Agenda: Azure App Services - Web Apps

- Introduction
- App Service - Application Types
- Deploy Web Apps
    - Deploying Web App directly from Visual Studio
    - Automate deployment from Dropbox and One Drive
    - Create, configure and deploy packages
    - Implement pre- and post-deployment actions
- App Service plans
    - Create App Service Plan
    - Migrate Web Apps between App Service plans;
    - Create a Web App within an App Service plan
- Configuring Web Apps
    - Application Settings Configuration,
    - Database Connection Strings,
    - Configuring Handlers and Virtual Directories,
- Manage App Services
    - Configure a custom domain name
    - Enable SSL for your custom domain
    - Understanding Deployment Slots and Roll back deployments;
    - App Service Protection
    - Manage Roles for an App service
- Configure Web Apps for scale and resilience
    - Horizontal and Vertical Scaling of a Web App
    - Configure auto-scale
    - Change the size of an instance
- Monitoring, Debugging and Diagnosis
    - Retrieve Diagnostics data
    - View Streaming Logs
    - Configure endpoint monitoring
    - Configuring Alerts
    - Configure diagnostics
    - Use remote debugging
    - Monitor Web App resources

Azure App Services Introduction

- Azure App Service is the only cloud service that integrates everything you need to **quickly and easily** build web and mobile apps for **any platform** and **any device**.
- Built for developers, App Service is a **fully managed platform** with powerful capabilities such as built-in DevOps, continuous integration with Visual Studio Team Services and GitHub, staging and production support, and automatic patching.

App Types

App Service allows you to create the following app types from a single development experience:

1. **Web Apps** - Quickly create and deploy mission critical Web apps that scale with your business.
2. **Mobile Apps** - Engage employees, partners and customers on any device at any time.
3. **API Apps** - Easily build and consume Cloud APIs.
4. **Logic Apps** - Automate the access and use of data across clouds without writing code.
5. **Serverless Functions –** Function written by developer and executed without any dedicated hardware.

As a single integrated service, App Service makes it easy to compose the above app types into a single solution, allowing you to easily build apps that target both **web and mobile clients** using the same back-end and integrate with on premise systems as well as popular services such as Office 365 and salesforce.com.

App Service Web Apps is a fully managed **compute platform** that is optimized for hosting websites and web applications. This platform-as-a-service (PaaS) offering of Microsoft Azure lets you focus on your business logic while Azure takes care of the infrastructure to run and scale your apps.
The compute resources may be on shared or dedicated virtual machines (VMs), depending on the pricing tier that you choose.
Your code can be in any language or framework that is supported by Azure App Service, such as ASP.NET, Node.js, Java, PHP, or Python.

Why Web Apps:

1. Multiple languages and frameworks
2. Application templates in Azure Marketplace
3. DevOps optimization. Continuous delivery and Continuous deployment.
4. Test in production using Deployment slots
5. Global scale with high availability
6. Connections to SaaS platforms and on-premises data.
7. Visual Studio Integration

Deploy Web Apps

Deploying your app to App Service is a matter of deploying your code, binaries, content files, and their respective directory structure, to the **/site/wwwroot** directory in Azure.

1. **Create a Web App in Azure Portal**
    a. Login to Azure Portal, https://portal.azure.com/
    b. Azure Portal → More Services → Web App → + Add
    c. Select Web Apps → Create
    d. Name = "DssDemoWebApp", Subscription = "Free Trail" Resource Group="DemoRG", App Service plan/Location=Create New Plan (Name=Standard_Plan, Location=Central US, Pricing tier=S1 Standard.
    e. Application Insights=Off
    f. Create
2. **Visual Studio → Create a new ASP.NET MVC Web Application**
    a. File → New → Project
    b. Visual C# → ASP.NET Web Application, Project Name="DemoWebApp" → OK
    c. Select Template = MVC, Change Authentication = No Authentication → OK

Note: In a few seconds, Visual Studio creates the web project in the folder that you specified

3. **Deploy / Publish the project from VS.NET.**
    a. In **Solution Explorer**, right-click the project, and choose **Publish**.
    The wizard opens to a *publish profile* that has settings for deploying the web project to the new web app. If you wanted to deploy to a different web app, you could click the **Profile** tab to create a different profile.
    b. Choose default options and finally click on Publish.
    The **Output** and **Azure App Service Activity** windows show what deployment actions were taken and report successful completion of the deployment.

4. **Getting Publish Profile from Azure Portal and publishing from Visual Studio**
    Ideally used when developer doesn't have direct access to Azure Subscription.
    a. Click **App Services**, and then click the name of your web app.
    b. In the tool bar click on Get Publish Profile
    c. Save the Profile locally on your disk.
    d. Go to VS.NET, Right Click on Project → Publish
    e. Select Profile Tab → Click Import → Provide the downloaded profile name → OK
    f. Click Publish.

