created your SQL Database instance.

You can find more information starting at SQL Server database migration to SQL Database in the cloud. You will find additional links to the aforementioned tools and techniques there also.

Additional resources

Hands-on labs, sample code, and more ➲

What's new in SQL Database V12 ➲

Azure Resource Manager ➲

Update connection information

Once you've got the database up in the server, you can update your application's web.config to use the new database. This can be useful to test your app locally from Visual Studio before you migrate the web app to Azure. You can find connection strings in various formats on the

Database connection strings blade of your SQL database in the Azure portal. You'll want to use the ADO.NET format. You will need to replace the {your_username} and {your_password} tokens with the correct values. Once you're done testing, you should remove this information, as it's not something you'd want to commit to your source code repository. Azure web apps support application settings where you can securely store your database connection string and make it available to your application. Changes to these settings override values stored in your web.config.

Publish the web site

With your database in Azure, you're now ready to move the web application. Visual Studio in conjunction with the latest Azure SDK and Tools provides everything you need to perform the migration. If you have additional post-deployment work, you can consider combining version control and some form of automated deployment.

With your solution open in Visual Studio, you use the **Publish** command off of your web project's node in the **Solution Explorer** window. The **Publish Web** wizard will walk you through the various steps required to get your local ASP.NET app running in Azure. When you first use the wizard, you'll need to log in using your Azure account credentials. This provides the wizard with a list of your subscriptions. Once you pick it, you'll be shown any Resource Groups you have access to in the subscription with web apps deployed. The assumption is that you're creating a new one and the wizard provides extra support for new deployments.

When creating a new deployment, the **Create App Service** dialog requires you to provide a number details including name, subscription, Resource Group, and an App Service Plan. The name forms the host name of the DNS entry for your app and must be globally unique. Azure will validate your choice before letting your continue. Thus if you type **myapp**, the fully qualified domain name will be myapp.azurewebsites.net. The subscription controls how much Azure resources you can use. In addition, this will scope which existing Resource Groups and App Service Plans you can access. It also affects the names you choose as both types of items need unique names within a subscription. It's at this point you can choose the same Resource Group you defined when you created your logical SQL Database server.

While it's possible you'll have an existing App Service Plan, you'll typically want one specific to your web app. When you create a new App Service Plan, you give it a name. You then set the location. This should be the same location you used when defining your logical SQL Database server. Finally, you'll need to specify a size. Starting with the free option is great for demos and initial configuration. However, you'll note that you don't get access to a number of important features including deployment slots, custom domain names and custom SSL certificates. The App Service plans page provides detailed information about the different plan categories. The various App Service Pricing pages provide specifics on pricing and the levels within each plan category.

Once you've created your App Service Plan, you can publish your app to Azure. Once the publication is done, you can access the Application Settings blade if necessary to update the SQL Server connection string information.

Additional steps

After you've got your web app loaded in Azure, there are additional steps you might want to perform.

Custom Domain

By default, when you publish your web site to Azure, you'll be able to access it via myapp.azurewebsites.net. In particular, this is the only option when using a free App

Service Plan. If you upgrade to at least the Basic plan, you can configure your web app with a customer domain name. If you don't have a custom domain already, you can buy and register one from the Azure portal.

SSL Certificate

One great thing about the free Azure App Service plan is that you get SSL included. However, once you add a custom domain, you'll need to provide your own SSL certificate. To do this you'll need to acquire a SSL certificate from a recognized certificate authority. You'll then need to ensure you're using the correct pricing tier. You then configure your web app to use the cert. Finally, you might need to force your app to support only SSL.

Continuous Delivery

While using the Visual Studio tools for initial deployment is great, long term use, especially when working on a team is not ideal. You'll really want to consider using one of the great tools out there, such as Visual Studio Team Services (VSTS) or GitHub. Both services provide mechanisms to update your web app after changes are pushed to a centralized source repository. VSTS in particular supports a full Release Management tool that supports release pipelines with deployment to staging slots and automated regression testing before going live. The Team Services docs provide additional details, pointers, and walkthroughs.

Additional resources

Configure a custom domain name in App Service ➲

Enable HTTPS for an app in Azure App Service ➲

Visual Studio Team Services ➲

Hands-on labs, sample code, and more ➲

Identity Management

Overview

Modern web apps often need to restrict what data and features are made available to individuals and/or groups of users. Azure App Service supports single sign-on (SSO) for your apps regardless of where users are logging in from, whether it be the public internet or your own internal network. In addition, you can integrate social logins—Facebook, Twitter, etc.—for customers and non-corporate members through integration with OAuth 2.0, OpenID Connect, and SAML 2.0.

