Cloud migration document

Contents

[Requirement- Move current database to cloud 2](#_Toc530956293)

[Firewall configuration 2](#_Toc530956294)

[Moving DB to Azure SQL Server VM 2](#_Toc530956295)

[Connecting to remote SQL Server from SQL management studio 4](#_Toc530956296)

[References 4](#_Toc530956297)

[Requirement – Hosting Web Application on Azure 5](#_Toc530956298)

[Custom domain name 5](#_Toc530956299)

[References 6](#_Toc530956300)

[Changes required in app to share cookie & session across the apps 6](#_Toc530956301)

[References 6](#_Toc530956302)

[Requirement-Application configurations 6](#_Toc530956303)

[Configuration transformation 7](#_Toc530956304)

[References 8](#_Toc530956305)

[Apply transformation at the time of deployment. 9](#_Toc530956306)

[Requirement-File share 10](#_Toc530956307)

[Handling files from .Net applications 11](#_Toc530956308)

[References 12](#_Toc530956309)

[Requirement – tracing & Logging 12](#_Toc530956310)

[Tracing 12](#_Toc530956311)

[Logging 13](#_Toc530956312)

[References 14](#_Toc530956313)

[Reporting – Using SSRS 15](#_Toc530956314)

[Configure reporting service 15](#_Toc530956315)

[Access provisioning 16](#_Toc530956316)

[Publish reports 17](#_Toc530956317)

[Render report in MVC app 17](#_Toc530956318)

[Reference: 18](#_Toc530956319)

[Azure Storage- file management 19](#_Toc530956320)

[Reference 20](#_Toc530956321)

[Requirement – Importing excel file 21](#_Toc530956322)

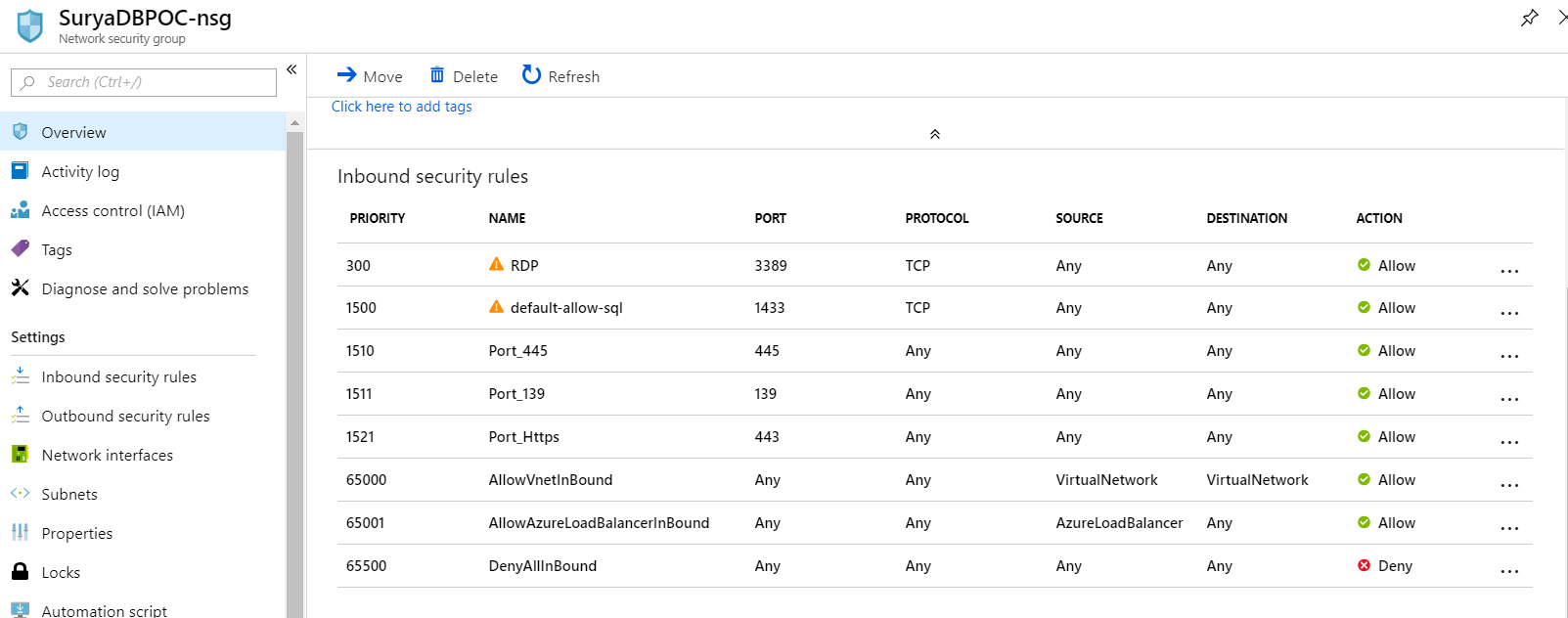
Cloud migration guidelines

# Requirement- Move current database to cloud

Out of the available options to port existing on-prem SQL server database to Azure, the easiest option is to use SQL server hosted on a VM (IaaS). In this model, SQL server database would be preinstalled on Azure VM running Windows Server OS.

## Firewall configuration

For remote connection to work make sure following port\services are enabled in Network Service Group.



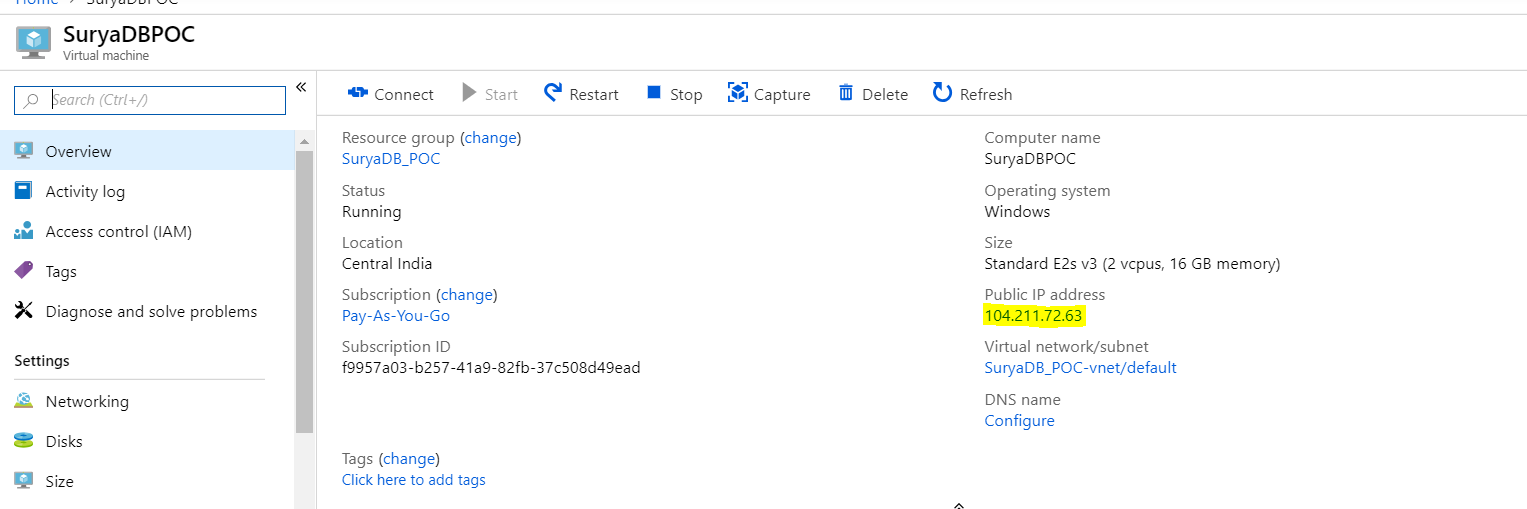
|  |  |
| --- | --- |
| **Port** | **Description** |
| 3389 | Enable this port for remote desktop protocol (rdp) to work. |
| 1433 | This is the SQL database port. |
| 445,139 | File sharing port. These ports are required for accessing shared folders remotely. |
| 443 | Required to access any web sites hosted on the VM over https. |

## Moving DB to Azure SQL Server VM

Create an Azure VM including SQL server standard edition with following configuration

*E2s v3 (2 vCPU(s), 16 GB RAM)*

As part of the configuration make sure to opt for public IP so that you can connect to VM remotely. Public IP of the VM can be obtained from its “Overview” page.



To port the Database, you can use one of the following options.

* Create a mapped drive to VM to f:\ , copy database files(.mdf) to VM & restore it on VM using SQL management studio.

|  |  |
| --- | --- |
|  |  |

* Place database files to be copied on VM on your local machine. In the

|  |  |
| --- | --- |
|  |  |
| Click on “Show options” button | Click on More button & select the local drive that you want to access from the VM. |
|  | |
| When you login into VM, you can see the shared drive.Now, copy the database files from shared drive onto VM for restoring. Connecting to remote SQL Server from SQL management studio SQL Server running on Azure VM can be connected from local instance of SQL management studio using VMs IP and SQL user configured in the database. | |

### References

Database restoration - https://campus.barracuda.com/product/backup/doc/15892599/how-to-restore-a-microsoft-sql-database-to-a-point-in-time/

# Requirement – Hosting Web Application on Azure

The existing Web applications can be hosted as Web App services in Azure & following are the different options available under App services.

