

Developing Microsoft Azure Infrastructure Solutions

Content

Module 1: Overview of the Microsoft Azure Platform	5
Objective	5
Prerequisites	5
Exercise 1: Navigate in the Azure portal	5
Task 1: Access to the portal	5
Task 2: Work with Blades and Journey	5
Task 3: See favourites resources.....	5
Task 4: Work with dashboard	5
Module 2: Building Application Infrastructure in Azure	6
Objectives	6
Prerequisites	6
Exercise 1: Work with Virtual Network.....	6
Task 1: Create a Virtual Network	6
Task 2: Explore Virtual Network.....	6
Exercise 2: Work with Virtual Machine.....	6
Task 1: Create a Virtual Machine	6
Task 2: Add a Disk	7
Task 3: Access to the created Virtual Machine	8
Task 4: Configure new disk	8
Module 3: Hosting Web Applications on the Azure Platform.....	9
Objectives	9
Prerequisites	9
Exercise 1: Work with Web Application.....	9
Task 1: Create a Web Application	9
Task 2: Configure Autoscaling based on a metric	9
Task 3: Configure Autoscaling based on time	10
Task 4: Monitor instances running.....	10
Exercise 2: Create a Web Application in VS	10
Task 1: Create as Web application	10
Task 2: Create Home page	11

Task 3: Test the new Web Application.....	12
Exercise 3: Publish Web Application.....	12
Task 1: Add Dev slots	12
Task 2: Add Staging slot	12
Task 3: Get publish settings	12
Task 4: Publish web Application.....	13
Task 5: Swap publish from Development to Staging	13
Module 4: Storing SQL Data in Azure.....	14
Objectives	14
Prerequisites	14
Exercise 1: Work with SQL Database	14
Task 1: Create a SQL Database.....	14
Task 2: Explorer SQL Database.....	14
Exercise 2: Integrate SQL Database to a Web Application	14
Task 1: Connect SQL DB to Web Application	14
Task 2: Create Controller and Views	15
Task 3: Test the Web Application	16
Task 4: Publish Web Application.....	16
Module 5: Storing Unstructured Data in Azure	17
Objectives	17
Prerequisites	17
Exercise 1: Work with Storage Account and Table	17
Task 1: Create a Storage Account	17
Task 2: Create a Table	17
Exercise 2: Integrate Table to a Web Application.....	18
Task 1: Connect Storage Account to Web Application	18
Task 2: Create Controller and Views	18
Task 3: Test the Web Application	21
Task 4: Publish Web Application.....	21
Exercise 3: Work with Cosmos DB	21
Task 1: Create a Cosmos DB using SQL API	21
Task 2: Change consistency to Cosmos DB	22
Task 3: Replicate Cosmos DB	22

Task 4: Add a DB and Collection to Cosmos DB	22
Exercise 4: Integrate Cosmos DB to a Web Application	23
Task 1: Connect Cosmos DB to Web Application	23
Task 2: Create Controller and Views	23
Task 3: Create Repository	24
Task 4: Test the Web Application	26
Task 5: Publish Web Application	26
Module 6: Storing and Consuming Files from Azure Storage	27
Objectives	27
Prerequisites	27
Exercise 1: Work with Blob	27
Task 1: Create a Container	27
Task 2: Create a Blob	27
Exercise 2: Integrate Blob to a Web Application	27
Task 1: Create Controller and Views	27
Task 2: Add Security	30
Task 3: Test the Web Application	30
Task 4: Publish Web Application	30
Task 5: Review Blobs in Azure portal	31
Exercise 3: Work with File	31
Task 1: Create a File	31
Task 2: Map the File share as a drive to the Virtual Machine (previously created)	31
Module 7: Designing a Communication Strategy by Using Queues and Service Bus	31
Objectives	31
Prerequisites	31
Exercise 1: Work with Storage Account Queue	31
Task 1: Create a Queue	31
Task 2: Add a Message	32
Exercise 2: Integrate Storage Account Queue to a Web Application	32
Task 1: Connect Queue to Web Application	32
Task 2: Create Controller and Views	32
Task 3: Test the Web Application	36
Task 4: Publish Web Application	36

Exercise 3: Work with Service Bus Queue	36
Task 1: Create a Service Bus namespace	36
Task 2: Create a Queue	37
Exercise 4: Integrate Service Bus Queue to a Web Application.....	37
Task 1: Connect Queue to Web Application	37
Task 2: Create Controller and Views	38
Task 3: Test the Web Application	41
Task 4: Publish Web Application.....	41
Module 8: Securing Azure Web Applications	41
Objectives	41
Prerequisites	41
Exercise 1: Secure Web Application with Active Directory.....	41
Task 1: Connect Web Application with Azure Active Directory.....	41
Task 2: Test the Web Application	42
References	42

Module 1: Overview of the Microsoft Azure Platform

Objective

- Know the azure portal.

Prerequisites

- Have a valid Microsoft account.
- Have an Azure subscription.

Exercise 1: Navigate in the Azure portal

Task 1: Access to the portal

- Go to the URL <https://portal.azure.com/>.
- Sign in with your Microsoft account.

Task 2: Work with Blades and Journey

- Click the + icon (Create a resource) at the top-left corner.
- In the Azure Marketplace, click on <Compute>.
- In the Featured resources, click on Windows Server 2016 Datacenter.
 - A Journey is opened (Home -> New -> Create virtual machine -> Basics), the current Blade is <Basics>.

Task 3: See favourites resources

- Click on All services from the left menu.
- Click on the start icon from the Recent resource, it would be added to the Favourites.
- Click again on the start icon from the Recent resource, it would be removed from the Favourites and the start will be grey.

Task 4: Work with dashboard

- Edit Dashboard.
 - Go to home page in Azure portal.
 - Click <Edit> from the top menu.
 - From the <Tile Gallery>, add a new tile.
 - Click on <Add> link beside the <Resource groups> tile.
 - Move the new Tile.
 - Drag and drop the new <Resource groups> tile to the left bottom side of the dashboard.
 - Change the new Tile size.
 - Click on the <...> button from the top right corner of the tile.
 - Select 4 x 4.
 - Click on <Done customizing> to save changes

Module 2: Building Application Infrastructure in Azure

Objectives

- Create a VN.
- Create a VM using the previously created VN.
- Manage new disks.
- Have a VM with VS to develop an application for the next modules.

Prerequisites

- Have an Azure subscription.

Exercise 1: Work with Virtual Network

Task 1: Create a Virtual Network

- Click on <Create a Resource>.
- Click on Networking -> Virtual Network.
- Add the next fields:
 - Add the Name.
 - Change the Address space¹.
 - Select a Subscription.
 - In Resource Group, select <Create new>.
 - Write the Resource Group name that you will be using for all the demos.
 - Select the closest Location.
 - In the Subnet section.
 - Change the Name¹.
 - Change the Address range¹.
 - Leave the rest of the fields with their default value.
 - Click on <Create>.

Task 2: Explore Virtual Network

- When the Web Application is created, from notifications click on <Go to resource>.
- Close the Notifications.
- See details in the <Overview> menu.
 - In the <Essentials> section you will see the specifications of the VN.
 - After the <Essential> there is the list of the Connected devices.

Exercise 2: Work with Virtual Machine

Task 1: Create a Virtual Machine

- Click on <Create a Resource>.
- In the <Search the Marketplace>, type: Visual Studio Community 2017 on Windows Server 2016 (x64).
- From the results, click on <Visual Studio Community 2017 on Windows Server 2016 (x64)>.
- In the new blade, click on <Create> (make sure, the model selected is <Resource Manager>).
- Complete the <Basics> section:
 - For the <Project details>.
 - Select a Subscription.

- Select the Resource Group created previously (Module 2, Exercise 1, Task 1).
- For the <Instance details>.
 - Add the VM Name.
 - Select the closest Region.
 - In Size, click on the <Change size> link.
 - Select a F1 size VM (You can filter by adding the VM size in the <Search by VM size...>).
 - Click on <Select>.
- For the <Administrator account>.
 - Add the Username².
 - Add the Password².
 - Add the Confirm Password².
- For the <Inbound port rules>.
 - In the Public inbound ports, select <Allow selected ports>.
 - In the Select inbound ports, Select <RDP (3389)>.
- Leave the rest of the fields with their default value.
- Click on <Next: Disks >.
- Complete the <Disks> section:
 - For the <Disk options>.
 - In the OS disk type, select <Standard HDD>.
 - Leave Use manage disks as No.
 - Click on <Next: Networking>
- Complete the <Networking> section:
 - For the <Network interface>.
 - In Virtual Network, select the VN created in the Exercise 1, Task 1 of this module.
 - In the Subnet, select the subnet created in the Exercise 1, Task 1 of this module.
 - Make sure the RDP port is selected, in the Selected inbound ports.
 - Leave the rest of the fields with their default value.
 - Click on <Review + create>.
- Complete the <Review + create> section:
 - Review the details of the new VM.
 - Click on <Create>.
- A new blade will be opened (Deployment), where you can see the progress also the list of resources that are being created.