The problem

How do you get an app that's run on premises to be cloud aware and support logins outside of your corporate network? What about customers and business partners who need access to your app but aren't employees?

The solution

Azure Active Directory (AD) provides organizations with enterprise-grade identity management for cloud applications. Azure AD integration gives your users a streamlined sign-in experience, and helps your application conform to IT policy. By enabling Azure AD support in your application, your users won't need to remember additional set of credentials. Instead, they'll use the same information already in use to access your organization's resources. In addition, using Azure AD, you gain access to a very large social network in that you can allow users with Microsoft accounts. Finally, Azure App Service enables integration with popular social media providers.

Common Web App Scenarios

Services used

- Azure Active Directory
- Azure App Service

Azure Active Directory

Adding Azure AD support to your application means that management is done with familiar concepts for managing user provisioning and access control. If you have an on-premises directory, you can sync with Azure AD. You can re-use existing Azure AD groups and distribution lists, and more. Administrators can assign access to apps, whether it be for specific users or entire groups.

In addition, your application has access to this information using the Azure AD Graph API. This API allows your app a number of identity and access control operations including:

- Create a new user in a directory
- Get a user's detailed properties, such as their groups
- Update a user's properties, such as their location and phone number, or change their password
- Check a user's group membership for role-based access
- Disable a user's account or delete it entirely

To get started, access the classic Azure portal at https://manage.windowsazure.com/ and log in (Microsoft's migration from the classic portal to the new portal is nearly complete, so only use the classic portal when necessary). The portal contains a section labelled **Active Directory** that provides the controls to define users, groups, and applications. To be able to control who can use your application, you can create your own Azure AD directory or use an existing one—it depends upon how you want to segment access to your application. Within your AD you will need to create an *application*, which is a set of configuration settings that tell AAD about your web app.

If you've created a new directory, you will need to add users. One of the great things about Azure AD is that it provides support for adding new and existing users from other sources. Using the Azure Management portal, you can add:

- A new "native" user to the directory
- An existing user in another Azure AD directory
- An existing user with a Microsoft account
- Users in partner companies

Assuming you have users in place, you need to define an Application object in the directory that you'll map to your web app. When you add your application you provide a name and specify the type of application. Currently you have two choices:

- Web Application and/or Web API
- Native Client Application

The choice in this case is obvious. Once you pick the web app related option, you need to provide two items. The first is the *Sign-on URL* which is the URL a user will use to login to your app. The second item is the *App ID URI*. This just needs to be a unique URI (often provided in the form of a URL) the Azure AD can use to identify your app. Once you've entered these basic items, you can then use the **Configure Azure AD Authentication** wizard in Visual Studio to add SSO support via Azure AD to your web app.

Social Sign In

When using Azure AD, you gain access to a very large social network in that you can allow users with Microsoft accounts—the same ID shared by Xbox and Outlook.com for example—to use your application. Adding social sign-in via other providers to your web app requires three steps. First, you need to gain developer access to your social networks of choice, register your application, and get items like access tokens, etc. Second, once you have this information, you use the Authentication / Authorization blade in the Azure Portal for your web app. Once you turn this feature on, you have access to a number of social networks like Facebook, Google, and Twitter.

Third, you will need to tweak your application so that users can choose the supported login types that you've enabled, both for successful and failed logins, and redirect the user to the right place in your web app once it has completed the authentication and authorization dance.

Additional resources

[Azure App Service Security](#) ➲

[Azure Active Directory Management Portal](#) ➲

[Hands-on labs, sample code, and more](#) ➲

Scaling your Web App

Overview

Scale up or scale out—Azure App Service has you covered with easy-to-configure tools right within the Azure Portal. You control how your web app scales by adjusting settings of your App Service Plan.

The problem

You want your users to have fast access to your web app. You need to be able to handle temporal pressures caused by end of month and seasonal demand. The bottom line is your web app needs to be up and ready when your users need it.

The solution

Azure App Service provides what you need, when you need it with easy-to-set scale up and scale out options that can provide your app with elastic scale with a few mouse clicks and a slider or two.

Services used

- Azure App Service

The three Basic options give you one or more dedicated CPU cores and dedicated memory, along with the ability to specify custom domains. With the Free plan, you have to use the azurewebsites.net domain. In addition to allocating up to four cores, you can also scale out, with up to three machines with four cores each. With the Standard plans, you can scale out to up to ten machines in a web farm, along with support for using SSL with your custom domains. As mentioned earlier, even free web sites support SSL automatically, but you have to use the azurewebsites.net domain if you want SSL on the Basic, Shared, or Free plans.