Note: Profile will be saved for further use in <Project>/Properties/PublishProfiles/*.pubxml

To Gets an Azure Web App publishing profile using PowerShell:

```
Get-AzureRmWebAppPublishingProfile -ResourceGroupName "DemoRG" -Name "DemoWebApp" -Format "WebDeploy" -OutputFile "D:\outputfile.publishsettings"
```

5. Publishing using FTP tools like FileZilla

a. Azure Portal → Click **App Services**, and then click the name of your web app.
b. Go to Settings → select Deployment Credentials
c. Provide FTP/deployment user name and password
d. Save
e. Look at Essentials Section of Selected App Service and copy FTPS hostname and user name
f. Open **Windows Explore** and use the above hostname and credentials to connect and upload files.

Note: The User Credentials: FTP Username/Password is same for all applications in a given subscription.

Note: Although it's easy to copy your web app's files to Azure using FTP utilities, they don't automatically take care of or coordinate related deployment tasks such as deploying a database or changing connection strings. Also, many FTP tools don't compare source and destination files in order to skip copying files that haven't changed. For large Apps, always copying all files can result in long deployment times even for minor updates since all files are always copied.

Automate deployment from Dropbox and One Drive

Dropbox is not a source control system, but if you store your source code in Dropbox you can automate deployment from your Dropbox account.

1. Create a drop box account @ **http://www.dropbox.com**
2. Go to http://portal.azure.com/
3. Select the App Service → Settings → Deployment→ **Deployment options** → Configure required Settings → Dropbox
4. Authorize Azure to access your drop box
5. Go to https://www.dropbox.com/ and go to folder Apps → Azure → Create a folder by name: <WebApp Name>
6. Upload the files to the above folder
7. Go to Azure Portal
8. Select the App Service → Settings → Publishing → Deployment Source → Sync the App Service
9. View the page in browser.

The cons of syncing with a cloud folder are:

- No version control for rollback when failures occur.

- No automated deployment, manual sync is required.

Note: Similar steps are required even for One Drive.

App Service Plan

- App Service plans represent the collection of physical resources used to host your apps.
- App Service plans define:
  - Region (West US, East US, etc.)
  - Scale count (one, two, three instances, etc.)
  - Instance size (Small, Medium, Large)
  - SKU (Free, Shared, Basic, Standard, Premium,Isolated)
- Web Apps, Mobile Apps, API Apps, or Functions, in Azure App Service all run in an App Service plan. Apps in the same subscription, region, and resource group can share an App Service plan.
- All applications assigned to an **App Service plan** share the resources defined by it allowing you to save cost when hosting multiple apps in a single App Service plan.

App Service Plans:

SKU	Custom Domains	SSL	Storage in GB	Horizontal Scaling	Traffic Manager	Slots	Backup	Biztalk Services
Free	N	N	1	N	N	0	N	N
Shared	Y	N	1	N	N	0	N	N
Basic	Y	Y (SNI SSL)	10	Y (Manual) (Max 3 instance)	N	0	N	N
Standard	Y	Y (SNI & **IP SSL**)	50	Y (Manual and Auto) (Max 10 instance)	Y	5	10 times daily	N
Premium	Y	Y (SNI & IP SSL)	250	Y (Auto) (Max 20 instance)	Y	20	50 times daily (Snapshot)	Y

Isolated. This tier runs dedicated Azure VMs on dedicated Azure Virtual Networks, which provides network isolation on top of compute isolation to your apps. It provides the maximum scale-out capabilities.

Because a single resource group can have multiple App Service plans, you can allocate different apps to different physical resources that spans geographical regions. For example, a highly available app running in two regions includes at least two plans, one for each region, and one app associated with each plan. In such a situation, all the copies of the app are then contained in a single resource group. Having a resource group with multiple plans and multiple apps makes it easy to manage, control, and view the health of the application.

- It is recommended to isolate an app into a new App Service plan when:
  - App is resource-intensive.
  - App has different scaling factors from the other apps hosted in an existing plan.
  - App needs resource in a different geographical region.
- You can move an app to a different App Service plan in the Azure portal. **You can move apps between plans as long as the plans are in the same resource group and geographical region.**

AS1 - AP1 (R1) - East US
AS2 - AP2 (R1) - West US
AS3 - AP3 (R1) - East India
- AP4 (R1) - East US
- AP5 (R2) - East US

- You can create an **empty App Service plan** and then select the same while creating an App Service or you can create an App Service Plan while creating an App Service.
- If you want to move the app to a different region, one alternative is app cloning. Cloning makes a copy of your app in a new or existing App Service plan in any region. You can find **Clone App** in the **Development Tools** section of the menu. The web app must be running in the **Standard** mode in order for you to create a clone for the web app.