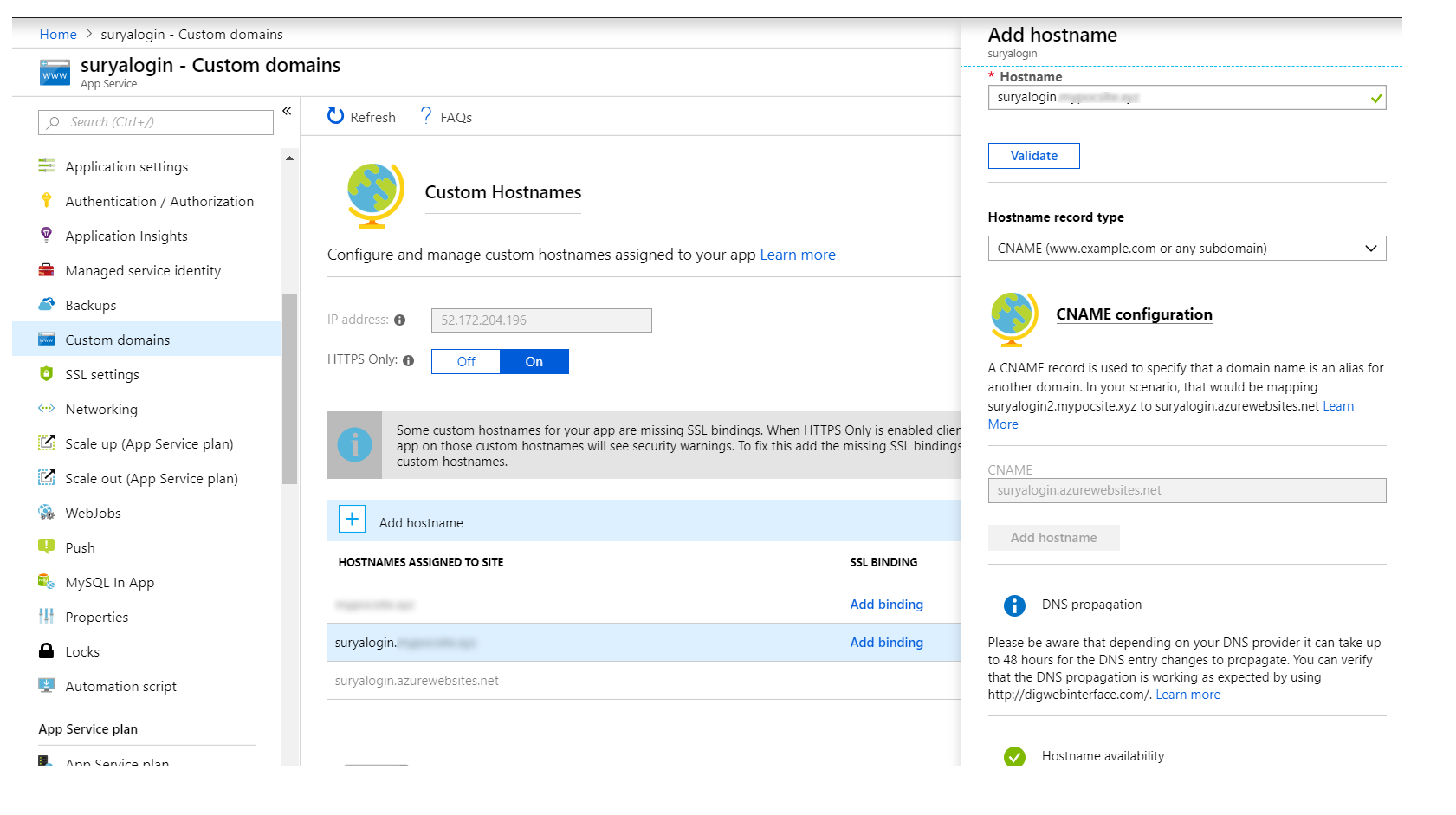
To start with Shared tier could be used and Standard tier can be opted in future based on requirement.

|  |  |
| --- | --- |
| **Tier** | **Description** |
| Free | Use the Free plan to quickly evaluate App Service and convert the app at any time to one of the paid plans without delays or downtime. |
| Shared | Develop and test in an environment using features such as custom domain names and more, before moving to production. **Shared is also suitable for low traffic sites.** In the Shared tier, each app receives a quota of CPU minutes, so each app is charged hourly for the CPU quota. |
| Basic | Designed for apps that have lower traffic requirements and do not need more advanced autoscale and traffic management features. |
| Standard | Designed for production API, Mobile and Web apps. There are no limits on the number of apps/domains you can host. |
| Premium | Designed for production apps. Supports larger numbers of scale instances, additional connectors and BizTalk capabilities, while including all the advanced capabilities found in the Standard plan. |

## Custom domain name

Since the current Web applications required to share session and cookie its suggested to configure custom domain names for these application since sharing of cookie is not allowed for \*.azurewebsites.net site.

For assigning custom domain name use the Custom domains menu in the app service window. Click on “Add hostname” button to add custom domain name to the app service. You need to first create A & CName records using your Domain name provider site & later try to assign custom host names using below shown window.



### References

App service plans-<https://docs.microsoft.com/en-us/azure/app-service/azure-web-sites-web-hosting-plans-in-depth-overview>

Custom domain name configuration - <https://github.com/Huachao/azure-content/blob/master/articles/app-service-web/web-sites-custom-domain-name.md>

## Changes required in app to share cookie & session across the apps

For sharing the cookie and sessions across the app make sure domain name is set for application & form authentication cookies.

Also, change session settings to use SQL server based session so that apps can share common session.

|  |
| --- |
| <httpCookies domain="mydomain.com"/>  <sessionState mode="SQLServer" sqlConnectionString="Data Source=104.211.72.63;Integrated Security=false; User Id=dbusername; pwd=XXXXXX" cookieless="false" timeout="20" />  <authentication mode="Forms">  <forms loginUrl="https://suryalogin.mydomain.com/Login/Account/Login" name=".ASPXFORMSAUTH" protection="Validation" timeout="40000" enableCrossAppRedirects="true" domain="mydomain.com" />  </authentication> |

In the above configuration 104.211.72.63 is the IP of the VM hosting SQL server DB on azure. Make the above changes in all the web.config file of web applications to be hosted as Web App service.

For using SQL server as session state repository, refer the steps explained in “Configure SQL Server to Store ASP.NET Session State” link. By following the steps explain in the link, you will end up creating a new database ASPState which would be used to track sessions.

### References

Configure SQL Server to Store ASP.NET Session State - <https://support.microsoft.com/en-in/help/317604/how-to-configure-sql-server-to-store-asp-net-session-state>

# Requirement-Application configurations

Currently application is using web.config to store following category of settings.

* Database connection strings
* Appsettings
  + Constant values (Ex: Application name, Email address, Passwords)

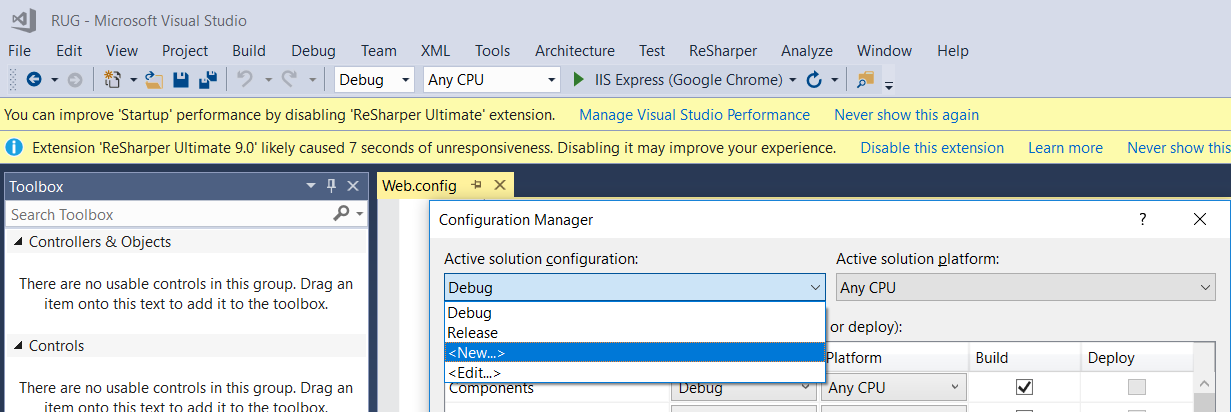
Few of these settings would vary based on the environment & having mechanism to maintain these values specific to environment & automatically applying these settings as part of the deployment would make the release process easier.

## Configuration transformation

Visual studio provides inbuilt feature to apply configuration transformation at the time of deployment.

Follow the below steps to use auto configuration transformation using visual studio.