Task 2: Add a Disk

- When the VM is created, click on the <Go to resource> button at the top of the Deployment blade.
- In the Virtual machine blade, click on Settings -> Disks from the left menu.
- Click on <Add data disk> button.
- From the new Data disks record:
 - In the Name field, click the arrow to show the options.
 - Click on <Create disk> link.

- A new blade will be opened.
- Add the next fields:
 - Add the Name of the disk.
 - In Resource Group, select <Use existing>.
 - Select the Resource Group created in Module 2, Exercise 1, Task 1.
 - Select Standard HDD in Account type.
 - Select 128 GB in Size¹.
 - Leave the rest of the fields with their default value.
 - Click on <Create>.
- Back to the Disks blade, click on <Save> from the top menu.

Task 3: Access to the created Virtual Machine

- Click on Overview from the left menu.
- Click on <Connect> from the top menu.
- In the new blade, click on <Download RDP File>
- When asked, click on <Open>.
- In the new Remote connection dialog box, click on <Connect>.
- Provide the username and password used when the VM was created.
- Click on <Accept>.
- If asked, click on <Yes> for <The Identity of the remote computer cannot be verified> message.
- After access to the VM, the Server Manager will start (in about 30 seconds).
 - If not, open the Server Manager from the Start screen.
- Click on <Local Server> from the left menu.
- from the properties, click on link beside the <IE Enhanced Security Configuration>
 - Set Administrators to <Off>.
 - Set Users to <Off>.
 - Click on <OK>.

Task 4: Configure new disk

- In the Server Manager, click on <File and Storage Services> from the left menu.
- In the new panel, click on <Disks> from the left menu.
- Right click on the section of Disks.
- Click on <New Volume...>.
- Complete the New Volume Wizard.
 - In the Before you begin, click <Next>.
 - In the Server and Disk, select the Disk 2 from the Disk section.
 - Click on <Next>.
 - Confirm the Offline or Uninitialized Disk message.
 - In the Size, select the size for the disk¹.
 - Click on <Next>.
 - In the Drive Letter or Folder, select the Drive Letter¹ that will be assigned.
 - Click on <Next>.
 - In the File System Settings, change the Volume label¹.
 - Click on <Next>.

- In the Confirmation, click on <Create>.
 - When it is finished, click on <Close>.
- Open a File explorer and confirm the new unit is created.

Module 3: Hosting Web Applications on the Azure Platform

Objectives

- Create a web Application from the Azure portal.
- Manage auto scaling.
- Create a web Application using VS.
- Publish the application created in VS to the application created in Azure.
- Manage slots.

Prerequisites

- Complete exercises from Module 2.

Exercise 1: Work with Web Application

Task 1: Create a Web Application

- Go to Azure portal.
- Click on <Create a Resource>.
- Select Web -> Web App.
- Complete:
 - Add the App Name.
 - Select a Subscription.
 - In Resource Group, select <Use existing>.
 - Select the Resource Group created in Module 2, Exercise 1, Task 1.
 - Change App Service plan.
 - Click on <App Service plan/Location>.
 - Click on <Create new>.
 - Add the App service plan name.
 - Select the closest Location.
 - Click on Pricing tier.
 - Select Production -> S1 tier.
 - Click on <Apply>.
 - In the New App Service Plan, click on <OK>.
 - Leave the rest of the fields with their default value.
 - Click on <Create>.
- When the Web Application is created, from notifications click on <Go to resource>.
- Close the Notifications.
- In the Overview, click on the URL.
- See the content of the development environment.

Task 2: Configure Autoscaling based on a metric

- Back to Azure portal, click on Settings -> Scale out from the left menu.
- In the new blade, click on <Enable autoscale>.

- Complete Autoscale:
 - Add the Autoscale setting name.
 - Select the Resource Group created in Module 2, Exercise 1, Task 1.
- Complete Default section:
 - Select <Scale based on a metric>.
 - To **scale out**.
 - Click on <Add a rule> link.
 - Complete Criteria from the <Scale rule>:
 - Leave the default values to autoscale out based on CPU Percentage.
 - Click on <Add>.
 - To **scale in**.
 - Click on <Add a rule> link.
 - Complete Criteria section from the <Scale rule>:
 - Make sure the same Metric is selected than the previous rule.
 - Select <Less than> in the Operator.
 - Select a smaller Threshold than the scale out rule (50).
 - Complete Action section:
 - Select < Decrease count by > in the Operation.
 - Click on <Add>.
 - Change Maximum instance to 5 in the Instance limits.
 - Click on <Save>.

Task 3: Configure Autoscaling based on time

- Click on <Add a scale condition> at the bottom of the Scale out (App Service plan) blade.
- Complete the new <Auto created scale condition>:
 - In the Scale mode, select <Scale to a specific instance count>.
 - In the Schedule, select <Specify start/end dates>.
 - In the Timezone, select your timezone.
 - In the Start date, select the current date, and in the time the current time plus 5 minutes.
 - In the End date, select the current date.
 - Click on <Save>.

Task 4: Monitor instances running

- Click on Monitoring -> Process explorer, from the left menu.
- When there it is the time you schedule, click on <Refresh>.
- Verify there is a new instance running.

Exercise 2: Create a Web Application in VS

Task 1: Create as Web application

- In the VM, open Visual Studio 2017 (VS).
- On the VS Welcome, click on <Sig in>.
- Provide the Microsoft account user and password.
- Once VS is opened, go to File -> New -> Project...
- On the New Project dialog box, complete:

- Select Installed -> Templates -> Visual C# -> Web, from the left menu.
 - Select ASP.NET Web Application (.NET Framework).
 - Select <.NET Framework 4.6> from the drop-down list on the top.
 - Add the Project name.
 - Change the location.
 - Click on <Browse...>.
 - Navigate to select the new Unit created.
 - Add the Solution name¹.
 - Click on <OK>.
- On the New ASP.NET Web Application, complete:
 - Select the Empty Template.
 - Check <MVC> in the <Add folders and core references for:>.
 - Click on <OK>.

Task 2: Create Home page

- Add Controller.
 - Right click on Controllers folder from the Solution Explorer.
 - Click on Add -> Controller.
 - Select <MVC 5 Controller - Empty>.
 - Click on <Add>.
 - Add the controller name (HomeController).
 - Click on <Add>.
- Add View.
 - Right click on the Views -> Home folder.
 - Click on Add -> View...
 - Add the View name (Index).
 - Select Empty (without model) Template.
 - Check <Use a layout page:>.
 - Click <Add>.
 - Replace the new file created with the next lines of code:

```
@{
    ViewBag.Title = "Azure";
}
<div class="jumbotron">
    <h1>Azure development</h1>
</div>
```

- Save the Index view.
- Add link to home.
 - Open the Views -> Shared -> _Layout.cshtml partial view.
 - Add the next line, after the <ul class="nav navbar-nav">.

```
<li>@Html.ActionLink("Home", "Index", "Home")</li>
```

- Change the title of the application, replace the next lines.
 - Line 6

```
<title>@ViewBag.Title - Azure Application</title>
```

- Line 20

```
@Html.ActionLink("Azure development", "Index", "Home", new { area = "" }, new { @class = "navbar-brand" })
```

- Line 34

```
<p>&copy; @DateTime.Now.Year - Azure Application</p>
```

- Save the _layout partial view.

Task 3: Test the new Web Application

- Right click on the Solution from the Solution Explorer.
- Click on <Build Solution>.
- Once the Solution is built, click on Debug-> Start Debugging menu or press or F5.
- Check the Web Application runs without problem.
- Close the internet explorer.

Exercise 3: Publish Web Application

Task 1: Add Dev slots

- Go to Azure portal.
- Open the web application created previously.
- Click on Deployment -> Deployment slots from the left menu.
- Click on <Add Slot> from the top menu.
- Complete the development Slot:
 - Slot Name: Dev.
 - Change the Configuration Source¹.
 - Click on <OK>.
- Click on the new Slot.
- In the Overview (make sure the blade is pointing to the development environment), click on the URL.
- See the content of the development environment.

Task 2: Add Staging slot

- Click on Deployment -> Deployment slots from the left menu.
- Click on <Add Slot> from the top menu.
- Complete the staging Slot:
 - Slot Name: Staging.
 - Change the Configuration Source¹.
 - Click on <OK>.
- Click on the new Slot.
- In the Overview (make sure the blade is pointing to the staging environment), click on the URL.
- See the content of the staging environment.

Task 3: Get publish settings

- Open the Azure portal from your VM.
- Open to the App Service you created before and select the dev slot (make sure you are in the dev slot).
 - Go to Deployment -> Deployment slots.
 - Click on the dev slot (youWebAppName - dev).

- Click on <Get publish profile> from the top menu of the Overview.
- Click on <Save as> and save the publish Settings file to the new drive created in the VM.