Application Settings Configuration

Azure Portal:

App Services → Select App Service → Settings → Application Settings

1. .NET Framework version = v4.6
2. Enable Debugging
3. Add Key – Value pair to the App Settings, Note that this overwrites the same key added to web.config in VS.NET project and published.
4. Add Connection String if required
5. Set Default page for the web site.
6. Http Handler mappings can be Set
7. Virtual applications and directories can be added

Web.Config

```
<configuration>
 <appSettings>
  <add key="Key1" value="Value1"/>
  <add key="Key2" value="Value2"/>
 </appSettings>
</configuration>
```

/Views/Home/Index.cshtml

```
<div>
  K1 = @System.Configuration.ConfigurationManager.AppSettings["Key1"]
  <br />
  K2 = @System.Configuration.ConfigurationManager.AppSettings["Key2"]
</div>
```

web.config = Least Precidence

web.release.config / web.debug.config

App Service -> Application Settings = Highest Precidence

Configure a Custom Domain Name in Azure App Service

Step 1: Reserve the domain name. There are many domain registrars to choose from eg: GoDaddy.com

Step 2: Create **DNS records** that map the domain to your Azure web app.

The Domain Name System (DNS) uses data records to map domain names into IP addresses. There are several types of DNS records. For web apps, you'll create either an ***A* record** or a ***CNAME* record**.

1. An A **(Address)** record maps a domain name to an IP address.
2. A **CNAME (Canonical Name)** record maps a domain name to another domain name. DNS uses the second name to look up the address. Users still see the first domain name in their browser. For example, you could map contoso.com to *<yourwebapp>*.azurewebsites.net.

Note: If the IP address changes, a CNAME entry is still valid, whereas an A record must be updated. However, some domain registrars do not allow CNAME records for the root domain or for wildcard domains. In that case, you must use an A record.

To get the IP Address:

1. App Services → Select App Service → Settings → **Custom domains** → Bring External Domains (blade)
2. The IP address is located at the bottom of this part.

Step 3: Add the domain name inside the Azure Portal.

1. App Services → Select App Service → Settings → Custom domains and SSL → Bring External Domains (blade)
2. Use the **DOMAIN NAMES** text boxes to enter the domain names to associate with this web app.

www.dssdemoapp.com

1. Buy that domain name from a registrar (Godaddy.com).
2. Find IP Address of you Host Machine (Website)
3. Name Server:
    a. Goto DNS Server Control panel provided by registrar
    b. Create a ARecord (domain name => IP)
    Or Create CName record (www.dssdemoapp.com => dssdemoapp.azurewebsites.net)
6. After around 24 hours, website can be visited using the domain name...
7. In Azure Web App → follow Step 3 above.

In Azure, DNS Zones / Name Server...

What happens when we browser http://www.dssdemoapp.com ->

1. Searcn www.dssdemoapp.com in DNS Server -> parent DNS Server --> ... --> return dssdemoapp.azurewebsites.net.

2. Searcn dssdemoapp.azurewebsites.net in DNS Server -> parent DNS Server --> ... --> return IP.

3. Browser creates a Socker (IP Address + port) and uses the same to send a request...

4. Web server is visited.

Enable SSL for your custom domain

Step 1: Get the Certificate for the custom domain from the Certificate Authority (Verisign)

OR

Generate a self-signed certificate using makecert.exe

1. **Search** makecert.exe in C:\Program Files (x86)\Microsoft SDKs\Windows\
2. Run command prompt as administrator
3. Go to folder of **makecert.exe** after knowing the same from step 1
4. Run the following command

    "C:\Program Files (x86)\Windows Kits\10\bin\10.0.15063.0\x64\makecert.exe" -r -pe -b 01/01/2015 -e 01/01/2019 -eku 1.3.6.1.5.5.7.3.1 -ss My -n CN=**<custom domain name>** -sky exchange -sp "Microsoft RSA SChannel Cryptographic Provider" -sy 12 -len 2048

5. From the **Start Menu** or **Start Screen**, search for **Windows PowerShell** and start this application.

    $mypwd = ConvertTo-SecureString -String "password" -Force -AsPlainText

get-childitem cert:\currentuser\my -dnsname **<custom domain name>** | export-pfxcertificate -filepath **dssdemoapp.pfx** -password $mypwd

Step 2: Configure Standard pricing tier:

Enabling HTTPS for a custom domain is only available for the **Standard** tier and above in Azure App Service. Use the following steps to switch your App Service plan to S2 **Standard** tier.

1. App Services → Select App Service → Settings → App Service Plan → Pricing Tier → S1 Standard

Step 3: Configure SSL in your App

3. App Services → Select App Service → Settings → SSL Settings
4. In the certificates section, click Upload
5. Upload **Error! Bookmark not defined.**password
6. In the SSL bindings section of the SSL Settings tab, use the dropdowns to select the domain name to secure with SSL, and the certificate to use. You may also select whether to use Server Name Indication (SNI) or IP based SSL.

How SSL Works:

https://www.deccansoft.com =>

1. Browser goes to server and finds who is the cetificate authority (CA).
2. It will go to CA store and get client certficate of that domain.
3. All data before sending to server is encrypted using the public key in the certificate.
4. Server receives the request and decrypt the data using the private key in the certificate.