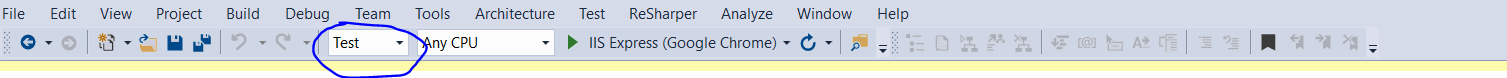
* Open the configuration manager window(Build 🡪 Configuration Manager)



* Create new solution configurations. Name it Test & Prod for production. Use appropriate “Copy settings from” option to copy existing build settings. Generally, test environment can have Debug version which would assist in troubleshooting issues. For production, Release settings can be used which generates optimized version of the application.

|  |  |
| --- | --- |
|  |  |
| **For test release** | **For production release** |

* Now, select appropriate build configuration in the solution configuration dropdown.



* Right click on Web.config & select “Add Config Transform” menu from the context menu.

|  |  |
| --- | --- |
|  | This would create Web.Test.Config since current build configuration selected is Test. Similary select Prod & create Web.Prod.config file.    You final transformation files would look something like above. |

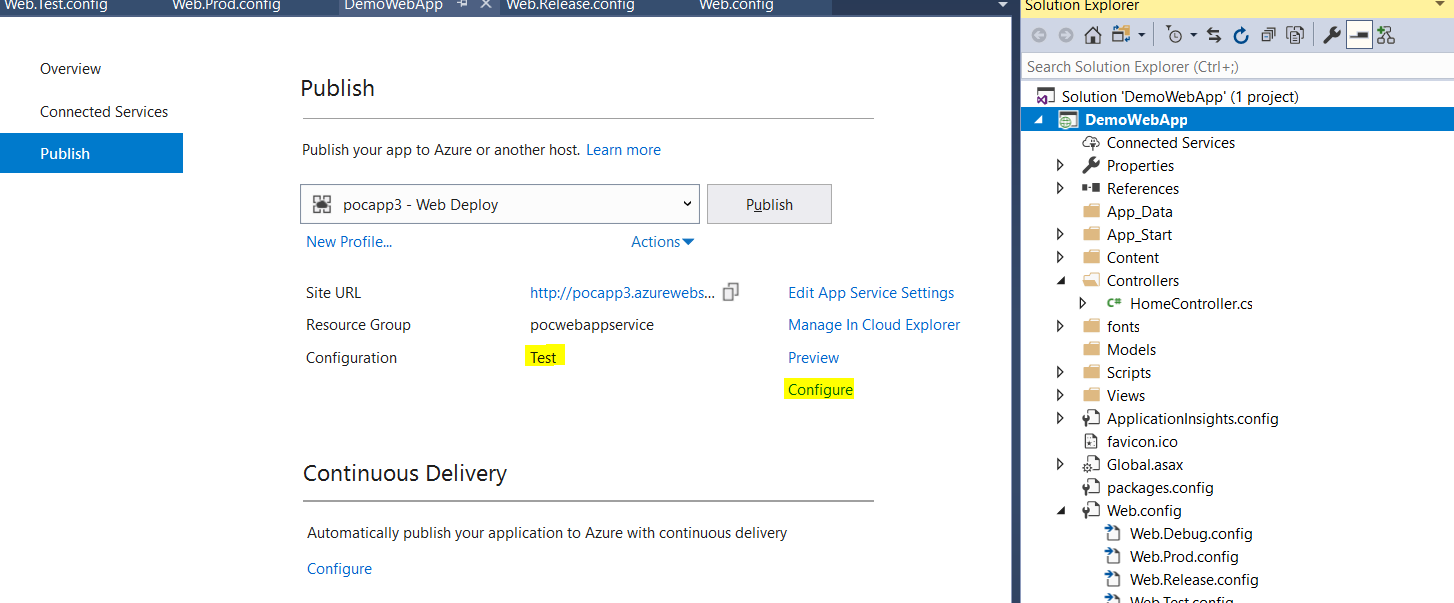
* Web.config is the master configuration file upon which transformation would be applied from Web.Test.Config & Web.Prod.Config at the time of deployment. Edit Web.Test.config & Web.Prod.config to apply necessary transformation appropriate to environment. Refer below link to learn about web transformation.

### References

<https://docs.microsoft.com/en-us/previous-versions/aspnet/dd465326(v=vs.110)>

## Apply transformation at the time of deployment.

To publish the application right click on any web application and select Publish menu option. In the publish screen select appropriate build configuration.



To change the configuration type, click on Configure link and change the configuration to appropriate value.

|  |  |
| --- | --- |
|  | You can preview the transformation before deployment to make sure transformation is working appropriately by selecting Preview Transformation menu option. |
|  | |

Applied transformation is displayed in comparison with the original Web.config file.

|  |  |
| --- | --- |
|  | Save &Click on publish button to initiate the deployment. |

# Requirement-File share

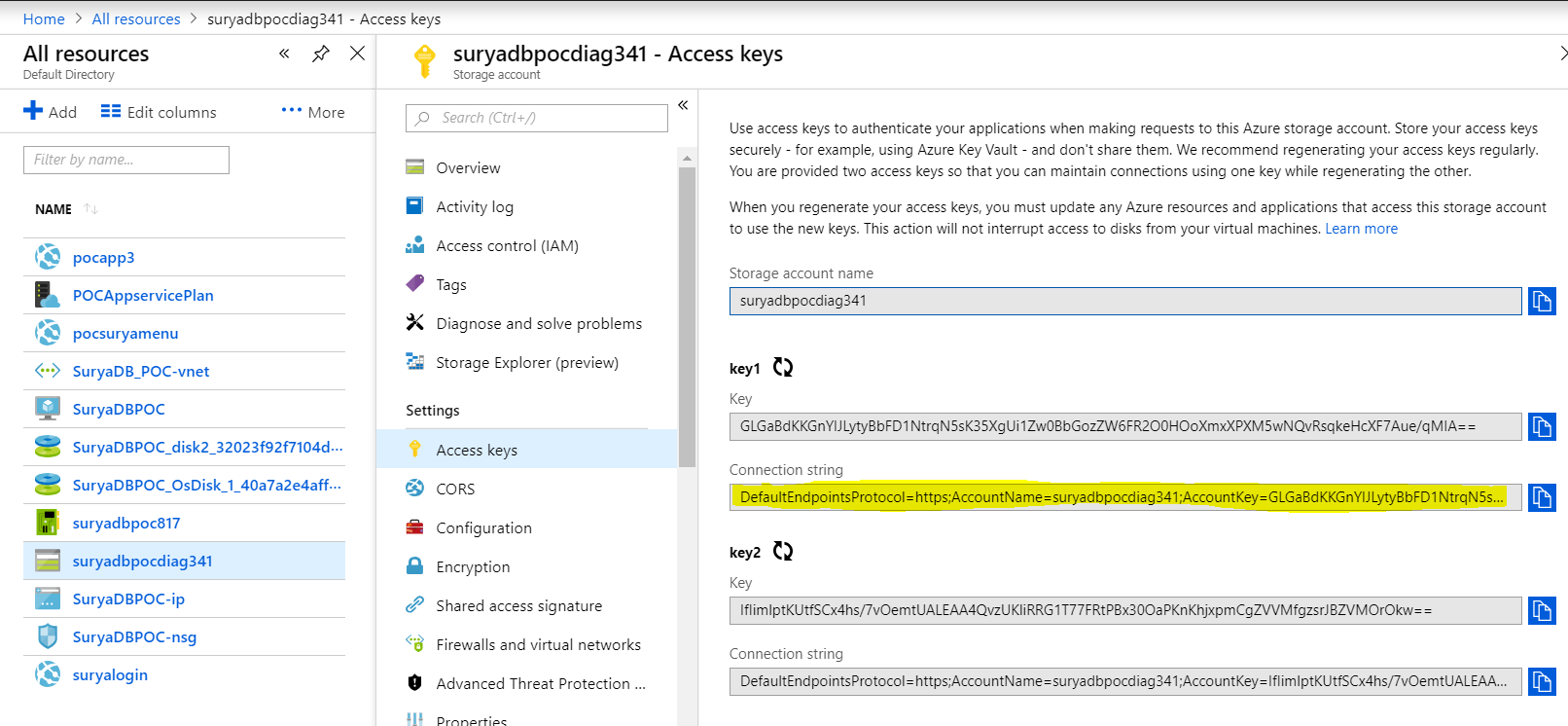
The current application requires storage primarily used for storing and retrieving uploaded files. Following are the available storage options in Azure & Azure file share is recommended since it’s the easiest option to move current files to cloud and it could also be attached to any machines as network drive if required.

| **Feature** | **Description** | **When to use** |
| --- | --- | --- |
| **Azure Files** | Provides an SMB interface, client libraries, and a [REST interface](https://docs.microsoft.com/en-us/rest/api/storageservices/file-service-rest-api) that allows access from anywhere to stored files. | You want to "lift and shift" an application to the cloud which already uses the native file system APIs to share data between it and other applications running in Azure. You want to store development and debugging tools that need to be accessed from many virtual machines. |
| **Azure Blobs** | Provides client libraries and a [REST interface](https://docs.microsoft.com/en-us/rest/api/storageservices/blob-service-rest-api)that allows unstructured data to be stored and accessed at a massive scale in block blobs. | You want your application to support streaming and random-access scenarios.  You want to be able to access application data from anywhere. |
| **Azure Disks** | Provides client libraries and a [REST interface](https://docs.microsoft.com/en-us/rest/api/compute/manageddisks/disks/disks-rest-api)that allows data to be persistently stored and accessed from an attached virtual hard disk. | You want to lift and shift applications that use native file system APIs to read and write data to persistent disks.  You want to store data that is not required to be accessed from outside the virtual machine to which the disk is attached. |

## Handling files from .Net applications

Azure provides client libraries to manage files in Azure storage service. *WindowsAzure.Storage* nugget package can be used from .Net application to performance various storage operations.