The Standard plan offers automated daily backup of your site, along with slots to support staged deployments with rollback. The Standard plan also offers support for Azure's Traffic Manager, enabling you to deploy to multiple sites across the world and direct incoming requests to whichever is nearest to the user. Finally, the Premium plans allow you to scale out to up to twenty instances, with multiple automated backups every day (up to fifty), as well as offering up to 250GB of filesystem storage and support for BizTalk Services.

As you dig in and move up out of the free plan, you can choose different "scale by" formats: CPU Percentage, manual instance count, and finally schedule and performance rules. The high-end options will vary based on the plan level you select. The first two options provide "large" levers to adjust the app service scaling properties by either CPU percentage load—add more power when the CPUs are taxed—or total instances which simply sets a hard-coded upper bound based on count.

You'll find the schedule and performance rules provide more granular control. You can create your own rules and create a schedule that adjusts your instance counts based on time and performance metrics. You can configure auto-scaling rules for different performance metrics, including CPU, memory, disk queue, HTTP queue, and data flow. One other feature that comes in handy is that you can send e-mails to administrators when scale actions kick in.

Also, don't forget: if your web app uses a SQL Database, the sizing choice you made when you configured it can affect your application's overall performance, too. The next section covers how you can mitigate that.

Additional resources

Scale a web app in Azure App Service ➲

Hands-on labs, sample code, and more ➲

Caching for Performance

Overview

Data persistence takes many forms. An early assumption in this document was that your web app uses SQL Server for the application's main persistence. However, modern web apps can use any number of additional storage facilities for supporting data and content. Traditional file repositories, NoSQL databases, and in-memory caches are all useful tools to have as your application's needs change. Adding caching support is one of the easier ways to enhance your application's performance.

The problem

Hitting the database every time information is needed by a web app can be expensive. It slows down the app and can cause your application run time costs to go up needlessly.

The solution

Azure Redis Cache provides high throughput, consistent low-latency data access to power your fast, scalable Azure hosted web app.

Services used

- Azure Redis Cache

Azure Redis Cache is a high-performance, memory-based cache which can significantly improve performance and scalability when you have data that needs to be accessed very frequently. Azure Redis Cache is based on the popular open-source Redis cache. It is accessed over the network, so it can be shared by multiple machines in a web farm if you choose to scale out your service. This makes it more flexible than simply caching data in a web server's memory; if you update a Redis cache entry, everything will have access to the new value. By using a Redis cache, you can take significant load off persistent storage systems such as SQL Database.

To use Azure Redis Cache, you need to first add it to your web app via the Azure Portal. You need to provide a DNS name that is globally unique. As with other Azure assets, you'll want to add it to the correct Resource Group and location used by your web app. Finally you'll need to determine your pricing tier. Once you've done that you can turn to your web app.

As with many other technologies, you add support to access Redis Cache to your application by importing the relevant NuGet package. The most common one used is the Stack Exchange Redis package. This package provides your application with access to the Redis Cache Connection Multiplexor and other objects necessary to cache your application's data. Once you've worked out the areas that need caching, you need to add the Redis Cache connection information to your app so it has access to the cache when you deploy your updated version to Azure.

Additional resources

How to Use Azure Redis Cache ➲

Hands-on labs, sample code, and more ➲

Better Customer Experience with a CDN

Overview

Rich web applications provide a dynamic experience tailored for each user. However, often, there is shared content that enhances the experience.

The problem

Your web app is used by people all over the globe. The reality is, even with widely deployed broadband, we're still limited by the speed of light. You want to get static content closer to your users.

The solution

Azure Content Delivery Network is a powerful multiple-provider solution that uses global points of presence to accelerate getting any content to any device that needs it.

Services used

- Azure Content Delivery Network
- Azure App Service

Microsoft Azure provides your web app access to a global audience. Scaling out with a Content Delivery Network (CDN) reduces latency between your users and the data needed via global points of presence to provide a rich and compelling experience. The Azure CDN caches your static web content at strategically placed locations to provide maximum throughput for delivering your content to your users.

You use a CDN in order to:

- Provide a better experience for end users—speed up the round trips required to load content
- Help your web app scale to handle on-demand load
- Distribute requests to edge servers thus sending traffic to your root servers

The Azure CDN is broken down into three tiers that offer many common services and a few unique features like real-time stats. There are two standard offerings and a premium offering. You can get more details at the Overview of the Azure Content Delivery Network (CDN) page.