Task 4: Publish web Application

- In VS, in the Object Explorer, right click on the Project Name.
- Click on <Publish...>.
- From the Publish window, select <Import profile> (If not visible, click on the right arrow).
- Click on <Publish>.
- Search and select the publish Settings file you saved from the Web Application.
 - Click on <Open>.
- A new page will be opened.
- Make sure the URL is the same of the development environment from the web Application.

Task 5: Swap publish from Development to Staging

- Go to the Azure portal.
- Open to the App Service you created before.
- Swap from Dev to Staging.
 - Go to Deployment -> Deployment slots from the left menu.
 - Click on <Swap> from the top menu.
 - Complete:
 - Select dev in the Source.
 - Select Staging in the Destination.
 - Click on <OK>.
 - Once the process finishes, select the Staging slot, if already selected, go to Overview from the left menu.
 - On the Overview, click on the URL to see the new version.
 - On the Deployment -> Deployment slots.
 - Select the Dev slot.
 - On the Overview, click on the URL to see now it has the previous version.
 - Confirm the versions from the two slots have changed.
- Swap from Staging to Production.
 - Go to Deployment -> Deployment slots from the left menu.
 - Click on <Swap> from the top menu.
 - Complete:
 - Select Staging in the Source.
 - Select Production in the Destination.
 - Click on <OK>.
 - Once the process finishes, select the Production slot, if already selected, go to Overview from the left menu.
 - On the Overview, click on the URL to see the new version.
 - On the Deployment -> Deployment slots.
 - Select the Staging slot.
 - On the Overview, click on the URL to see now it has the previous version.
 - Confirm the versions from the two slots have changed.

Module 4: Storing SQL Data in Azure

Objectives

- Create a SQL Database.
- Integrate DB to Web Application.
- Use Web Application Settings.

Prerequisites

- Complete Exercises from Module 3.

Exercise 1: Work with SQL Database

Task 1: Create a SQL Database

- Go to Azure Portal.
- Click on <Create a Resource>.
- Select Databases -> SQL Database.
- Complete:
 - Add the Database name.
 - Select a Subscription.
 - In Resource Group, select <Use existing>.
 - Select the Resource Group created in Module 2, Exercise 1, Task 1.
 - Select <Sample (AdventureWorksLT)> in the source.
 - Click on <Configure required settings> in Server.
 - If not opened, click on <Create a new server>.
 - Add the Server name
 - Add the Server admin login³.
 - Add the Password³.
 - Confirm the Password³.
 - Select the closest Location.
 - Leave the rest of the fields with their default value.
 - Click on <Select>
 - Click on <Configure required settings> in Pricing tier.
 - Select <Basic>.
 - Click on <Apply>.
 - Leave the rest of the fields with their default value.
 - Click on <Create>.

Task 2: Explorer SQL Database

- When the SQL Database is created, from notifications click on <Go to resource>.
- Close the Notifications.
- On the Overview menu, see the details.
- Save the Server name in a notepad for further use.

Exercise 2: Integrate SQL Database to a Web Application

Task 1: Connect SQL DB to Web Application

- Open in VS the Web Application created in Module 3, Exercise 2.

- Right click on `[Solution_Name]` -> `[Project_Name]` -> Models folder from the Object Explorer.
 - Where `[Solution_Name]` is the name of your solution and `[Project_Name]` is the name of your project.
- Click on Add -> New Item...
- Select Installed -> Visual C# -> Data, from the left menu in the <Add New Item> wizard.
- Select <ADO.NET Entity Data Model>.
- Add the Name of the model (AdventureSQL).
- Click on <Add>.
- On the Entity Data Model Wizard, complete:
 - Select <EF Designer from database>.
 - Click on <Next>.
 - Click on <New Connection...>.
 - On Data source, select <Microsoft SQL Server>.
 - Click on <Continue>.
 - On Server name, add the previously saved Server name (name of the server when creating the SQL database resource).
 - On Authentication select <SQL Server Authentication>.
 - Add the User name you used to create the SQL Server.
 - Add the Password you used to create the SQL Server.
 - On Select or enter a database name, select the database created.
 - Click on <OK>.
 - Select <Yes, include the sensitive data int the connection string.> option from the Entity Data Model Wizard.
 - Change the Connection string name¹.
 - Click <Next>.
 - Select the 6.x Entity Framework.
 - Click <Next>.
 - In the Choose Your Database Objects and Settings window, check all the tables.
 - Click on <Finish>.
 - If the Security Warning is displayed, click on <OK>.
 - Rebuild the solution to confirm it is compiling.
 - Right click on the Solution from the Solution Explorer.
 - Click on <Build Solution>.

Task 2: Create Controller and Views

- Right click on <Controllers> from the Object Explorer.
- Click on Add -> Controller.
 - Select <MVC 5 Controller with views, using Entity Framework>.
 - Click on <Add>.
- On Add Controller:
 - Select in Model class the table <Product (`[ApplicationName].Models`)>.
 - Where `[ApplicationName]` is the namespace, which is the same of the application name.
 - Select in Data context class the connection created to the SQL DB.

- Leave the rest of the fields with their default value.
- Click on <Add>.
- By default, the Controller's name is *productsController*, and there will be created a folder *Products* in the Views folder, with 5 views (Create, Delete, Details, Edit, Index).
- Create a link to the new view.
 - Open the Views -> Home -> Index.cshtml view.
 - Add the next line after `<h1>Azure development</h1>`.

```
<h2>@Html.ActionLink("SQL Products", "Index", "Products")</h2>
```

 - Save the Index.cshtml file.
- Create a menu to access to the new view.
 - Open the Views -> Shared -> _Layout.cshtml partial view.
 - Add the next line after the Home menu (`@Html.ActionLink("Home", "Index", "Home")`).

```
<li>@Html.ActionLink("SQL Products", "Index", "Products")</li>
```

 - Save the _Layout.cshtml file.

Task 3: Test the Web Application

- Right click on the Solution from the Solution Explorer.
- Click on <Build Solution>.
- Once the Solution is builded, click on Debug-> Start Debugging menu or press or F5.
- Click in the new link <SQL Products>.
- You will be able to see, create, delete and edit products from the SQL table.
- Close the internet explorer.

Task 4: Publish Web Application

- In VS, in the Object Explorer, right click on the Project Name.
- Click on <Publish...>.
- On the publish window, click on <Publish>.
- A new page will be opened.
- Make sure the URL is the same of the development environment from the web Application.
- You will be able to see, create, delete and edit products from the SQL table.
- Close the internet explorer.

Module 5: Storing Unstructured Data in Azure

Objectives

- Create a Storage Account.
- Work with tables in a Storage Account.
- Connect table to a Web Application.
- Work with Cosmos DB.
- Connect Cosmos DB to a Web Application.

Prerequisites

- Complete exercises from Module 4.

Exercise 1: Work with Storage Account and Table

Task 1: Create a Storage Account

- Open Azure portal.
- Click on <Create a Resource>.
- Click on Storage -> Storage account – blob, file, table, queue.
- Complete <Basics> section:
 - For Project details.
 - Select a Subscription.
 - Select the Resource Group created in Modul 2, Exercise 1, Task 1.
 - For Instance details.
 - Add the Storage account name.
 - Select the closest Location.
 - Select <Storage V2 (General purpose v2)> in Account kind.
 - Select <Locally-redundant storage (LRS)> in Replication.
 - Leave the rest of the fields with their default value.
 - Click on <Review + create >.
- Complete the <Review + create> section:
 - Review the details of the new Storage Account.
 - Click on <Create>.
- A new blade will be opened (Deployment), where you can see the progress.

Task 2: Create a Table

- When the Storage Account is created, click on the <Go to resource> button at the top of the Deployment blade.
- Click on Table Service -> Tables from the left menu, or on <Tables> from the Services in the Overview.
- Click on <+ Table> from the top menu.
 - Add the Table name.
 - Click on <OK>.
- Save Storage Account key.
 - From your storage account, go to Settings -> Access keys.
 - Copy the Storage account name to a notepad for further use.

Exercise 2: Integrate Table to a Web Application

Task 1: Connect Storage Account to Web Application

- Open the solution you created in VS.
- Connect to Storage Account.
 - Right click on the Project from the Object Explorer.
 - Click on Add -> Connected Service.
 - In the new window, click on <Cloud Storage with Azure Storage>.
 - In the new dialog box select the Storage Account created.
 - Click on <Add>.
 - Save in a notepad the name of the new app setting created in the Web.config file, for further use.
- Create a Model.
 - Right click on the Models folder.
 - Click Add -> Class.
 - In the new dialog box:
 - Add the Name (ProductsFromTable.cs).
 - Click on <Add>.
 - In the new file, add the next library.

```
using Microsoft.WindowsAzure.Storage.Table;
```

```
using System.ComponentModel.DataAnnotations;
```

- Extend the class using the base class TableEntity.

```
public class ProductsFromTable : TableEntity
```

- Create two constructors, one empty and other to fill the *partitionKey* and the *rowKey*.

```
public ProductsFromTable(string name, string category)
```

```
{  
    PartitionKey = category;  
    RowKey = name;  
}
```

```
public ProductsFromTable() { }
```

- Add properties.

```
[Key]
```

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

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

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

- Rebuild the solution to confirm it is compiling.
 - Right click on the Solution from the Solution Explorer.
 - Click on <Build Solution>.