Connecting to Azure storage requires connection string which could be obtained from storage 🡪 Access Keys window.



In below shown code snippet storage connection string is configured in **storagekey** appsetting key.

|  |
| --- |
| [HttpPost]  **public** **async** Task < ActionResult > UploadFile(HttpPostedFileBase fileupd) {  **if** (fileupd.ContentLength > **0**) {  **if** (fileupd.ContentLength > **1024**) {  ViewBag.OperationResponseMsg = "File shouldn't be more than 1KB";  **return** **RedirectToAction**("About");  }  **var** storageAccount = CloudStorageAccount.Parse(ConfigurationManager.AppSettings["storagekey"]);  **var** fileClient = storageAccount.CreateCloudFileClient();  CloudFileShare share = fileClient.GetShareReference("fileupload");  **if** (share.Exists()) {  // Get a reference to the root directory for the share.  CloudFileDirectory rootDir = share.GetRootDirectoryReference();  // Get a reference to the directory we created previously.  CloudFileDirectory uploadDir = rootDir.GetDirectoryReference("Uploads");  // Get a reference to the file we created previously.  CloudFile file = uploadDir.GetFileReference(Guid.NewGuid().ToString("N"));  file.Metadata["filename"] = Path.GetFileName(fileupd.FileName);  **await** file.UploadFromStreamAsync(fileupd.InputStream);  ViewBag.OperationResponseMsg = "File uploaded successfully.";  }  **return** **RedirectToAction**("About");  } |

### References

<https://docs.microsoft.com/en-us/azure/storage/files/storage-dotnet-how-to-use-files>

# Requirement – tracing & Logging

Often, we encounter situation wherein we may have to log the program flow for troubleshooting issues. In such cases, enabling tracing would help the developers in knowing the control flow. Whereas logging would provide detailed information which can help in further troubleshooting the issue.

Logging is something that would be always kept turned on and tracing can be enabled on demand at the time of troubleshooting.

## Tracing

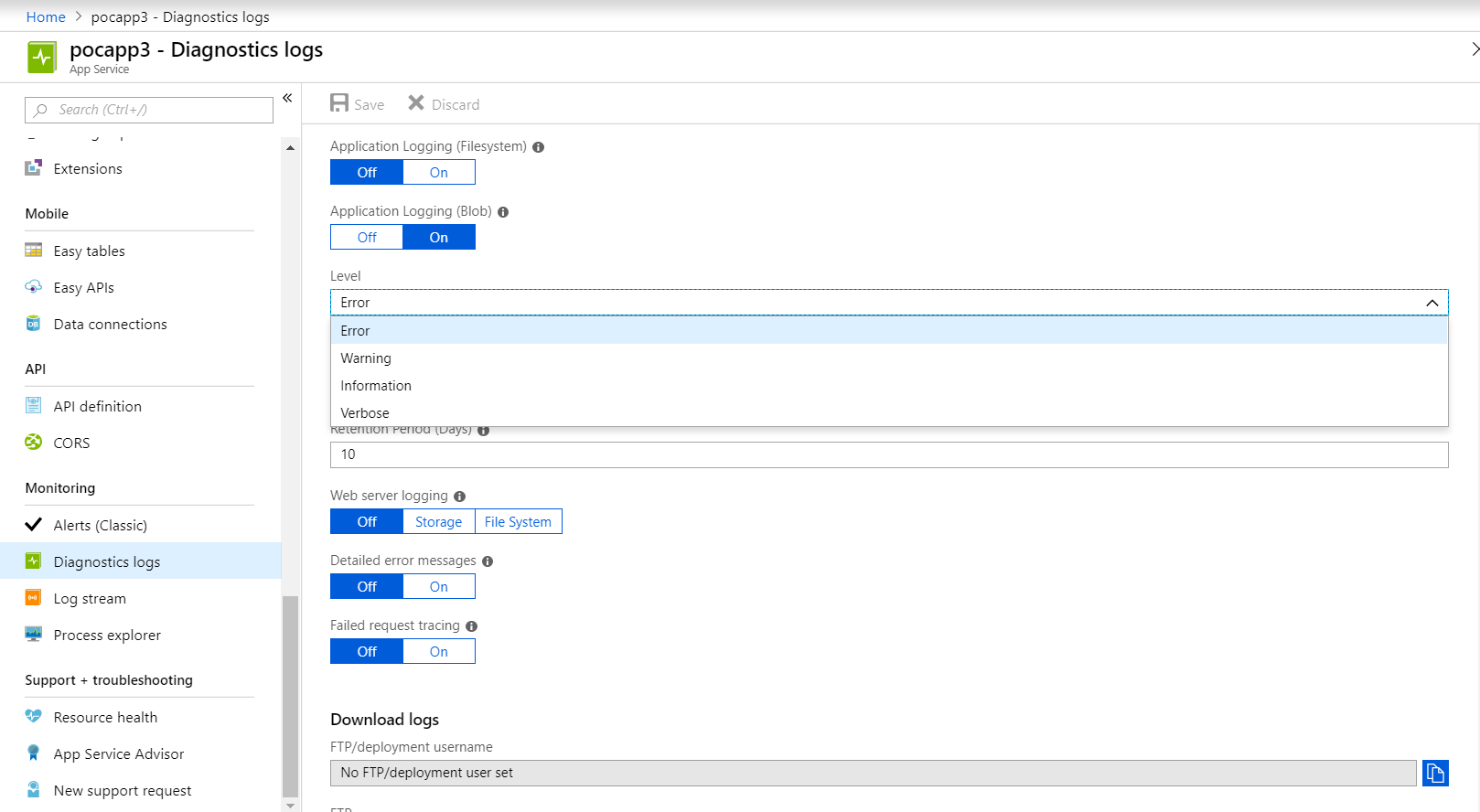
Tracing can be used to log information like method that gets called, parameters that gets passed, important checks performed & its response etc.

In .net, tracing can be easily implemented by using Trace classes which provides apis for logging different category of information namely Information, warning & error.

|  |
| --- |
| System.Diagnostics.Trace.TraceInformation(…);  System.Diagnostics.Trace.TraceWarning(…);  System.Diagnostics.Trace.TraceError(…); |

Since tracing is mainly used to track program flow its always recommend to associate a unique id which gets logged along with other information so that we can differentiate trace logs b/w users or scenarios.

|  |
| --- |
| //In session start or after login store traceid into session  Session["traceid"]= Guid.NewGuid();  //create Trace initializer attribute which restores traceid from session to Trace. CorrelationManager.ActivityId  public class TraceInitAttribute : ActionFilterAttribute  {  public override void OnActionExecuting(ActionExecutingContext filterContext)  {  if (filterContext.HttpContext.Session["traceid"] is Guid traceidguid)  {  System.Diagnostics.Trace.CorrelationManager.ActivityId = traceidguid;  }  base.OnActionExecuting(filterContext);  }  }  //Decorate the controllers with trace init attribute so that controllers can call trace methods which would now log co-relation id which would be handy in trace the program flow  [TraceInitAttribute]  public class LoggerController : Controller  {  ….  …. |



## Logging

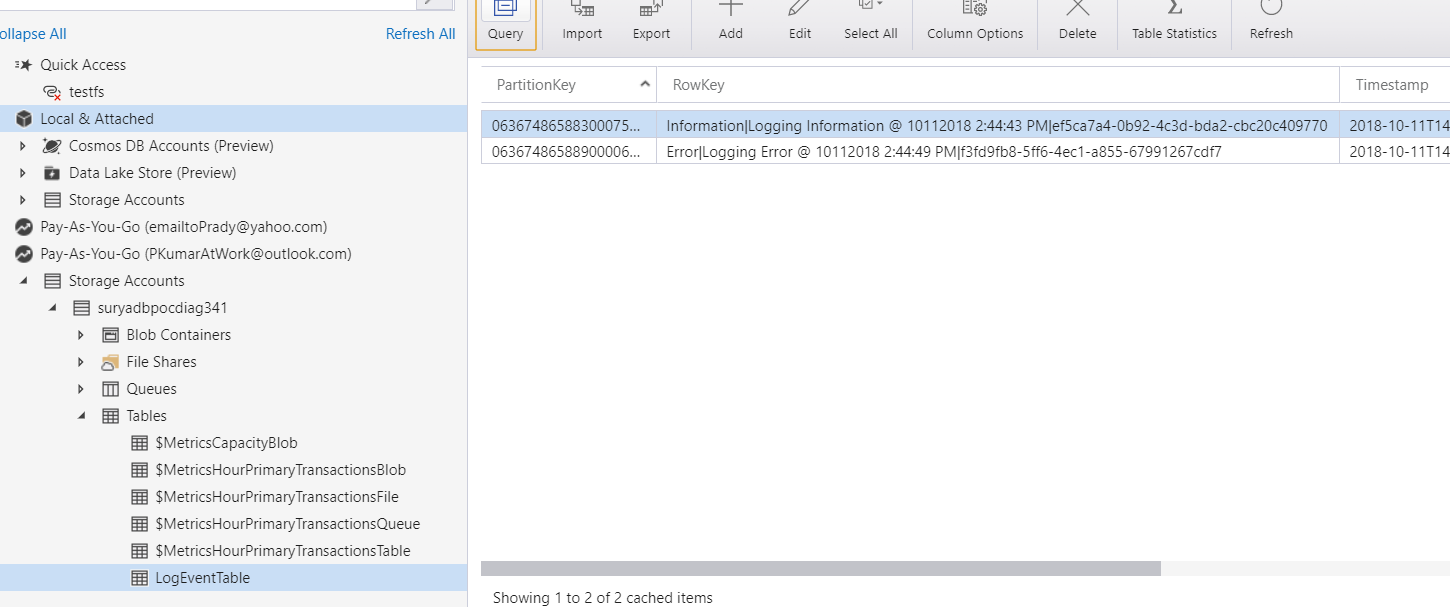
For logging implementation, we can use Serilog component by referring to following nuget packages.