CDNs are most effective for static content—files that do not change frequently. The usual best candidates are image files (e.g., JPG, PNG, or SVG), CSS, and JavaScript. HTML does not always benefit—it depends on the application. To get started with Azure CDN you need to create a CDN profile and then define what content gets delivered.

You create your CDN profile via the Azure portal. In doing so, you need to provide a name, assign it to a Resource Group, and pick your Pricing Tier. Once Azure has finished creating the CDN Profile, you create an endpoint with a unique name. Azure will validate your name. You then set the Origin type to Web App and pick your web app. Once you do this, you can move to your application to make adjustments to take advantage of the CDN.

Normally, your application will use relative URLs to reference content like images, CSS, and JavaScript files. However, in order for your web app to take advantage of the CDN, it needs to pull the content from the CDN edge servers. If you want to test the CDN, you would test in logical groups: CSS first, JavaScript files, and then images. Depending upon your content, you would possibly test other files like movie files and HTML files. The final task is to configure your ASP.NET app to define how long content should be cached at the edge servers before a refresh is served. You might want to select an interval like five to ten days, depending upon how often you update your static content. Be aware that a single CDN *endpoint* must be configured to use exactly one server as its *origin*.

If you have an endpoint called *myendpoint*, then:

http://myendpoint.azureedge.net/content/logo.jpg

… is going to fetch the underlying resource from …

http://mywebsite.azurewebsites.net/content/logo.jpg.

You can't configure a CDN endpoint so that assets under */content* caches content from web app A, while stuff under */othercontent* caches content from web app B. One particular endpoint basically gives you a cache for one particular web app.

Azure CDN works with more than just web apps. You can point a CDN endpoint at any valid endpoint like storage, a cloud service, or a plain old IaaS web server. The point is that this one CDN endpoint will always get its underlying content from one source location.

Additional resources

Using Azure CDN ➲

Hands-on labs, sample code, and more ➲

Detect Failures Faster

Overview

Application Insights helps you understand how your application is performing by tracking exceptions and gathering performance diagnostics. It allows you to use real-time metrics to analyze request load, server performance counters, and response times across dependencies. In addition you can diagnose exceptions and failed requests and correlate them with events and traces. Designed to help you discover the root cause of abnormal app performance behavior, Application Insights provides interactive data analytics so you can perform ad-hoc queries and use full-text search to find the information you need. In addition, Application Insights uses machine learning capabilities to continually analyze your application. This allows it to learn your app's normal behavior so service degradations or disruptions are automatically detected and reported—helping you respond to issues at the speed your customers demand.

The problem

Web pages load slowly. Users are seeing errors. Satisfaction is low and your phone and email are about to explode.

The solution

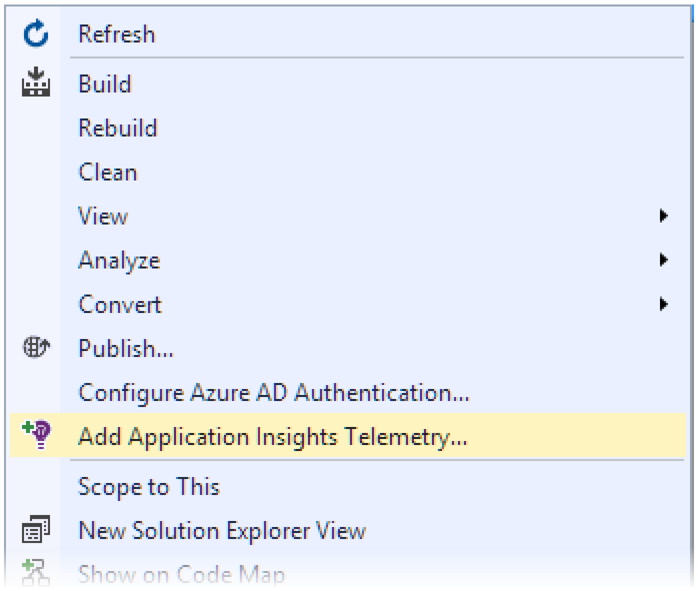
Application Insights in Azure is a service that helps you detect failures in you web app and roll out fixes before your customers are aware of these failures.

Services used

- Application Insights

As with other features covered so far, Application Insights has a management function in the Azure Portal and SDK entry points in your application. Application Insights is something you can add to your application at the very beginning, or you can add it after the fact and integrate it as deep as you need including defining customer events.

Application Insights comes with Visual Studio. This makes it easy to enable Application Insights during app creation or in an existing application. With your solution open in Visual Studio, you can right-click on your web app's project node and choose the Add Application Insights Telemetry command.