Task 2: Create Controller and Views

- Create a Controller.
 - Right click on the Controllers folder.
 - Click Add -> Controller.
 - In the new dialog box:
 - Select <MVC 5 Controller - Empty>.
 - Click on <Add>.
 - Add the Name (ProductsFromTableController).
 - Click on <Add>.

- Rebuild the solution to confirm it is compiling.
 - Right click on the Solution from the Solution Explorer.
 - Click on <Build Solution>.
- Create Views.
 - Add a view to **list** all the entities from a table
 - Right click on the Views -> ProductsFromTable folder.
 - Click Add -> View.
 - In the new dialog box:
 - Add the View name (Index).
 - Select <List> in the Template.
 - Select <ProductsFromTable ([ApplicationName].Models) > in the Model class.
 - Where [ApplicationName] is the namespace, which is the same of the application name.
 - Select ~/Views/Shared/_Layout.cshtml.
 - Click on <Add>.
 - Add a view to **create** a new entity.
 - Right click on the Views -> ProductsTable folder.
 - Click Add -> View for the list of products.
 - In the new dialog box:
 - Add the View name (Create).
 - Select <Create> in the Template.
 - Select <ProductsFromTable ([WebApplicationName].Models) > in the Model class.
 - Where [ApplicationName] is the namespace, which is the same of the application name.
 - Select ~/Views/Shared/_Layout.cshtml.
 - Click on <Add>.
- Modify the web.config
 - Open the Web.config file.
 - Add a setting with the table name to the <appSettings> section.

```
<add key="tableName" value="[tableName]" />
```

 - Where *tableName* is the name of the table previously created.- Modify the controller to list and create products in the storage account table.
 - Open the ProductsFromTableController.cs file.
 - Add the next libraries.

```
using Microsoft.Azure;
using Microsoft.WindowsAzure.Storage;
using Microsoft.WindowsAzure.Storage.Table;
using [ApplicationName].Models;
```

 - Where [ApplicationName] is the namespace, which is the same of the application name.
 - Create a constructor.

```
CloudStorageAccount storageAccount;
CloudTableClient tableClient;
CloudTable table;
public ProductsFromTableController()
```

```
{
    storageAccount = CloudStorageAccount.Parse( CloudConfigurationManager.GetSetting("[connectString]"));
    tableClient = storageAccount.CreateCloudTableClient();
    table = tableClient.GetTableReference( CloudConfigurationManager.GetSetting("tableName"));
}
```

- Where [connectString] is the name of the app setting where the storage account connection string is saved.

- Modify the Index method to retrieve the list of products from the table.

```
public ActionResult Index()
{
    TableQuery<ProductsFromTable> query = new TableQuery<ProductsFromTable>();
    List<ProductsFromTable> products = new List<ProductsFromTable>();
    TableContinuationToken token = null;
    do
    {
        TableQuerySegment<ProductsFromTable> resultSegment =
        table.ExecuteQuerySegmented(query, token);
        token = resultSegment.ContinuationToken;
        foreach (ProductsFromTable product in resultSegment.Results)
        {
            products.Add(product);
        }
    } while (token != null);
    return View(products);
}
```

- Create an action result to redirect to the create page.

```
public ActionResult Create()
{
    return View();
}
```

- Create the HttpPost method to add an entity.

```
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Create([Bind(Include = "PartitionKey, RowKey, id, ProductModel, Description")] ProductsFromTable product)
{
    if (ModelState.IsValid)
    {
        TableOperation insertOperation = TableOperation.Insert(product);
        table.Execute(insertOperation);
    }
    return RedirectToAction("Index");
}
```

- Add a link to the new Index.

- Open the Views -> Home -> Index.cshtml file.
- Add the next line after the line `<h2>@Html.ActionLink("Products", "Index", "Products")</h2>`

```
<h2>@Html.ActionLink("Table Products", "Index", "ProductsFromTable")</h2>
```

- Save the Index.cshtml file.

- Create a menu to access to the new view.

- Open the Views -> Shared -> _Layout.cshtml partial view.

- Add the next line after the SQL Products menu (`@Html.ActionLink("SQL Products", "Index", "Products")`).
- `@Html.ActionLink("Table Products", "Index", "ProductsFromTable")`
- Save the `_Layout.cshtml` file.

Task 3: Test the Web Application

- Right click on the Solution from the Solution Explorer.
- Click on `<Build Solution>`.
- Once the Solution is built, click on `Debug-> Start Debugging` menu or press or F5.
- Click in the new link `<Table Products>`.
- You will be able to see and create products from the Storage Account table.
- Close the internet explorer.

Task 4: Publish Web Application

- In VS, in the Object Explorer, right click on the Project Name.
- Click on `<Publish...>`.
- On the publish window, click on `<Publish>`.
- A new page will be opened.
- Make sure the URL is the same of the development environment from the web Application.
- Click in the new link `<Table Products>`.
- You will be able to see and create products from the Storage Account table.
- Close the internet explorer.

Exercise 3: Work with Cosmos DB

Task 1: Create a Cosmos DB using SQL API


- Open Azure portal.
- Click on `<Create a Resource>`.
- Click on `Databases -> Azure Cosmos DB`.
- Complete `<Basics>` section:
 - For Project details.
 - Select a Subscription.
 - Select the Resource Group created in Module 2, Exercise 1, Task 1.
 - For Instance details.
 - Add the account name.
 - Select SQL API.
 - Select the closest Location.
 - Leave the rest of the fields with their default value.
 - Click on `<Review + create>`.
- Complete the `<Review + create>` section:
 - Review the details of the new VM.
 - Click on `<Create>`.
- A new blade will be opened (Deployment), where you can see the progress.
- When the Cosmos DB is created, click on the `<Go to resource>` button at the top of the Deployment blade.
- Go to Overview in the Quick start blade.

- Review the Cosmos DB configuration.

Task 2: Change consistency to Cosmos DB

- Click on Settings -> Default Consistency, from the left menu.
- Select <Consistent Prefix>.
- Click on <Save> from the top menu.

Task 3: Replicate Cosmos DB

- Click on Settings -> Replicate data globally, from the left menu.
- In the Configure region section, click on <Add region> link.
- Select a new region from the <Search for a region> drop down list.
- Click on <OK>.
- Click again on <Add region>.
- Select another region from the map.
- Click on <Save> from the top menu.
 - The creation and synchronisation based on the consistency will start.
- After updated. To change automatically the Write Region in case of a failover.
 - Click on <Automatic Failover> from the top menu.
 - Click <ON> in Enable Automatic Failover.
 - Swap the Read regions priorities by dragging (click on the  icon) one region the position of the other region.
 - Click on <OK>.

Task 4: Add a DB and Collection to Cosmos DB

- Click on <Data Explorer> from the left menu.
- Click on <New Collection> from the top menu.
- Complete.
 - Select <Create new> in Database id.
 - Add the Database Id.
 - Add the Collection Id (ProductsDescription).
 - Change Throughput to 400.
 - Leave the rest of the fields with their default value.
 - Click <OK>.
- Add a Product.
 - Expand the new Collection [*CosmosDBName*] -> ProductsDescription.
 - Where [*CosmosDBName*] is the name of your Cosmos DB.
 - Select Documents.
 - Click on <New Document> from the Documents tab menu.
 - Replace the code between the {} for:


```
"ProductID": "1234",
"Name": "LL Bottom Bracket",
"ProductModel": "LL Bottom Bracket",
"Culture": "en",
"Description": "Chromoly steel."
```

- Click on <Save> from the Documents tab menu. You will see the new document is created with additional properties.

Exercise 4: Integrate Cosmos DB to a Web Application

Task 1: Connect Cosmos DB to Web Application

- In the Cosmos DB resource, go to Settings -> Keys from the left menu.
- Save the URI and Primary Key values in a notepad. Make sure they are Read-write Keys.
- Open the solution you created in VS.
- Add the CosmosDB library.
 - Right click on the project in the Object Explorer.
 - Click on Manage NuGet Packages for Solution...
 - Click on the <Browse> tab.
 - Type on the <Search> box: Microsoft.Azure.DocumentDB, and press Intro.
 - From the result select Microsoft.Azure.DocumentDB.
 - Click on <Install>.
 - Confirm the Reviewing Changes message.
 - Accept the License Acceptance message.
- Create a Model.
 - Right click on the Models folder.
 - Click Add -> Class.
 - In the new dialog box:
 - Add the Name (ProductsFromCosmos.cs).
 - Click on <Add>.
 - Add the next libraries in the new file.