* Serilog
* Serilog.Sinks.AzureTableStorage
* unity.mvc 🡨 required for dependency injection.

Initialize serilog component within the UnityConfig class thats get added as part of Unity nuget package.

|  |
| --- |
| public static class UnityConfig  {  public static void RegisterTypes(IUnityContainer container)  {    var storage = CloudStorageAccount.Parse(ConfigurationManager.AppSettings["storagekey"]);  var log = new LoggerConfiguration().ReadFrom.KeyValuePairs(new KeyValuePair<string, string>[]  {  new KeyValuePair<string, string>("OutputTemplate" ,"{Timestamp:yyyy-MM-dd HH:mm:ss zzz} [{Level}] {RequestId}-{SourceContext}: {Message}{NewLine}{Exception} {ActivityId}")  })  .WriteTo.AzureTableStorageWithProperties(storage, storageTableName: "LogEventTable", propertyColumns: new[] { "ActivityId" }).Enrich.With(new SeriLogEncricher())  .MinimumLevel.Warning()  .CreateLogger();  container.RegisterInstance<ILogger>(log);  }  }  //Later ILogger can be referred within controllers for logging.  public class LoggerController : Controller  {  private readonly ILogger \_logger;  public LoggerController(ILogger logger)  {  this.\_logger = logger;  }  [HttpPost]  public ActionResult Log([Form]int logType)  {  \_logger.Warning("testing");  }  } |

Now logged contents can be found within the "LogEventTable" table of storage account mentioned in the connection string.



The logger will also log the activity id which would be useful to co-related trace message with log entries & log information of specific user\session can be easily traced.