Deployment Slots

- A deployment slot technically is an **independent** web app with its own content, configuration, and even a unique host name. So, it functions just like any other web app.
- Each Slot is reachable from its unique URL. For example for Staging deployment slot:
  http://contoso-web.azurewebsites.net/
  *http://contoso-web-**staging**.azurewebsites.net/*
- This option is available only in Standard and Premium pricing tier.

Benefits of Deployment Slots:

1. You can deploy changes for your application to a **staging deployment slot** and test the changes without impacting users who are accessing the **production deployment slot**. When you are ready to move the new features into production, you can just **swap the staging and production** slots **with no downtime**.

2. You can do **A/B testing** with a small set of users to try out new features of your application without impacting the majority of users who are using the production slot.

Note: A/B testing (also known as split testing or bucket testing) is a method of comparing two versions of a webpage or app against each other to determine which one performs better. AB testing uses data & statistics to validate new design changes and improve your conversion rates.

3. You can "**warm up**" your application in a staging slot before swapping it into the production slot, avoiding the long delays a cold start of your application may incur because of some lengthy initialization code.
4. You can **swap back** to the previous deployment if you realize that the new version of your application is not working as you expected.

Configuration for deployment slots

When you clone configuration from another deployment slot, the cloned configuration is editable. Furthermore, some configuration elements will follow the content across a swap (not slot specific) while other configuration elements will stay in the same slot after a swap (slot specific).

Settings that are swapped:	**Settings that are not swapped**:
• General settings - such as framework version, 32/64-bit, Web sockets • App settings (can be configured to stick to a slot) • Connection strings (can be configured to stick to a slot) • Handler mappings • Monitoring and diagnostic settings • WebJobs content	• Publishing endpoints • Custom Domain Names • SSL certificates and bindings • Scale settings • WebJobs schedulers

Adding a Deployment slot:

1. App Services → Select App Service → Settings → Deployment Slots (Publishing) → Add Slot (blade)
2. Set Name and Configuration Source

Format of **Domain Name** for Deployment Slot = <WebApp>-<DeploymentSlotName>.azurewebsite.net

To Swap with Production:

1. Go to Deployment Slot Blade → Swap
2. Set Swap type, Source and Destination.

Note: Make sure that the swap source and swap target are set properly. Usually, the swap target is the production slot.

PowerShell to Manage Slots:

Create a new deployment slot for an existing web app:

$rg = "DemoRG"

$site = "DssDemoWebApp"

New-AzureRmWebAppSlot -ResourceGroupName $rg -name $site -slot **staging**

Specify slot app settings and connection strings

This cmdlet specifies the names of app settings and connection strings that will be treated as slot settings, meaning that they will remain with the slot during swap.

Set-AzureRmWebAppSlotConfigName -ResourceGroupName $rg -name $webApp **-AppSettingNames** "Key1","Key2" -**ConnectionStringNames** "PrimaryDB","SecondaryDB"

Set the app setting values on web app and on its slots:

Set-AzureRmWebApp -ResourceGroupName $rg -Name $site -**AppSettings** @**{**"Key1" = "PValue1"; "Key2" = "PValue2"**}**

Set-AzureRmWebAppSlot -ResourceGroupName $rg -Name $site -Slot staging -AppSettings @**{**"Key1" = "SValue1"; "Key2" = "SValue2"**}**

Perform a regular slot swap:

Switch-AzureRmWebAppSlot -ResourceGroupName $rg -Name $site -SourceSlotName **staging** -DestinationSlotName **production**

Perform swap with preview:

1. First check that the staging slot's app setting have staging values:

   $stagingSite = **Get-AzureRmWebAppSlot** -ResourceGroupName $rg -Name $site -Slot staging

2. Print the current settings

   $stagingSite.SiteConfig.AppSettings

Name	Value
Key1	SValue1
Key2	SValue2

3. Next perform the first step of the swap, which is to apply target slot's setting values to the current staging slot:

   Switch-AzureRmWebAppSlot -ResourceGroupName $rg -Name $site -SourceSlotName staging -DestinationSlotName production -SwapWithPreviewAction ApplySlotConfig

4. Now confirm that the app setting values have changed:

   $stagingSite = **Get-AzureRmWebAppSlot** -ResourceGroupName $rg -Name $site -Slot staging

5. Print the current settings

   $stagingSite.SiteConfig.AppSettings

Name	Value

Key1 PValue1

Key2 PValue2

6. At this point you would want to verify that the web app in the staging slot works as expected. You can make multiple requests to it to warm it up and fill up its caches, etc.
7. Once you are sure the app works as expected call this cmdlet to finish the swap:

   Switch-AzureRmWebAppSlot -ResourceGroupName $rg -Name $site -SourceSlotName staging -DestinationSlotName production -SwapWithPreviewAction CompleteSlotSwap
8. If for any reason you do not want to proceed with the swap you can reset the configuration on the staging slot back to its original state:

   Switch-AzureRmWebAppSlot -ResourceGroupName $rg -Name $site -SourceSlotName staging -DestinationSlotName production -SwapWithPreviewAction ResetSlotSwap