`using Newtonsoft.Json;`

`using System.ComponentModel.DataAnnotations;`

- Add the next properties to the class ProductsFromCosmos.

```
[Key]
[JsonProperty(PropertyName = "Product Id")]
public string ProductId { get; set; }
[JsonProperty(PropertyName = "Name")]
public string Name { get; set; }
[JsonProperty(PropertyName = "Product model")]
public string ProductModel { get; set; }
[JsonProperty(PropertyName = "Culture")]
public string Culture { get; set; }
[JsonProperty(PropertyName = "Description")]
public string Description { get; set; }
```

- Save the file.
- Rebuild the solution to confirm it is compiling.
 - Right click on the Solution from the Solution Explorer.
 - Click on <Build Solution>.

Task 2: Create Controller and Views

- Create a controller.
 - Right click on the Controllers folder.
 - Click Add -> Controller.
 - In the new dialog box:

- Select <MVC 5 Controller - Empty>.
 - Click on <Add>.
 - Add the Name (ProductsFromCosmosController).
 - Click on <Add>.
- Create Views.
 - Right click on the Views -> ProductsFromCosmos folder.
 - Click Add -> View, for the list of products.
 - In the new dialog box:
 - Add the View name (Index).
 - Select <List> in the Template.
 - Select <ProductsFromCosmos ([ApplicationName].Models) > in the Model class.
 - Where [ApplicationName] is the namespace, which is the same of the application name.
 - Select ~/Views/Shared/_Layout.cshtml.
 - Click on <Add>.
 - Right click on the Views -> ProductsFromCosmos folder.
 - Click Add -> View, to create a new product.
 - In the new dialog box:
 - Add the View name (Create).
 - Select <Create> in the Template.
 - Select <ProductsFromCosmos ([ApplicationName].Models) > in the Model class.
 - Where [ApplicationName] is the namespace, which is the same of the application name.
 - Select ~/Views/Shared/_Layout.cshtml.
 - Click on <Add>.
- Add connection to CosmosDB.
 - Open <Web.config>.
 - Add the settings you saved from the CosmosDB resource in the <appSettings> section.

```
<add key="uri" value="[Resource_URI]" />
<add key="authenticationKey" value="[Resource_Key]" />
<add key="databaseId" value="[DatabaseId_name]" />
<add key="collectionId" value="[CollectionId_name]" />
```

 - Where :
 - [Resource_URI] is the URI.
 - [Resource_Key] is the Key whether primary or secondary, but make sure is a Read-write key.
 - [DatabaseId_name] is the name of the database.
 - [CollectionId_name] is the name of the collection (*ProductsDescription*).
- Save the file.

Task 3: Create Repository

- Right click on the Project from the Object Explorer.
- Click Add -> Class...
- In the new dialog box:
 - Add the name (CosmosDbRepository.cs).

- Click on <Add>.

- In the new file, add the next libraries.

```
using Microsoft.Azure.Documents.Client;
using Microsoft.Azure.Documents.Linq;
using System.Configuration;
using System.Threading.Tasks;
using System.Linq.Expressions;
using Microsoft.Azure.Documents;
```

- Change the code for the class CosmosDbRepository with the next code to list and create documents.

```
public static class CosmosDbRepository<T> where T : class
{
    private static readonly string dbId =
ConfigurationManager.AppSettings["databaseId"];
    private static readonly string CollectionId =
ConfigurationManager.AppSettings["collectionId"];
    private static DocumentClient client;

    public static void Initialize()
    {
        client = new DocumentClient(new
Uri(ConfigurationManager.AppSettings["uri"]),
ConfigurationManager.AppSettings["authenticationKey"]);
    }
    public static async Task<IEnumerable<T>> GetItemsAsync()
    {
        IDocumentQuery<T> query = client.CreateDocumentQuery<T>
(UriFactory.CreateDocumentCollectionUri(dbId, CollectionId))
.AsDocumentQuery();
        List<T> results = new List<T>();
        while (query.HasMoreResults)
        {
            results.AddRange(await query.ExecuteNextAsync<T>());
        }
        return results;
    }
    public static async Task<Document> CreateItemAsync(T item)
    {
        return await client.CreateDocumentAsync(
UriFactory.CreateDocumentCollectionUri(dbId, CollectionId), item);
    }
}
```

- Modify the ProductsFromCosmosController file.

- Add the next libraries.

```
using System.Threading.Tasks;
using [ApplicationName].Models;
```

- Where [ApplicationName] is the namespace, which is the same of the application name.

- Modify the Index method.

```
[ActionName("Index")]
public async Task<ActionResult> Index()
{
    var items = await CosmosDbRepository<ProductsFromCosmos>.GetItemsAsync();
    return View(items);
}
```

```

}
    ○ Add the Create action to redirect to the create page.
public ActionResult Create()
{
    return View();
}
    ○ Add the Create method.
[HttpPost]
[ActionName("Create")]
[ValidateAntiForgeryToken]
public async Task<ActionResult> CreateAsync([Bind(Include =
"ProductID,Name,ProductModel,Culture,Description")] ProductsFromCosmos item)
{
    if (ModelState.IsValid)
    {
        await CosmosDbRepository<ProductsFromCosmos>.CreateItemAsync(item);
        return RedirectToAction("Index");
    }
    return View(item);
}
    ○ Save the file.
• Add a link to the new Index.
    ○ Open the Views -> Home -> Index.cshtml file.
    ○ Add the next line after the line <h2>@Html.ActionLink("Table Products", "Index",
        "Table")</h2>.
<h2>@Html.ActionLink("Cosmos Products", "Index", "ProductsFromCosmos")</h2>
    ○ Save the Index.cshtml file.
• Create a menu to access to the new view.
    ○ Open the Views -> Shared -> _Layout.cshtml partial view.
    ○ Add the next line after the SQL Products menu (<li>@Html.ActionLink("Table
        Prodcuts", "Create", "Table")</li>).
<li>@Html.ActionLink("Cosmos Products", "Index", "ProductsFromCosmos")</li>
    ○ Save the _Layout.cshtml file.
• Add an initializer
    ○ Open the Global.asax.cs file
    ○ Add the next line to the method Application_Start()
CosmosDbRepository<Models.ProductsFromCosmos>.Initialize();

```

Task 4: Test the Web Application

- Right click on the Solution from the Solution Explorer.
- Click on <Build Solution>.
- Once the Solution is builded, click on Debug-> Start Debugging menu or press or F5.
- Click in the new link <Cosmos Products>.
- You will be able to see and create products from Cosmos DB.
- Close the internet explorer.

Task 5: Publish Web Application

- In VS, in the Object Explorer, right click on the Project Name.
- Click on <Publish...>.
- On the publish window, click on <Publish>.

- A new page will be opened.
- Make sure the URL is the same of the development environment from the web Application.
- Click in the new link <Cosmos Products>.
- You will be able to see and create products from Cosmos DB.
- Close the internet explorer.

Module 6: Storing and Consuming Files from Azure Storage

Objectives

- Create a Blob.
- Connect Blob to a Web Application.
- Create a File.
- Map File.

Prerequisites

- Complete exercises from Module 5.

Exercise 1: Work with Blob

Task 1: Create a Container

- Click on <All Services>.
- Click on Storage -> Storage accounts.
- Click on the previously created storage account.
- Click on Blob Service -> Blobs from the left menu, or on Blobs from the Services in the Overview.
- Click on <+ Container> from the top menu.
 - Add the Container name.
 - Click on <OK>.

Task 2: Create a Blob

- Click on the new container.
- Click on <Upload> from the top menu.
- In Files, browse for a new file.
 - Go to the new drive you created.
 - Right click.
 - Click on New -> Text Document.
 - Select the new document and click on <Open>.
 - Select SAS as the Authentication type.
 - Click on <Upload>.
- You will see your new blob listed in the container.