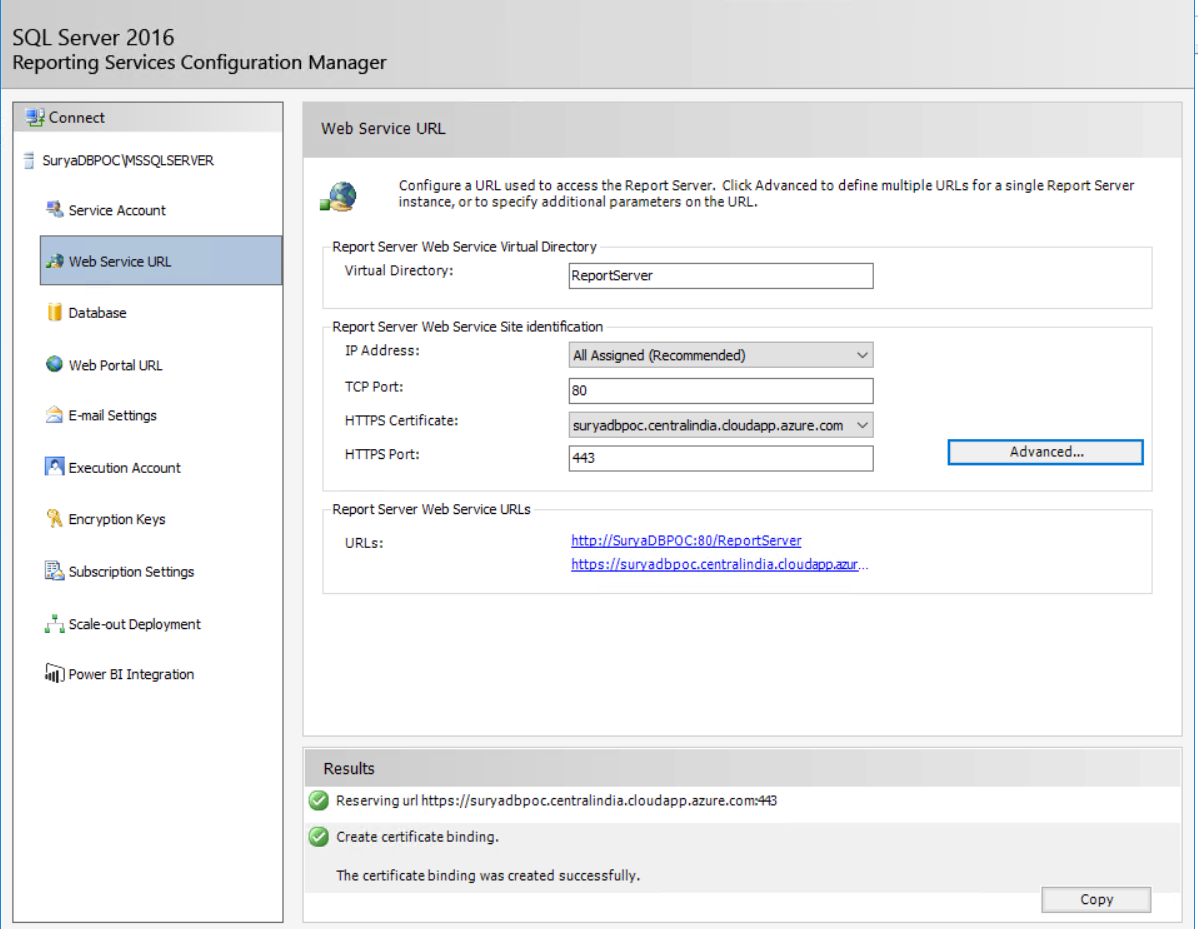
### References

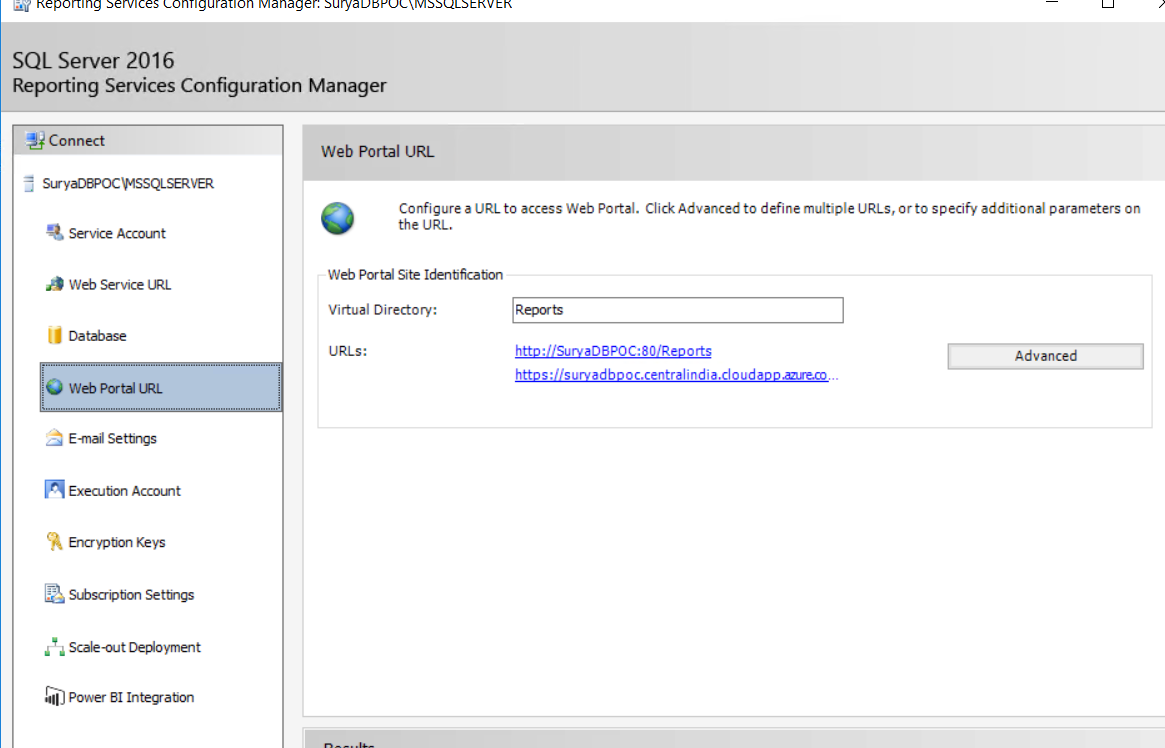
Serilog -https://serilog.net/

# Reporting – Using SSRS

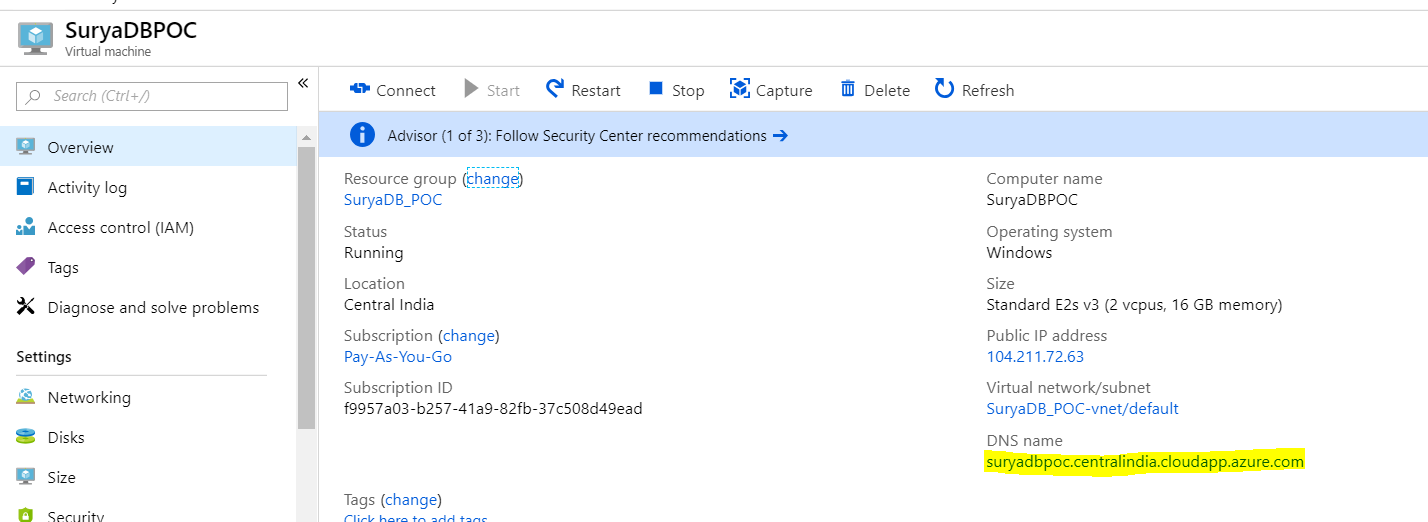
## Configure reporting service

For configuring reporting service to be available over https, bind the reporting service & portal services with the certificate.





|  |  |
| --- | --- |
|  | Make sure https certificate is issued in the name of server hosting the report service. In our case its VM (suryadbpoc.centralindia.cloudapp.azure.com).  DNS name of VM (DB server) for creating certificate can be obtained from VM’s overview page in Azure portal(see below snap).  **Note:** Self-signed certificate can be created using following PowerShell script  New-SelfSignedCertificate -DnsName suryadbpoc.centralindia.cloudapp.azure.com -CertStoreLocation cert:Localmachine\My |

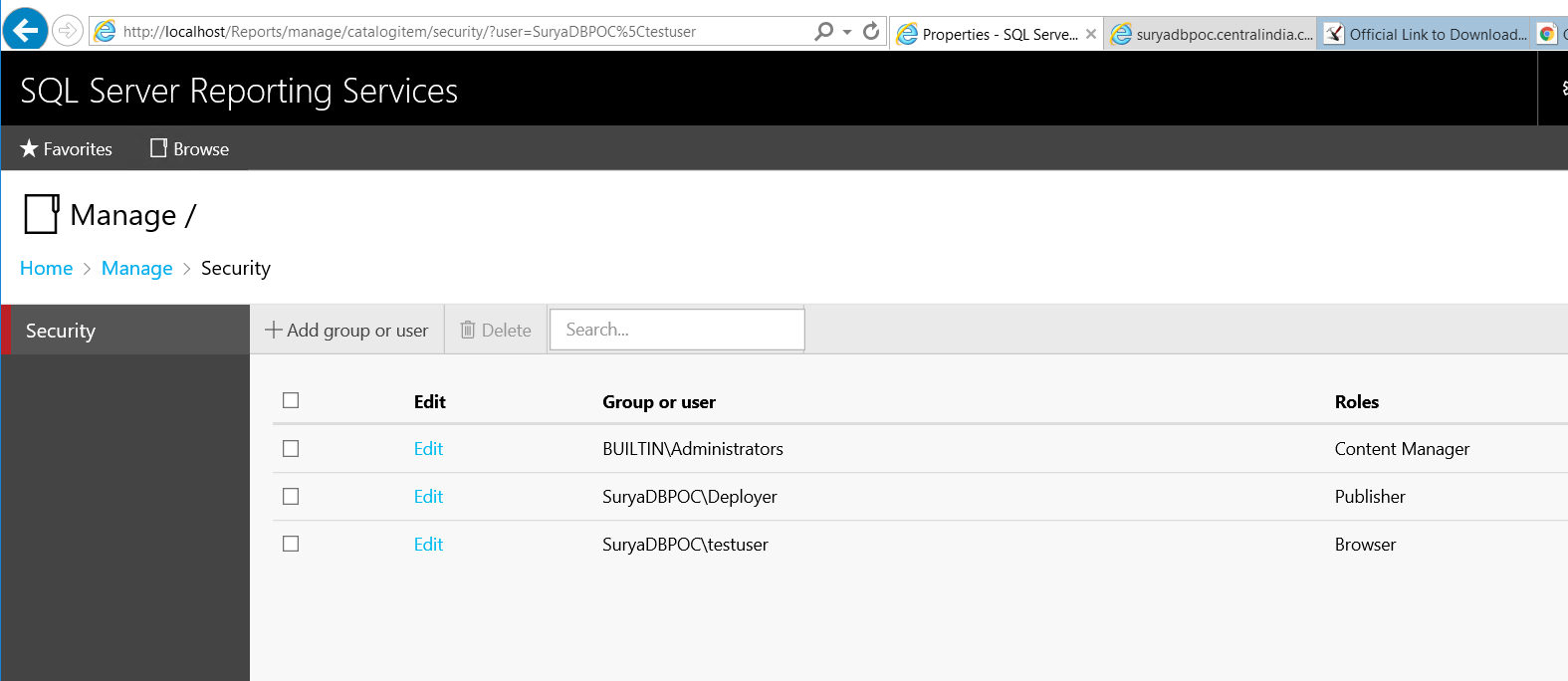
-

Post configuring reporting services, Make sure reporting service can be accessed over https using report service web portal URL (Ex: <https://suryadbpoc.centralindia.cloudapp.azure.com/Reports>).

### Access provisioning

Add users & provide required access appropriately. Use separate user accounts for publishing and viewing reports.

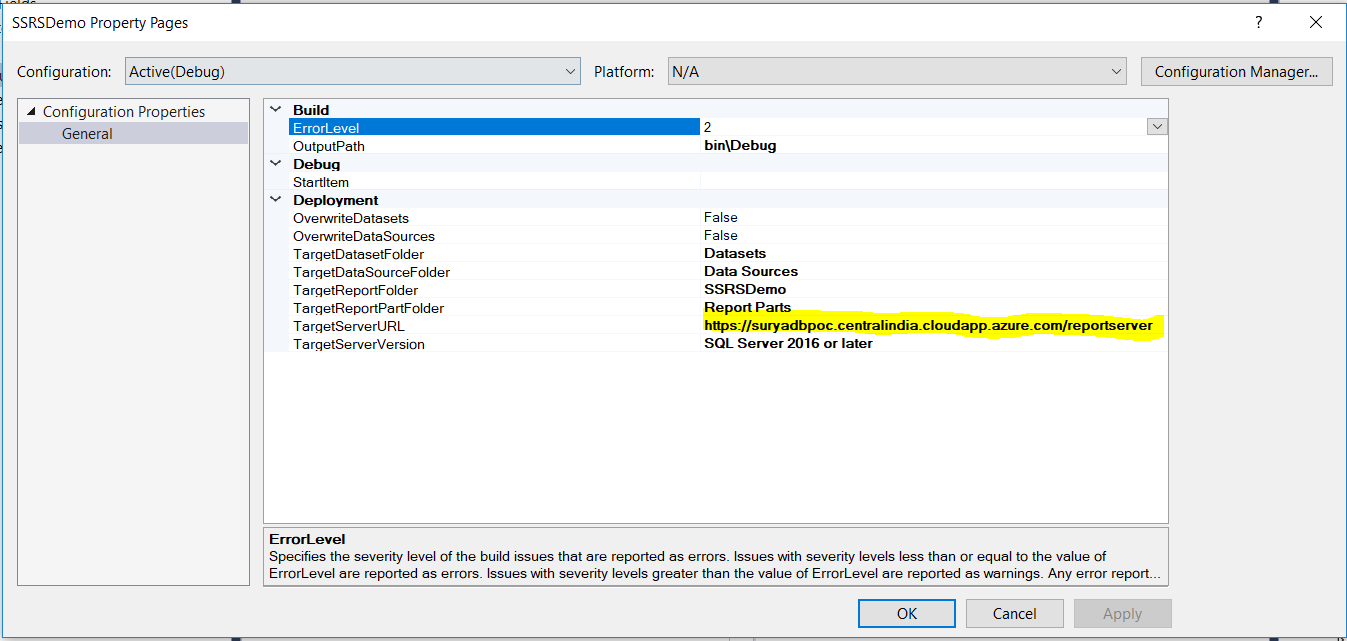
Publisher account would be used by developers for publishing the reports & user with browser role would be used by the web app to render the reports.