Scaling a ~~Web App~~ App Service Plan

- Whether your application needs to handle a few hundred requests per day or a few million requests per day, the Azure Web Apps scalability features provide ways for you to deliver the right level of scale in a robust, cost-effective manner.
- When you consider the scalability requirements of an application, you should look at its resource requirements **vertically** (scaling up) **horizontally** (scaling out).
- You typically choose to **scale up** when any single request demands more memory and processing power to complete, and the ***bottleneck / latency in the system is the intensive number of software objects created in the computer's memory or the intensive algorithms and business logic that is performed.*** When you scale up a web app, you increase the resource capacity, **such as RAM and CPU cores**, of the virtual machine on which your web app is running.
- You typically **scale out** when any single request requires less memory and processing power to complete, but the real ***bottleneck / latency is in network communication, disk access, etc.*** *In this case, the key to completing each request more efficiently is to run it in parallel to other requests as each wait on external components to complete***.** To scale out a web app, you **increase the number of virtual machine** instances on which your web app is running. For the properly architected app, this means your web app can handle more load and therefore service more user requests.

Scale Up the Azure Web App:

- The ability to scale up a web app exists only for web apps configured for Basic, Standard, or Premium pricing tiers.

1. The scale settings take only seconds to apply and affect all web apps in your App Service plan. They do not require your code to be changed or your applications to be redeployed.

To Scale Up

1. App Services → Select App Service → Settings → **Change App Service Plan** (In App Service Plan) → Select / Create New Plan
2. Select the Pricing tier based on following options:
    a. Number of Cores
    b. RAM
    c. Storage
    d. Slots (Number of CPU Instances)
    e. Backup frequency
    f. Traffic Manager facility

To Scale Out:

The number of Virtual Machine Instances you can scale out is limited by the pricing tier configured for your web app.

1. App Services → Select App Service → Settings → App Service Plan
2. Select Scale Out (App Service Plan) to configure settings
    a. Scale by: **Manual** - Manual setup means that the number of instances you choose won't change, even if there are changes in load.
    b. Scale by: **CPU percentage**: Automatically scale based on CPU Percentage used. You can choose an average value you want to target.
    c. Scale by: **Schedule and Performance Rules** - Create your own set of rules. Create a schedule that adjusts your instance counts based on time and performance metrics.

Auto scale based on CPU percentage:

- The Target range setting defines the minimum and maximum CPU percentage to target.
- As long as the CPU percentage is within this range, Autoscale will not increase or decrease the number of instances.
- When the CPU percentage exceeds the maximum CPU percentage you specify, Autoscale will add an instance. If CPU percentage continues to exceed the maximum CPU specified, then Autoscale will add another instance. At no point will you have more than the maximum number of instances specified in the Instances setting.
- Similarly, when CPU percentage falls below the minimum CPU percentage you specify, Autoscale will remove an instance. If CPU percentage continues to all below the minimum CPU percentage specified, then Autoscale will remove another instance. At no point will you have fewer than the minimum number of instances specified in the Instances setting

Note: The CPU percentage is measured as an **average across all instances**. For example, if you have two instances, one of which is running at 50 percent CPU and the other of which is running at 100 percent CPU, then the CPU percentage would be 75 percent for all the instances at that point in time

Auto scale based on a recurring schedule:

This can be particularly useful when demand for your web app is predictable. For example, if your web app provides services for an industry where most work is done Monday through Friday, then you could configure Autoscale to increase the number of instances during the week to support peak demand and decrease the number of instances on weekends when demand is very light.

App Service Authentication and Authorization

Azure App Service provides built-in authentication and authorization support, so you can sign in users and access data by writing minimal or no code in your web app, RESTful API, and mobile back end, and also Azure Functions. App Service uses federated identity, in which a third-party identity provider manages the user identities and authentication flow for you. Five identity providers are available by default:

Provider	Sign-in endpoint
Azure Active Directory	/.auth/login/aad
Microsoft Account	/.auth/login/microsoftaccount
Facebook	/.auth/login/facebook
Google	/.auth/login/google
Twitter	/.auth/login/twitter

Authentication flow

The authentication flow is the same for all providers, but differs depending on whether you want to sign in with the provider's SDK:

- **Without provider SDK:** The application delegates federated sign-in to App Service. This is typically the case with **browser apps**, which can present the provider's login page to the user. The server code manages the sign-in process, so it is also called *server-directed flow* or *server flow*. This case applies to browser apps. It also applies to native apps that sign users in using the Mobile Apps client SDK because the SDK opens a web view to sign users in with App Service authentication.
- **With provider SDK**: The application signs users in to the provider manually and then submits the authentication token to App Service for validation. This is typically the case with **browser-less apps**, which can't present the provider's sign-in page to the user. The application code manages the sign-in process, so it is also called *client-directed flow* or *client flow*. This case applies to REST APIs, Azure Functions, and JavaScript

browser clients, as well as browser apps that need more flexibility in the sign-in process. It also applies to native mobile apps that sign users in using the provider's SDK.