Exercise 2: Integrate Blob to a Web Application

Task 1: Create Controller and Views

- Open your Web Application in VS.
- Create the setting *containerName* in the web.config, here the value is the name of the container created in the Azure portal.

```
<add key="containerName" value="[containerName]" />
```

- Where the *ContainerName* is the name of the container created in the portal.
- Create a controller.
 - Right click on the Controllers folder.
 - Click Add -> Controller.
 - In the new dialog box:
 - Select <MVC 5 Controller - Empty>.
 - Click on <Add>.
 - Add the Name (BlobController).
 - Click on <Add>.
 - In the new Controller add the next references.

```
using Microsoft.Azure;
using Microsoft.WindowsAzure.Storage;
using Microsoft.WindowsAzure.Storage.Blob;
using System.IO;
```

- Add the next method, to get blobs from a container.

```
private CloudBlobContainer GetCloudBlobContainer()
{
    CloudStorageAccount storageAccount =
    CloudStorageAccount.Parse(CloudConfigurationManager.GetSetting("[connectionString]"));
    CloudBlobClient blobClient = storageAccount.CreateCloudBlobClient();
    CloudBlobContainer container = blobClient.GetContainerReference(
    CloudConfigurationManager.GetSetting("containerName"));
    container.CreateIfNotExists();
    return container;
}
```

- Where the *connectionString* is the app setting created in the Web.config.
- Update the Index method to list the available blobs.

```
public ActionResult Index()
{
    CloudBlobContainer container = GetCloudBlobContainer();
    List<string> blobs = new List<string>();
    foreach (IListBlobItem item in container.ListBlobs(useFlatBlobListing: true))
    {
        if (item.GetType() == typeof(CloudBlockBlob))
        {
            CloudBlockBlob blob = (CloudBlockBlob)item;
            blobs.Add(blob.Name);
        }
        else if (item.GetType() == typeof(CloudPageBlob))
        {
            CloudPageBlob blob = (CloudPageBlob)item;
            blobs.Add(blob.Name);
        }
        else if (item.GetType() == typeof(CloudBlobDirectory))
        {
            CloudBlobDirectory dir = (CloudBlobDirectory)item;
            blobs.Add(dir.Uri.ToString());
        }
    }
    return View(blobs);
}
```

- Add the next method to upload a blob.

```
[HttpPost]
```

```

public ActionResult UploadBlob(HttpPostedFileBase file)
{
    CloudBlobContainer container = GetCloudBlobContainer();
    CloudBlockBlob blob =
container.GetBlockBlobReference(Path.GetFileName(file.FileName));
    using (var fileStream = file.InputStream)
    {
        blob.UploadFromStream(fileStream);
    }
    return RedirectToAction("Index", "Blob");
}

```

- Add the next method to download a blob.

```

public ActionResult DownloadBlob(string blobName)
{
    CloudBlobContainer container = GetCloudBlobContainer();
    CloudBlockBlob blob = container.GetBlockBlobReference(blobName);
    MemoryStream stream = new MemoryStream();
    blob.DownloadToStream(stream);
    System.Web.HttpContext.Current.Response.ContentType =
blob.Properties.ContentType.ToString();
    System.Web.HttpContext.Current.Response.AddHeader("Content-Disposition",
"Attachment; filename=" + blobName);
    System.Web.HttpContext.Current.Response.AddHeader("Content-Length",
blob.Properties.Length.ToString());
    System.Web.HttpContext.Current.Response.BinaryWrite(stream.ToArray());
    System.Web.HttpContext.Current.Response.Flush();
    System.Web.HttpContext.Current.Response.Close();
    return RedirectToAction("Index", "Blob");
}

```

- Create View.
 - Right click on the Views -> Blob folder.
 - Click Add -> View for the list of products.
 - In the new dialog box:
 - Add the View name (Index).
 - Select <Empty (without model) > in the Model class.
 - Select ~/Views/Shared/_Layout.cshtml.
 - Click on <Add>.
 - Change the view content for the next lines of code:

```

@model List<string>
@{
    ViewBag.Title = "List of blobs";
}
<h2>List of blobs</h2>
<ul>
    @foreach (var item in Model)
    {
        <li>@item @Html.ActionLink("Download", "DownloadBlob", new { blobName = item })
        </li>
    }
</ul>
<h2>Upload a Blob</h2>
<form action="/Blob/UploadBlob" method="post" enctype="multipart/form-data">
    <input type="file" id="file" name="file" />
    <input type="submit" value="Upload" />
</form>

```

- Add a link to the new Index.
 - Open the Views -> Home -> Index.cshtml file.
 - Add the next line after `<h2>@Html.ActionLink("Cosmos Products", "Index", "ProductsFromCosmos")</h2>`.
`<h2>@Html.ActionLink("List of Blobs", "Index", "Blob")</h2>`
 - Save the Index.cshtml file.
- Create a menu to access to the new view.
 - Open the Views -> Shared -> _Layout.cshtml partial view.
 - Add the next line after the SQL Products menu (`@Html.ActionLink("Cosmos Products", "Index", "ProductsFromCosmos")`).
`@Html.ActionLink("List of Blobs", "Index", "Blob")`
 - Save the _Layout.cshtml file.

Task 2: Add Security

- Open the BlobController file.
- Add the next method in the class to create a SAS policy.

```
private string GetContainerSasUri(CloudBlockBlob blob)
{
    SharedAccessBlobPolicy sasConstraints = new SharedAccessBlobPolicy();
    sasConstraints.SharedAccessExpiryTime = DateTimeOffset.UtcNow.AddSeconds(10);
    sasConstraints.Permissions = SharedAccessBlobPermissions.List |
    SharedAccessBlobPermissions.Write;
    string sasContainerToken = blob.GetSharedAccessSignature(sasConstraints);
    return blob.Uri + sasContainerToken;
}
```

- Add the next line to the UploadBlob method, before returning the Action (`return RedirectToAction("Index", "Blob");`).

GetContainerSasUri(blob);

Task 3: Test the Web Application

- Right click on the Solution from the Solution Explorer.
- Click on <Build Solution>.
- Once the Solution is built, click on Debug-> Start Debugging menu or press or F5.
- Click in the new link <List of blobs>.
- You will be able to see the list of Blobs, upload new files and download them.
- Close the internet explorer.

Task 4: Publish Web Application

- In VS, in the Object Explorer, right click on the Project Name.
- Click on <Publish...>.
- On the publish window, click on <Publish>.
- A new page will be opened.
- Make sure the URL is the same of the development environment from the web Application.
- Click in the new link <Cosmos Products>.
- You will be able to see the list of Blobs, upload new files and download them.
- Close the internet explorer.

Task 5: Review Blobs in Azure portal

- Go to Azure portal.
- Open Storage Account resource.
- Click on the storage account you are using to store the blobs.
- Click on Blob from the Overview.
- Click on the container created.
- You will see the list of blobs added from the web application.

Exercise 3: Work with File

Task 1: Create a File

- Click on <All Services>.
- Click on Storage -> Storage accounts.
- Click on the previously created storage account.
- Click on File Service -> Files from the left menu, or on Files from the Services in the Overview.
- Click on <+ File share> from the top menu.
 - Add the File share name.
 - Click on <Create>.

Task 2: Map the File share as a drive to the Virtual Machine (previously created)

- Click on the created File share.
- On the File share blade, click on <Connect> from the top menu.
- Change the Drive letter if you already use the default one.
- Copy the alternatively command from the <Connecting from Windows>.
- Open the VM created in Module 2.
- Open a Command Prompt window.
- Paste the command copied before.
- Press intro.
- Open a Windows Explorer to check the new unit.

Module 7: Designing a Communication Strategy by Using Queues and Service Bus

Objectives

- Create a Storage Account Queue.
- Connect Storage Account Queue to a Web Application.
- Create a Service Bus Queue.
- Connect Service Bus Queue to a Web Application.

Prerequisites

- Complete exercises from Module 6.

Exercise 1: Work with Storage Account Queue

Task 1: Create a Queue

- Click on <All Services>.

- Click on Storage -> Storage accounts.
- Click on the previously created storage account.
- Click on Queue Service -> Queues from the left menu, or on Queues from the Services in the Overview.
- Click on <+ Queue> from the top menu.
 - Add the Queue name.
 - Click on <OK>.

Task 2: Add a Message

- Click on the new queue to access.
- Click on <+ Add message> from the top menu.
- Complete:
 - Add a message on the Message test.
 - Set the expiration time¹, where the maximum time is 7 days and the minimum is 1 second.
 - Click on <OK>.

Exercise 2: Integrate Storage Account Queue to a Web Application

Task 1: Connect Queue to Web Application

- Open your solution in VS.
- Connect to Storage Account.
 - Open the Web.config.
 - Save in a notepad the new connection created in the Web.config file.
 - Create a new app setting for the queue name in the <appSettings> section.


```
<add key="queueName" value="[queueName]"/>
```

 - Where *queueName* is the name of the Queue created in the portal.
- Create a Model.
 - Right click on the Models folder.
 - Click on Add -> class.
 - In the new dialog box:
 - Add the model name (QueueSa.cs).
 - Click on <Add>.
 - Add the next library.


```
using Microsoft.WindowsAzure.Storage.Queue;
```
 - Add the next properties in the new class.


```
public int TotalOfMessages { get; set; }
public string Message { get; set; }
public IEnumerable<CloudQueueMessage> Messages { get; set; }
```
 - Rebuild the solution to confirm it is compiling.
 - Right click on the Solution from the Solution Explorer.
 - Click on <Build Solution>.

Task 2: Create Controller and Views

- Create a controller.
 - Right click on the Controllers folder.
 - Click Add -> Controller.