Refresh
Build
Rebuild
Clean
View
Analyze
Convert
Publish...
Configure Azure AD Authentication...
Add Application Insights Telemetry...
Scope to This
New Solution Explorer View
Show on Code Map

When you do this, Visual Studio downloads the necessary supporting Application Insights NuGet packages and adds them as project dependencies. Up in Azure, it creates an Application Insights resource that you can then access for reporting. It generates a unique instrumentation key for your application (a GUID), that is added to an *ApplicationInsights.config* file so that the SDK can send telemetry to the portal.

Once you've done the basic configuration, you can build and run your app locally. Your app will immediately start sending telemetry data to Azure. You can view the data right from inside Visual Studio by using the Search Debug Session Telemetry command. This command is a sub-command on your web project's Application Insights menu which appears via a right-click on the project node in Solution Explorer. On the other hand, if you choose the Open Application Insights command, you'll be sent to the Azure portal where you'll have access to dashboards, more charts, and analytic tools. In the Azure portal, you can search for events and create charts over aggregated metrics.

You can also use the Analytics feature of Application Insights. Application Insights Analytics has a browser-based UI like the Azure Portal. The UI is easy to use but the real power is in the data and query engines. It's designed to make it easy to query over petabytes of data. Microsoft currently uses it to ingest over 1 trillion events and 600TB of data a day. The language has many attractive features allowing you to filter your raw app telemetry by any fields, including your custom properties and metrics. You can join multiple tables as well as correlate requests with page views, dependency calls, exceptions and log traces. It supports powerful statistical aggregations, as well as immediate and rich visualizations. You can use this feature to ingest your app data and run queries. As of June 2016, this feature is still in preview and query results are limited to just over a week of past data.

Application Insights and the standard telemetry provide a lot of data. However, often you need to get application-specific data also so you can correlate application behavior with system response times and exceptions. With a few lines of code, you can send telemetry to the service. There's an entire set of "track" methods your code can call which will then push data to Azure for later analysis.

Additional resources

What is Application Insights? ➲

Hands-on labs, sample code, and more ➲

Building a New Website on Azure App Service

Overview

Someone has an idea for "the next big thing" and they want it built yesterday. Microsoft Azure and Visual Studio provide everything you need to make dreams come true.

The problem

Ideas are great, but until you can see them and use them, it's hard to tell if what is imagined can really work.

The solution

Visual Studio 2015's File | New dialog provides is the gateway to easily building a rich, modern web app where you focus on the code and let Azure provide all the services to make the "next big thing" a reality.

Services used

- Azure Active Directory
- Azure App Service

Starting from "File | New" is a wonderful place to be when building any software solution. Building a new web app that runs from the beginning on Azure means you have an amazing array of choices in how you build out your solution.

In order to build a new ASP.NET solution, you'll want the same core tools as you would when migrating:

- Visual Studio 2015
- Azure SDK and Tools for Visual Studio 2015
- SQL Server Management Studio 2016

With these tools in place, you'll find many features are options to embrace during the initial project creation process.

Building a new ASP.NET web app for Azure starts with the File | New dialog box in Visual Studio 2015. As you choose an ASP.NET template type, you have the option of configuring the project for Azure as you carry out the project creation.

In fact, the list of items you can configure at project creation includes:

- Adding Azure Active Directory support
- Application Insights
- Resource Group creation
- App Service creation

The Visual Studio 2015 tools and templates ensure you can get straight to the code yet still build a great application that scales on Azure.

Additional resources

Create an ASP.NET MVC app with auth and SQL DB and deploy to Azure App Service ➲

Azure Tools for Visual Studio and Azure SDK ➲

SQL Server 2016 ➲

Conclusion

Web apps once thought impossible due to scale, complexity, or because they simply couldn't be imagined, are now a reality with the cloud. In this guide, we've explored the Azure App Service and highlighted Azure's support for platform as a service (PaaS).

We've shown you how you can take an existing website backed by a SQL Server database and move it to the cloud. You've seen how you can easily add features such as identity management and caching to your web app. You've also learned that you can enable rich application monitoring with a few clicks.

Cut right to the code today. Welcome to Azure.

Recommended Next Steps

- Download the hands-on labs, sample code, and ARM templates to cut right to the code.
- Create an Azure account and get started for free with $200 in Azure credit.
- Explore the range of free options available to get you started, like hosting up to ten free web and mobile apps on Azure App Service, and use Visual Studio Application Insights to monitor live apps.
- Be our guest for up to an hour of Azure App Service experience with no subscription, free of charge and commitment.