Step	Without provider SDK	With provider SDK
1. Sign user in	Redirects client to `/.auth/login/<provider>`.	Client code signs user in directly with provider's SDK and receives an authentication token.
2. Post-authentication	Provider redirects client to `/.auth/login/<provider>/callback`.	Client code posts token from provider to `/.auth/login/<provider>` for validation.
3. Establish authenticated session	App Service adds authenticated cookie to response.	App Service returns its own authentication token to client code.
4. Serve authenticated content	Client includes authentication cookie in subsequent requests (automatically handled by browser).	Client code presents authentication token in `X-ZUMO-AUTH` header (automatically handled by Mobile Apps client SDKs).

Register your application with Facebook

1. Navigate to https://developers.facebook.com/
2. (Optional) If you have not already registered, click **Apps** > **Register as a Developer**, then accept the policy and follow the registration steps.
3. My Apps → Add a New App
4. Display Name = "Demo Web App", contact email = <your login> → Create App ID
5. +Add Product → Facebook Login → Set Up → Web
6. Facebook Login → Settings → Client Oauth Settings Section, **Valid OAuth redirect URIs** = https://**dssdemods**.azurewebsites.net/.auth/login/facebook/callback
7. Settings → Basics → Copy and Store Application ID and App Secret,
8. Settings → Basics
    1. Set Privacy Policy URL = https://www.bestdotnettraining.com/pdf/PrivacyAgreement.pdf

Terms of Service URL = https://www.bestdotnettraining.com/pdf/TermsAndConditions.pdf

    2. Save Changes
9. Make App public: Switch Status from Development to Live

Add Facebook information to your application

10. App Service → Settings → **Authentication / Authorization**
11. App Service Authentication = On
12. **Action to take when request is not authenticated** = **Facebook**.

13. Click **Facebook**, paste in the App ID and App Secret values which you obtained previously.
14. Save.
15. Navigate to web app and note that you are redirected to Facebook to Authenticate.

Steps For Google Authentication: https://docs.microsoft.com/en-us/azure/app-service/configure-authentication-provider-google

Note: If required you can use ***ClaimsPrincipal.Current*** to get all claims from the Authentication Provider.

Back up your app in Azure

The Backup and Restore feature in Azure App Service lets you easily create app backups manually or on a schedule. You can restore the app to a snapshot of a previous state by overwriting the existing app or restoring to another app.

App Service can back up the following information to an Azure storage account and container that you have configured your app to use.

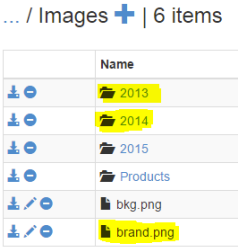
- App configuration
- File content
- Database connected to your app

Backups can be up to 10 GB of app and database content. If the backup size exceeds this limit, you get an error.

1. Azure Portal → App Services → Select the App Service → Settings → Backups
2. Configure the Backup
    - Backup Storage
    - Backup Schedule
    - Backup Database
3. Backup page → Click Backup Button
4. You see a progress message during the backup process.

Configure Partial Backups

1. Click **Advanced Tools -> Go** setting for your web app to access Kudu
2. Identify the folders that you want to exclude from your backups. For example, you want to filter out the highlighted folder and files.



3. Create a file called **_backup.filter** and put the preceding list in the file, but remove D:\home. List one directory or file per line. So the content of the file should be:

```
\site\wwwroot\Images\brand.png
\site\wwwroot\Images\2014
\site\wwwroot\Images\2013
```

Upload _backup.filter file to the D:\home\site\wwwroot\ directory of your site using **ftp** or any other method. If you wish, you can create the file directly using Kudu DebugConsole and insert the content there.

Monitoring, Debugging and Diagnostics

- Visual Studio, and the Azure platform collectively provide a rich set of services and tools that you can use to monitor and troubleshoot your applications.
- You can monitor your application **in real time** and interact with **near-real-time data** using the Azure portal. Or you can have the platform alert you if performance degrades or your application becomes unavailable.
- If you need to debug your app live or post-mortem, you will find the Azure Web Apps platform rich with data and analysis to get you to the root cause and resolution as fast as possible.

Remote debugging:

It enables you to debug your web app interactively while it is running in Azure.

Step1: Enable Remote Debugging

In Portal

Settings → Application Settings → Debugging →

Select Remote Debugging = On,

Select Remote Visual Studio version = 2012/2013/2015/2017

Step 2: Publish the **Debug Build** Configuration

Step 3: In Server Explorer, right-click the web app and **select Attach Debugger**.

Diagnostic logs for a web app generally fall into one of two categories:

1. **Application Diagnostic Logs** are generated as a result of **logging code** you add to your application
2. **Site diagnostic logs** are generated automatically by a monitoring service configured on the web server on which your web app is running.