- In the new dialog box:
 - Select <MVC 5 Controller - Empty>.
 - Click on <Add>.
 - Add the Name (QueueSaController).
 - Click on <Add>.
- In the new Controller add the next references.

```
using Microsoft.Azure;
using Microsoft.WindowsAzure.Storage;
using Microsoft.WindowsAzure.Storage.Queue;
```

- Add the constructor, to create the connection to the Storage Account.

```
private CloudQueue queue;
public QueueSaController()
{
    CloudStorageAccount storageAccount =
    CloudStorageAccount.Parse(CloudConfigurationManager.GetSetting("[connectionString]"));
    CloudQueueClient queueClient = storageAccount.CreateCloudQueueClient();
    queue =
    queueClient.GetQueueReference(CloudConfigurationManager.GetSetting("queueName"));
    queue.CreateIfNotExists();
}
```

- Where the *connectionString* is the app setting created in the Web.config.
- Add the next method to get the total of messages listed in the memory.

```
private int? GetTotalOfMessage()
{
    queue.FetchAttributes();
    return queue.ApproximateMessageCount;
}
```

- Replace the index method.

```
public ActionResult Index()
{
    int total = GetTotalOfMessage() ?? 0;
    return View(new Models.QueueSa()
    {
        TotalOfMessages = total,
        Messages = queue.GetMessages(total == 0 ? 1 : total, new TimeSpan(5000))
    });
}
```

- Add the next method to insert a message.

```
[HttpPost]
public ActionResult AddMessage(string message)
{
    queue.AddMessage(new CloudQueueMessage(string.Format(message)));
    return View("Index", new Models.QueueSa()
    {
        TotalOfMessages = GetTotalOfMessage() ?? 0,
        Messages = queue.GetMessages(GetTotalOfMessage() ?? 0, new TimeSpan(1000))
    });
}
```

- Add the next method to peek a message.

```
public ActionResult PeekMessage()
{
    int total = GetTotalOfMessage() ?? 0;
    CloudQueueMessage peekMessage = null;
    peekMessage = queue.PeekMessage();
}
```

```

if (peekMessage != null)
{
    return View("Index", new Models.QueueSa()
    {
        TotalOfMessages = total,
        Message = string.Format("PEEK: {0}", peekMessage.AsString),
        Messages = queue.GetMessages(total == 0 ? 1 : total, new
        TimeSpan(1000))
    });
}
return View("Index", new Models.QueueSa()
{
    TotalOfMessages = total,
    Message = "PEEK: There are no messages."
});
}

```

- Add the next method to get a message.

```

public ActionResult GetMessage()
{
    int total = 0;
    CloudQueueMessage getMessage = null;
    getMessage = queue.GetMessage();
    if (getMessage != null)
    {
        queue.DeleteMessage(getMessage);
        total = GetTotalOfMessage() ?? 0;
        return View("Index", new Models.QueueSa()
        {
            TotalOfMessages = total,
            Message = string.Format("GET: {0}", getMessage.AsString),
            Messages = queue.GetMessages(total == 0 ? 1 : total, new
            TimeSpan(1000))
        });
    }
    return View("Index", new Models.QueueSa()
    {
        TotalOfMessages = total,
        Message = "GET: There are no messages."
    });
}

```

- Add the next methods to modify a message.

```

[HttpPost]
public ActionResult UpdateMessage([Bind(Include = "message")]string message)
{
    int total = 0;
    CloudQueueMessage getMessage = null;
    getMessage = queue.GetMessage();
    if (getMessage != null)
    {
        getMessage.SetMessageContent(message);
        queue.UpdateMessage(getMessage, TimeSpan.FromSeconds(1),
        MessageUpdateFields.Content | MessageUpdateFields.Visibility);
        total = GetTotalOfMessage() ?? 0;
        return View("Index", new Models.QueueSa()
        {
            TotalOfMessages = total,
            Message = string.Format("UPDATE: {0}", getMessage.AsString),

```

```

        Messages = queue.GetMessages(total == 0 ? 1 : total, new
        TimeSpan(1000))
    });
}
return View("Index", new Models.QueueSa()
{
    TotalOfMessages = total,
    Message = "UPDATE: There are no messages."
});
}

```

- Save the controller file.
- Create View.
 - Right click on the Views -> QueueSa folder.
 - Click Add -> View for the list of products.
 - In the new dialog box:
 - Add the View name (Index).
 - Select <Empty (without model)> in the Template.
 - Click on <Add>.
 - Change the view content for the next lines of code:

```

@model [ApplicationName].Models.QueueSa
@{
    ViewBag.Title = "SA Queue messages";
}
<br />
<h1>Storage Account Queue</h1>
<br />
<form action="/QueueSa/AddMessage" method="post" enctype="multipart/form-data">
    <input type="text" name="message" id="message" />
    <br />
    <button type="submit" value="Add message">Add</button>
    <button type="submit" formaction="PeekMessage" formenctype="multipart/form-data"
formmethod="post">Peek</button>
    <button type="submit" formaction="GetMessage" formenctype="multipart/form-data"
formmethod="post">Get</button>
    <button type="submit" formaction="UpdateMessage" formenctype="multipart/form-data"
formmethod="post">Update</button>
</form>
<br /><br />
@Html.DisplayTextFor(m => m.Message)
<br /><br />
<h3>Total of messages:</h3>
@Html.DisplayTextFor(m => m.TotalOfMessages)
<br /><br />
<h1>List of messages</h1>
<ul>
    @if (Model.Messages != null)
    {
        foreach (var item in Model.Messages)
        {
            <li>@item.AsString</li>
        }
    }
</ul>

```

- Where [ApplicationName] is the namespace, which is the same of the application name.

- Add a link to the new Index.
 - Open the Views -> Home -> Index.cshtml file.
 - Add the next line after `<h2>@Html.ActionLink("List of Blobs", "ListBlob", "Blob")</h2>`.
`<h2>@Html.ActionLink("Storage Account Queue", "Index", "QueueSa")</h2>`
 - Save the Index.cshtml file.
- Create a menu to access to the new view.
 - Open the Views -> Shared -> _Layout.cshtml partial view.
 - Add the next line after the Blobs menu `@Html.ActionLink("List of Blobs", "ListBlob", "Blob")`.
`@Html.ActionLink("SA Queue", "Index", "QueueSa")`
 - Save the _Layout.cshtml file
- Test project.
 - Press F5 to start the project.
 - Click on <Queue messages > link.
 - You will see the list of messages available.
 - Click on <Add> to add a message.
 - Click on <Peek> to see the next message.
 - Click on <Peek> to retrieve the next message.

Task 3: Test the Web Application

- Right click on the Solution from the Solution Explorer.
- Click on <Build Solution>.
- Once the Solution is built, click on Debug-> Start Debugging menu or press or F5.
- Click in the new link <Storage Account Queue>.
- You will be able to see the list of Messages available in the Queue, Add, Peek, Get and Update.
- Close the internet explorer.

Task 4: Publish Web Application

- In VS, in the Object Explorer, right click on the Project Name.
- Click on <Publish...>.
- On the publish window, click on <Publish>.
- A new page will be opened.
- Make sure the URL is the same of the development environment from the web Application.
- Click in the new link <Storage Account Queue>.
- You will be able to see the list of Messages available in the Queue, Add, Peek, Get and Update.
- Close the internet explorer.

Exercise 3: Work with Service Bus Queue

Task 1: Create a Service Bus namespace

- Open the Azure portal.
- Click on <Create a Resource>.
- Click on Integration -> Service Bus.
- Complete:
 - Add the Name.

- Select the Pricing tier (Basic).
- Select a Subscription.
- Select the Resource Group created previously.
- Select the closest Location.
- Click on < create>.

Task 2: Create a Queue

- When the Service Bus is created, from notifications click on <Go to resource group>.
- Close the Notifications.
- Click on Entities-> Queues from the left menu, or on <+ Queue> from the top menu in the Overview.
- Complete:
 - Add the queue Name.
 - Change the Max queue size¹.
 - Leave the rest of the fields with their default value.
 - Click on <Create>.
- Save the connection string
 - Click on Settings -> Shared access policies from the left menu of the Service Bus blade.
 - Select the policy listed.
 - Copy any of the Connection String to a notepad for further use.

Exercise 4: Integrate Service Bus Queue to a Web Application

Task 1: Connect Queue to Web Application

- Open your project in VS.
- Add the ServiceBus library.
 - Right click on the project.
 - Click on Manage NuGet Packages for Solution...
 - Click on the <Browse> tab.
 - Type on the <Search> box: Microsoft.Azure.ServiceBus, and press Intro.
 - From the result select Microsoft.Azure.ServiceBus.
 - Select the version **1.0.0**.
 - Click on <Install>.
 - Confirm the Reviewing Changes message.
 - Accept the License Acceptance message
- Connect to Service Bus.

- Create two app settings.
 - Open the Web.config from the project.
 - Add the next two lines, in the <appSettings> section.

```
<add key="ServiceBusConnectionString" value="[ConnectionString]"/>
<add key="ServiceBusQueue" value="[QueueName]"/>
```

- Where the *ConnectionString* is the connection string from the Service Bus you saved before and *QueueName* is the queue name created in the Service Bus resource.

- Save the Web.config file.
- Create Model.