## Publish reports

Once the reporting service portal is up & running, reports can easily published by visual studio.

1. Edit the project property & specify reporting server url.



1. Initiate deployment from Visual studio by selecting Reporting service project & using Build 🡪 Deploy XXXX (Project name) menu option.

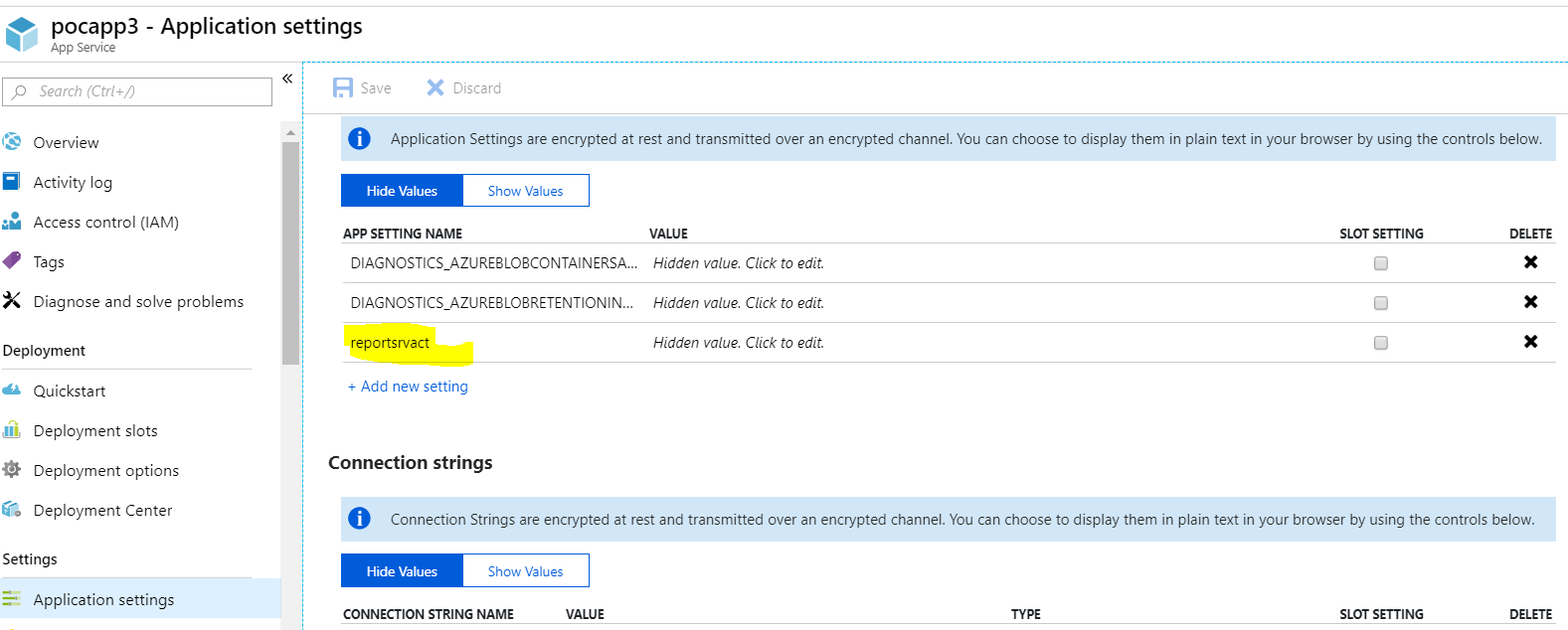
## Render report in MVC app

Rdl reports can easily rendered in MVC using ReportViewerForMvc Nuget package. Once nuget package is added to web application, following code would render the report.

|  |
| --- |
| **MVC controller:**  var reportViewer = new ReportViewer()  {  ProcessingMode = ProcessingMode.Remote,  SizeToReportContent = true,  Width = Unit.Percentage(100),  Height = Unit.Percentage(100),  };  reportViewer.ServerReport.ReportServerCredentials=new ReportServerCredentials();  reportViewer.ServerReport.ReportPath = "/SSRSDemo/Sales Orders";  reportViewer.ServerReport.ReportServerUrl = new Uri("https://suryadbpoc.centralindia.cloudapp.azure.com/ReportServer/");  ViewBag.ReportViewer = reportViewer;  **View:**  @Html.ReportViewer(  ViewBag.ReportViewer as Microsoft.Reporting.WebForms.ReportViewer,  new { scrolling = "no", width="100%" }) |

Here, ReportServerCredentials implements Microsoft.Reporting.WebForms.IReportServerCredentials interface to provide credential details required for rendering the reports.

Credentials can be securely saved using appsetting page of Azure app service blade.



Note: Appsetting values can saved both in Azure & web.config. When app is running locally these settings would be picked from web.config & post deploying onto Azure these settings would be picked from Azure.

Appsettings can be read in Azure using usual ConfigurationManager.AppSettings property. Refer the sample application for more detail.

### Reference:

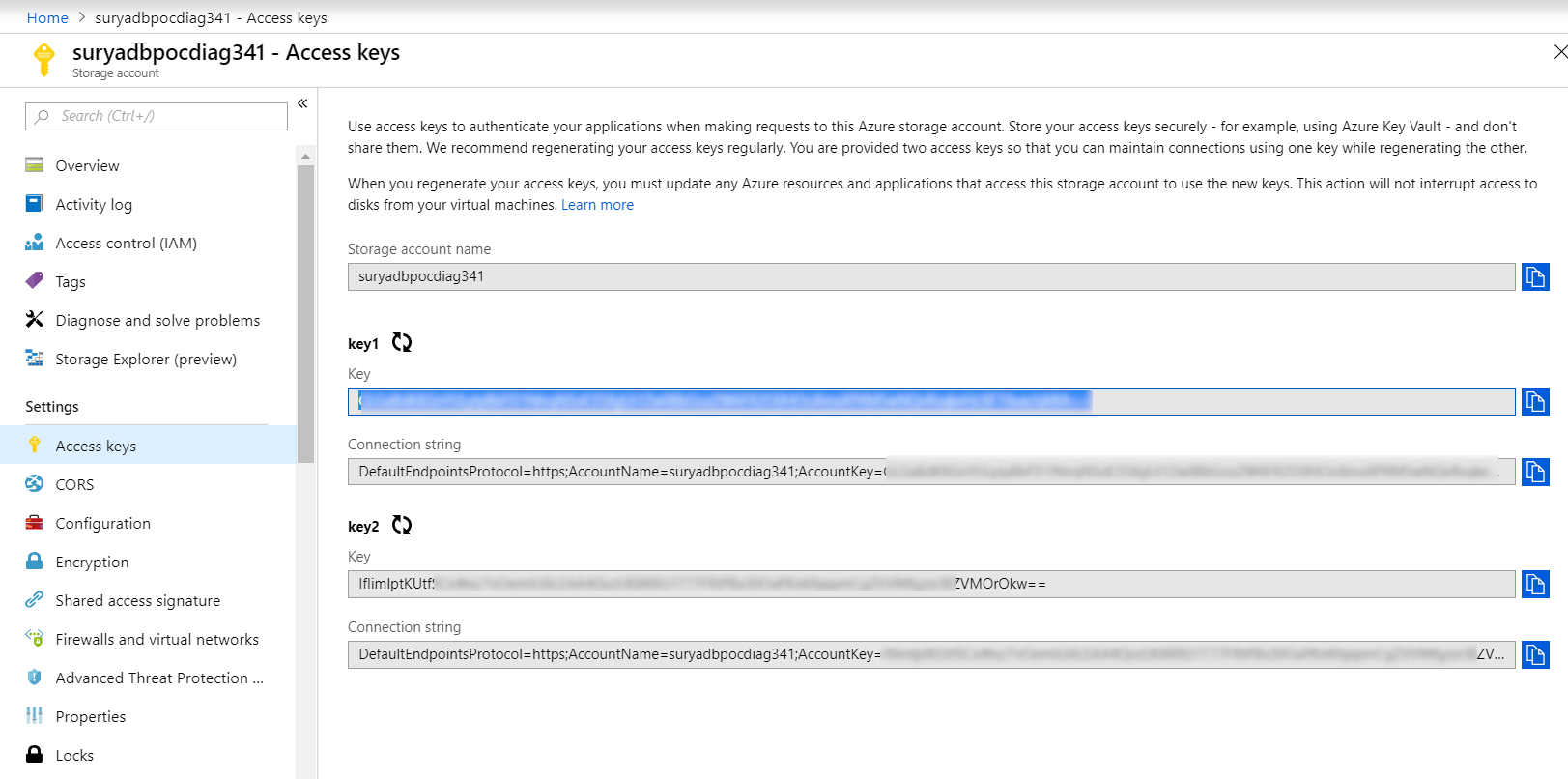
<https://github.com/armanio123/ReportViewerForMvc>