The types of logs that can be enabled for a web app are as follows:

1. Application Logging

- These are logs that are written specifically from your application code using diagnostic classes such as the **System.Diagnostics.Trace** or the **System.Diagnostics.Debug**.
- When you enable application logging, you also must specify a **log level**, which can be **Error, Warning, Information or Verbose.**

```
public ActionResult Index()
{
   Trace.TraceError("Something is definitely wrong here.");
   Trace.TraceWarning("Something could be wrong here.");
   Trace.TraceInformation("Entered {0}.", this.GetType().Name);
   Debug.WriteLine("This is a debug only trace message.");
   return View();
}
```

2. Web Server Logging

- These are HTTP logs (that is, IIS logs) that are written by the web server on which your web app is running.
- Data in these logs contains fields defined in the W3C extended log file format defined at https://msdn.microsoft.com/library/windows/desktop/aa814385.aspx and includes things such as the **time it took** the server to process a request, **cookies** that were sent to the client or received by the client, the **client's IP address**, and much more.

3. Detailed error messages

- These are HTML files written by the web server for any requests to the server that result in an HTTP status code 400 or above response. For example, if you request a resource that doesn't exist on the server, you will get an HTTP 404 (Not Found) response.
- With detailed error messages enabled, an HTML file also will be generated containing suggested causes, possible solutions, and additional details about the request.

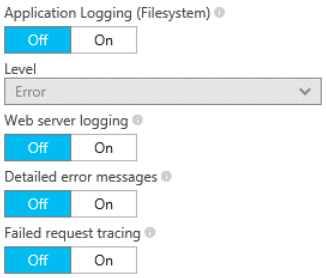
4. Failed request tracing

- These are XML files written by the web server containing a deeper level of trace information for failed requests.
- These logs contain visibility into the HTTP modules that were invoked when processing the request, time taken in each module, module tracing messages, and more.

- A new XML file is generated for each failed request and is named **fr<x>.xml** where <x> is an incrementing number. Failed request logs are intended to be viewed using a browser, and Azure Web Apps facilitates this by generating a **style sheet file** named in the directory where these files are stored.

Enable application and site diagnostic logs:

App Service → Settings → Diagnostic Logs → Enable features as required



Log files for an Azure web app are stored on the web server's file system hosting your web app.

1. Application logs: D:\home\LogFiles\application
2. Web server logs: D:\home\LogFiles\http\RawLogs
3. Detailed error messages: D:\home\LogFiles\DetailedErrors
4. Failed request tracing: D:\home\LogFiles\W3SVC<x>, where <x> is a random number

Note: To Access log files stored in the web app file system either FTP can be used or using Server Explorer in VS.NET

OR

Azure Portal → Select Web App → Settings → Console (Development tools) → Change to directory as required.

Log Stream:

The log-streaming service in Azure Web Apps enables you to view application logs, web server logs, and detailed error messages in nearly real time.

Azure Portal → Select Web App → Settings → Log Stream

Log streaming using Visual Studio:

Server Explorer → Double Click App Service → Right Click on Web App → View Streaming Logs

In the Output window, you will see a message stating **You Are Now Connected to Log-Streaming Service**.

Note: You can change the logs the log-streaming service monitors by clicking the gear icon in the toolbar.

Using Site Control Manager (Kudu) to retrieve log files

Site Control Manager, often referred to as "Kudu", is a website extension that you can use to retrieve log files, browse the file system, edit files, delete files, view environment variables, and even capture diagnostic dump files.

To access the Site Control Manager, open your browser and navigate to

https://<your site name>.scm.azurewebsites.net

App Service Environment

The Azure App Service Environment is an Azure App Service feature that provides a **fully isolated and dedicated environment** for securely running App Service apps at high scale.

App Service environments (ASEs) are appropriate for application workloads that require:

- Very high scale.
- Isolation and secure network access.
- High memory utilization.

ASEs are isolated to running only a single customer's applications and are always deployed into a virtual network. Customers have fine-grained control over inbound and outbound application network traffic. Applications can establish high-speed secure connections over VPNs to on-premises corporate resources.

Apps also frequently need to access corporate resources such as internal databases and web services. If you deploy the ASE in a virtual network that has a VPN connection to the on-premises network, the apps in the ASE can access the on-premises resources.

Scaling ASE

- An ASE is dedicated exclusively to a single subscription and can host 100 App Service Plan instances. The range can span 100 instances in a single App Service plan to 100 single-instance App Service plans, and everything in between.
- ASE is composed of front ends and workers. Front ends are responsible for HTTP/HTTPS termination and automatic load balancing of app requests within an ASE. Front ends are automatically added as the App Service plans in the ASE are scaled out.
- Workers are roles that host customer apps. Workers are available in three fixed sizes:
    - One vCPU/3.5 GB RAM
    - Two vCPU/7 GB RAM
    - Four vCPU/14 GB RAM
- Customers do not need to manage front ends and workers. All infrastructure is automatically added as customers scale out their App Service plans. As App Service plans are created or scaled in an ASE, the required infrastructure is added or removed as appropriate.

Pricing ASE

There is a flat monthly rate for an ASE that pays for the infrastructure and doesn't change with the size of the ASE. In addition, there is a cost per App Service plan vCPU. All apps hosted in an ASE are in the Isolated pricing SKU.