- Right click on the Models folder.
- Click on Add -> class.
- In the new dialog box:
 - Add the model name (QueueSb.cs).
 - Click on <Add>.
- Add the next properties in the new class.


```
public int TotalOfMessages { get; set; }
public string Message { get; set; }
```
- Rebuild the solution to confirm it is compiling.
 - Right click on the Solution from the Solution Explorer.
 - Click on <Build Solution>.

Task 2: Create Controller and Views

- Create a controller.
 - Right click on the Controllers folder.
 - Click Add -> Controller.
 - In the new dialog box:
 - Select <MVC 5 Controller - Empty>.
 - Click on <Add>.
 - Add the Name (QueueSbController).
 - Click on <Add>.
 - In the new controller, add the next references.

```
using Microsoft.Azure;
using Microsoft.Azure.ServiceBus;
using Microsoft.Azure.ServiceBus.Core;
using System.Threading.Tasks;
using System.Text;
```

- Create the class constructor and the queue client and receiver.

```
private IQueueClient queueClient;
private MessageReceiver receiver;
public QueueSbController()
{
    queueClient = new
    QueueClient(CloudConfigurationManager.GetSetting("ServiceBusConnectionString"),
    CloudConfigurationManager.GetSetting("ServiceBusQueue"));
    receiver = new
    MessageReceiver(CloudConfigurationManager.GetSetting("ServiceBusConnectionString"),
    CloudConfigurationManager.GetSetting("ServiceBusQueue"));
}
```

- Replace the index method to show the number of messages available in the queue.

```
public async Task<ActionResult> Index()
{
    return View(new Models.QueueSb()
    {
        TotalOfMessages = await GetTotalMessages()
    });
}
```

- Add the next method to insert new messages, receiving a total of messages to insert.

```
[HttpPost]
public async Task<ActionResult> AddMessage(int totalMessage)
{
}
```

```

        Message message;
        for (int i = 0; i < totalMessage; i++)
        {
            message = new Message(Encoding.UTF8.GetBytes(string.Format("Test message {0}", i)));
            message.Label = string.Format("msj {0}", i);
            await queueClient.SendAsync(message);
        }
        await queueClient.CloseAsync();
        return RedirectToAction("Index", "QueueSb");
    }

```

- Add the next method to retrieve the total messages available in the queue.

```

private async Task<int> GetTotalMessages()
{
    int cont = 0;
    while (true)
    {
        var message = await receiver.PeekAsync();
        if (message != null)
            cont++;
        else
            break;
    }
    return cont;
}

```

- Add the next methods to peek the next message.

```

public async Task<ActionResult> PeekMessage()
{
    string result = "PEEK: There are no messages";
    var message = await receiver.PeekAsync();
    int total = await GetTotalMessages();
    if (message != null)
    {
        result = string.Format("PEEK:{0}", Encoding.UTF8.GetString(message.Body));
        total++;
    }
    await receiver.CloseAsync();
    return View("Index", new Models.QueueSb()
    {
        TotalOfMessages = total,
        Message = result
    });
}

```

- Add the next method to get the next message.

```

public async Task<ActionResult> GetMessage()
{
    string result = "GET: There are no messages ";
    var message = await receiver.ReceiveAsync();
    if (message != null)
    {
        result = string.Format("GET :{0}",
        Encoding.UTF8.GetString(message.Body));
        await receiver.CompleteAsync(message.SystemProperties.LockToken);
    }
    Task.WaitAll();
    int total = await GetTotalMessages();
    await receiver.CloseAsync();
}

```

```

        return View("Index", new Models.QueueSb()
        {
            TotalOfMessages = total,
            Message = result
        });
    }
}

```

- Save the controller file.
- Create View.
 - Right click on the Views -> QueueSb folder.
 - Click Add -> View for the list of products.
 - In the new dialog box:
 - Add the View name (Index).
 - Select <Empty (without model) > in the Model class.
 - Select ~/Views/Shared/_Layout.cshtml.
 - Click on <Add>.
- Change the view content for the next lines of code.

```

@model [ApplicationName].Models.QueueSb
@{
    ViewBag.Title = "SB Queue messages";
}
<br />
<h1>Service Bus Queue</h1>
<br />
<form action="/QueueSb/AddMessage" method="post" enctype="multipart/form-data">
    <input type="number" name="totalMessage" id="totalMessage" />
    <br />
    <button type="submit" value="Add message">Add</button>
    <button type="submit" formaction="PeekMessage" formenctype="multipart/form-data"
formmethod="post">Peek</button>
    <button type="submit" formaction="GetMessage" formenctype="multipart/form-data"
formmethod="post">Get</button>
</form>
<br /><br />
@Html.DisplayTextFor(m => m.Message)
<br /><br />
<h1>Total of messages:</h1>
@Html.DisplayTextFor(m => m.TotalOfMessages)

```

- Where [ApplicationName] is the namespace, which is the same of the application name.
- This code will ask for a number of messages to insert; will show 3 buttons: Add, Peek message and Get message; will show the total of messages available in the queue.
- Add a link to the new Index.

```

<h2>@Html.ActionLink("Storage Account Queue", "Index", "QueueSa")</h2>
<h2>@Html.ActionLink("Service Bus Queue", "Index", "QueueSb")</h2>
    ○ Save the Index.cshtml file.
    • Create a menu to access to the new view.
        ○ Open the Views -> Shared -> _Layout.cshtml partial view.
        ○ Add the next line after the Queue menu (<li>@Html.ActionLink("SA Queue", "Index", "QueueSa")</li>).
<li>@Html.ActionLink("SB Queue", "Index", "QueueSb")</li>

```


- Save the _Layout.cshtml file.

Task 3: Test the Web Application

- Right click on the Solution from the Solution Explorer.
- Click on <Build Solution>.
- Once the Solution is builded, click on Debug-> Start Debugging menu or press or F5.
- Click in the new link <Service Bus Queue>.
- You will be able to see the Add the number of messages to insert in the queue, Peek, Get and see the total of messages available.
- Close the internet explorer.

Task 4: Publish Web Application

- In VS, in the Object Explorer, right click on the Project Name.
- Click on <Publish...>.
- On the publish window, click on <Publish>.
- A new page will be opened.
- Make sure the URL is the same of the development environment from the web Application.
- Click in the new link <Service Bus Queue>.
- You will be able to see the Add the number of messages to insert in the queue, Peek, Get and see the total of messages available.
- Close the internet explorer

Module 8: Securing Azure Web Applications

Objectives

- Connect Web Application to Azure Active Directory.

Prerequisites

- Complete exercises from Module 2.

Exercise 1: Secure Web Application with Active Directory

Task 1: Connect Web Application with Azure Active Directory.

- Connect to Authentication with Azure Active Directory.
 - Right click on the Project.
 - Click on Add -> Connected Service.
 - In the new dialog box, click on <Authentication with Azure Active Directory>.
 - In the new dialog box.
 - Click <Next> for the Introduction.
 - In the Single Sign-on
 - Select a Domain from your Azure subscription.
 - Select <Use settings from an existing Azure AD application to configure your project (a new application will not be created)>.
 - Add the URI from the web application resource, we have been using.
 - Click on <Next>.

- Leave the Directory Access with the <Read directory data> option unchecked.
 - Click on <Finish>.
- Once the service is connected, you will see:
 - Each one of your Controllers have the attribute [Authorize].
 - A new controller: Controllers -> AccountController.
 - A new partial view: Views -> Account -> SignoutCallback.cshtml.
 - A new class for authentication startup: App_Start -> Startuop.Auth.cs
 - A new partial view to show the login user details: Views -> Shared -> _LoginPartial.cshtml.
 - Beside new references and changes in the we.config file.
- Add to _Layout the new created partial view for showing the log in details after.
 - Open the _Layout.cshtml file from Views -> Shared path.
 - Add the next line after the list of menus ().

@Html.Partial("_LoginPartial")

- Start debugging your project.
 - Sign in with an account from your directory.
 - Accept the <Permissions requested>.
- You will see on the top right corner the signed user name.

Task 2: Test the Web Application

- Right click on the Solution from the Solution Explorer.
- Click on <Build Solution>.
- Once the Solution is builded, click on Debug-> Start Debugging menu or press or F5.
- Click in the new link <Service Bus Queue>.
- You will be able to see the Add the number of messages to insert in the queue, Peek, Get and see the total of messages available.
- Close the internet explorer.

¹. Optional.

².Remember this value to access to the VM.

³.Remember this value to access to the SQL Server instance.

References

https://docs.microsoft.com/en-us/azure/cosmos-db/sql-api-dotnet-application#_Toc395637764

<https://docs.microsoft.com/en-us/azure/visual-studio/vs-storage-aspnet-getting-started-blobs>

<https://docs.microsoft.com/en-us/azure/storage/blobs/storage-dotnet-shared-access-signature-part-2>

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

<https://docs.microsoft.com/en-us/azure/service-bus-messaging/service-bus-dotnet-get-started-with-queues>