<https://docs.microsoft.com/en-us/sql/reporting-services/security/configure-ssl-connections-on-a-native-mode-report-server?view=sql-server-2017>

# Azure Storage- file management

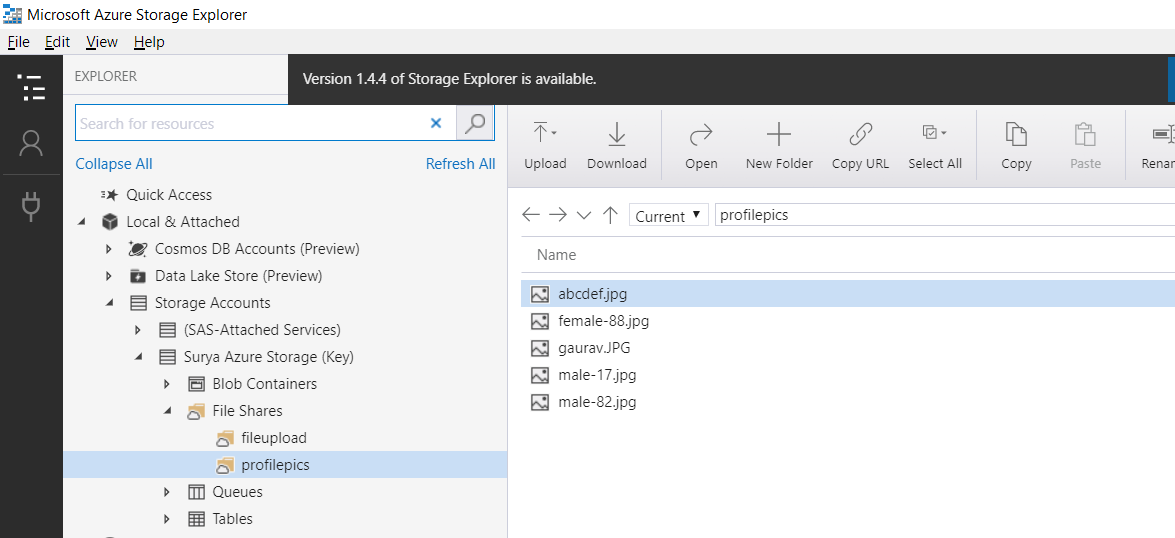
For managing files stored in Azure Storage, Microsoft Azure Storage explorer can be used. For connecting to Azure storage, we can use following options.

|  |  |
| --- | --- |
|  | Get the account name & access key from storage account blade in azure portal as shown in below screen shot. |



|  |  |
| --- | --- |
|  | Here, suryadbpocdiag341 is the account name Key1 & Key2 are the key required for connecting to Azure storage. |

Click on Next & continue with subsequent wizard window & complete the connection. Once connected, Microsoft Azure Storage explorer would list available blob, Queues & Tables. Using this tool we can perform common file management activities like add, delete etc…



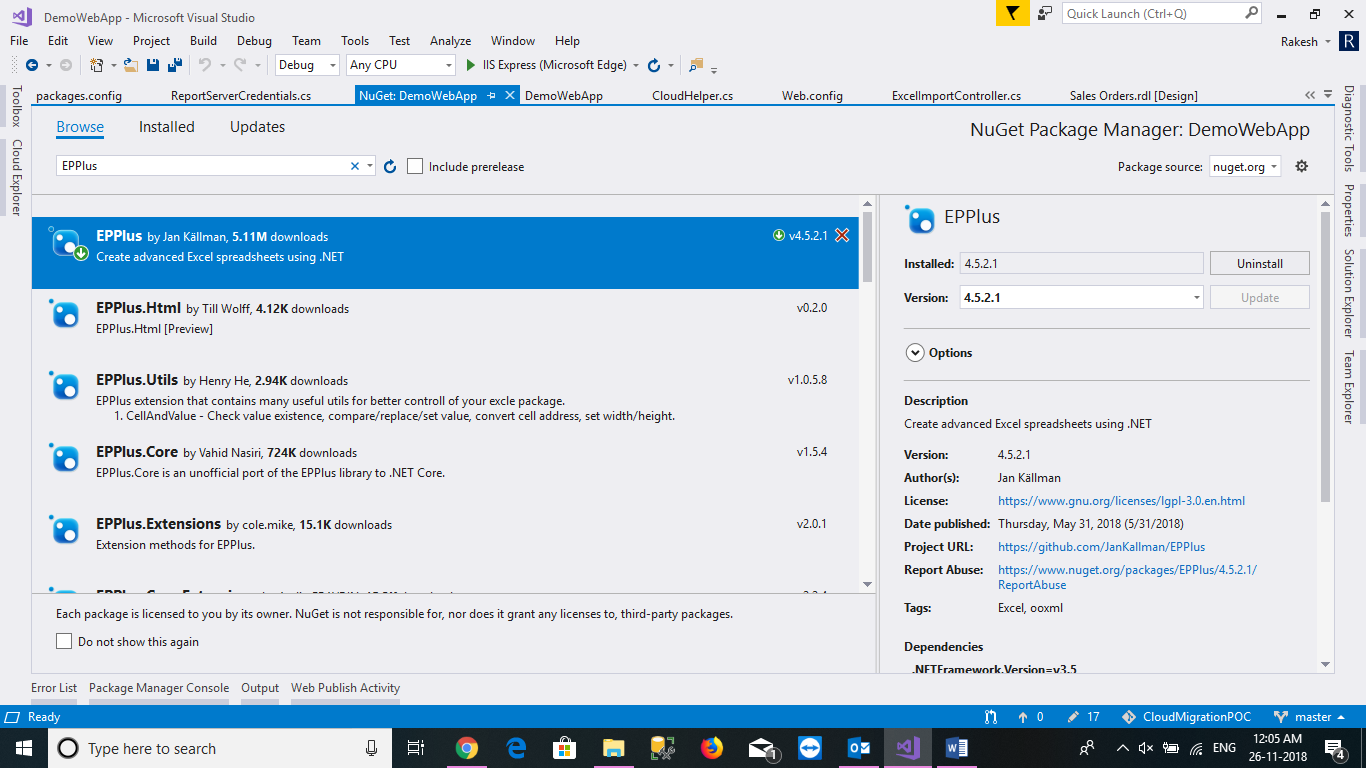
### Reference

Microsoft Azure Storage explorer: <https://azure.microsoft.com/en-in/features/storage-explorer/>

# Requirement – Importing excel file

The existing Web applications is using com component to read the excel and translate it to the entity. But Microsoft don’t recommend the com component uses in the application, hence Azure app service don’t support the com components. To overcome this problem, we can use some “OpenXml” library to read the data from excel spreadsheet.

We have used EPPlus OpenXml library to read from the excel. Below screenshot for the nuget package which needs to be installed for this.



We have created one utility which will take excel file as input and return an entity out of it.

Our sample app can be referred for the code.

|  |
| --- |
| public static class ExcelImporter  {  public static IList<T> ReadToList<T>(string path,int fromRow,int fromColumn, int toColumn = 0) where T: class  {  using (var excelpack = new OfficeOpenXml.ExcelPackage())  {  List<T> getList = new List<T>();  using (var stream = File.OpenRead(path))  {  excelpack.Load(stream);  }  var ws = excelpack.Workbook.Worksheets.First();  toColumn = toColumn == 0 ? typeof(T).GetProperties().Count() : toColumn;  for(var rownum= fromRow; rownum<=ws.Dimension.End.Row;rownum++)  {  T objT = Activator.CreateInstance<T>();  Type myType = typeof(T);  PropertyInfo[] myprop = myType.GetProperties();  var wsRow = ws.Cells[rownum, fromColumn, rownum, toColumn];  for(int i=0;i<myprop.Count();i++)  {  myprop[i].SetValue(objT, Convert.ChangeType(wsRow[rownum, fromColumn + i].Value, myprop[i].PropertyType));  }  getList.Add(objT);  }  return getList;  }  }  } |

How to consume the above method.

|  |
| --- |
| IList<SaleOrder> saleorderlist = ExcelImporter.ReadToList<SaleOrder>(excelPath, 2, 1); |

Few Important Assumption which needs to take care when using this method:

1. The excel sheet column sequence and the entity property sequence should be same.

For Ex : SalesOrder Class property sequence is same as SaleOrder Excel column sequence

|  |
| --- |
| public class SaleOrder  {  public string SaleToBuyer { get; set; }  public DateTime OrderDate { get; set; }  public string OrderNo { get; set; }  public DateTime DueDate { get; set; }  public string Product { get; set; }  public string BuyerUpcCode { get; set; }  public int Quantity { get; set; }  public double Rate { get; set; }  } |

1. OpenXml support .xlsx extension and not the .xls extension, hence where ever using .xls , please convert the template into .xlsx.
2. The method takes “fromrow” as one of the parameters which means from which row it will start reading the data, generally the first row is for the title and the real data starts from 2nd row, so pass 2 as input or from which row it starts pass that. From column is also to specify from which column we should start reading data. One more parameter is there which is optional and it is toColum, if we want to restrict the reading till any specific column then pass that otherwise it will take all the column where data is there in account and read it. Normal scenarios we don’t need to pass the value for that parameter.