There are two ways to deploy an App Service environment (ASE):

- With a VIP on an external IP address, often called an External ASE.
- With the VIP on an internal IP address, often called an ILB ASE because the internal endpoint is an internal load balancer (ILB).

NOTE1: After you create your ASE, you can't change the following:

- Location
- Subscription
- Resource group
- VNet used
- Subnet used
- Subnet size

NOTE2: When you choose a VNet and specify a subnet, make sure that it's large enough to accommodate future growth and scaling needs. We recommend a size of `/24` with 256 addresses.

Create an ASE and an App Service plan together

1. **Create a resource** > **Web + Mobile** > **Web App**.
2. App Service Plan → Create New → Select **Pricing tier**, and choose one of the **Isolated** pricing SKUs. If you choose an **Isolated** SKU and a location that's not an ASE, a **new ASE** is created in that location.
3. Specify your Azure virtual networking details. Select either **Create New** or **Select Existing**.
   - If you select **Create New**, enter a name for the VNet. A new Resource Manager VNet with that name is created. The new VNet has the address range 192.168.250.0/23 and a subnet named default. The subnet is defined as 192.168.250.0/24.
   - If you select **Select Existing**, you need to:
     a) Select the VNet address block, if you have more than one.
     b) Enter a new subnet name.
     c) Select the size of the subnet.

     Remember to select a size large enough to accommodate future growth of your ASE. We recommend `/25`, which has 128 addresses and can handle a maximum-sized ASE. We don't recommend `/28`, for example, because only 16 addresses are available. Infrastructure uses at least seven addresses and Azure Networking uses another 5. In a `/28` subnet, you're left with a

maximum scaling of 4 App Service plan instances for an External ASE and only 3 App Service plan instances for an ILB ASE.

d) Select the subnet IP range.

4. Create (THIS WOULD TAKE AROUND 60 to 70 MINUTES)

Create and use an internal load balancer with an App Service Environment

You can deploy an ASE with an IP address in your VNet.To set the IP address to a VNet address, the ASE must be deployed with an ILB. When you deploy your ASE with an ILB, you must provide:

- Your own domain that you use when you create your apps.
- The certificate used for HTTPS.
- DNS management for your domain.

In return, you can do things such as:

- Host intranet applications securely in the cloud, which you access through a site-to-site or Azure ExpressRoute VPN.
- Host apps in the cloud that aren't listed in public DNS servers.
- Create internet-isolated back-end apps, which your front-end apps can securely integrate with.

Steps

1. **Create a resource** > **Web Mobile** > **App Service Environment**.
2. Enter the name of your ASE
3. Select your subscription. This subscription is also the one that all apps in the ASE use. You can't put your ASE in a VNet that's in another subscription.
4. Select or specify a new resource group. The resource group used for your ASE must be the same one that's used for your VNet. If you select an existing VNet, the resource group selection for your ASE is updated to reflect that of your VNet. *You can create an ASE with a resource group that is different from the VNet resource group if you use a Resource Manager template.*
5. Select your VNet and location. You can create a new VNet or select an existing VNet
6. The **VIP Type** selection determines if your ASE can be directly accessed from the internet (External) or if it uses an ILB.
   - If you select **External** for the **VIP Type**, you can select how many external IP addresses the system is created with for IP-based SSL purposes.
   - If you select **Internal** for the **VIP Type**, you must specify the domain that your ASE uses. You can deploy an ASE into a VNet that uses public or private address ranges. To use a VNet with a public address range, you need to create the VNet ahead of time.
7. Enter a domain name = contoso-internal.com

The custom domain name used for apps and the domain name used by your ASE can't overlap.

For example, you can use something like ***contoso-internal.com*** for the domain of your ASE so that won't conflict with custom domain names for app like www.***contoso.com***.

8. Create a Self Signed Certificate: **Open PowerShell in Administrator mode**

```
$certificate = New-SelfSignedCertificate -certstorelocation cert:\localmachine\my -dnsname "*.internal-contoso.com","*.scm.internal-contoso.com"
$certThumbprint = "cert:\localMachine\my\" +$certificate.Thumbprint
$password = ConvertTo-SecureString -String "CHANGETHISPASSWORD" -Force -AsPlainText
$fileName = "exportedcert.pfx"
Export-PfxCertificate -cert $certThumbprint -FilePath $fileName -Password $password
```

9. ILP ASE → Settings → ILB Certificate → Set ILB Certificate → Select the certificate .pfx file and enter the password
10. To ASE add Web App with App Service plan = <ILB ASE as created above>
11. Create a VM if you don't have one in that VNet. (Don't try to create this VM in the same subnet as the ASE because it will fail or cause problems.)
12. Get the ILB address for your ASE. Select **ASE → Properties → Virtual IP Address**.
13. RDP to VM → Edit Hosts file in folder *C:\Windows\System32\drivers\etc*
14. Set *mytestapp.internal-contoso.com to* IP Address found above
15. Use a browser on the VM and go to https://*mytestapp.internal-contoso.com*.