



Microsoft Cloud Workshop

Modern cloud apps

Hands-on lab step-by-step

March 2018

Information in this document, including URL and other Internet Website references, is subject to change without notice. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

The names of manufacturers, products, or URLs are provided for informational purposes only and Microsoft makes no representations and warranties, either expressed, implied, or statutory, regarding these manufacturers or the use of the products with any Microsoft technologies. The inclusion of a manufacturer or product does not imply endorsement of Microsoft of the manufacturer or product. Links may be provided to third party sites. Such sites are not under the control of Microsoft and Microsoft is not responsible for the contents of any linked site or any link contained in a linked site, or any changes or updates to such sites. Microsoft is not responsible for webcasting or any other form of transmission received from any linked site. Microsoft is providing these links to you only as a convenience, and the inclusion of any link does not imply endorsement of Microsoft of the site or the products contained therein.

© 2018 Microsoft Corporation. All rights reserved.

Microsoft and the trademarks listed at <https://www.microsoft.com/en-us/legal/intellectualproperty/Trademarks/Usage/General.aspx> are trademarks of the Microsoft group of companies. All other trademarks are property of their respective owners.

Contents

Modern cloud apps hands-on lab step-by-step	1
Abstract and learning objectives.....	1
Overview	1
Requirements	1
Help references.....	3
Before the hands-on lab	5
Task 1: Setup a development environment.....	5
Task 2: Disable IE Enhanced Security.....	5
Task 3: Install SQL Server Management Studio.....	6
Task 4: Validate connectivity to Azure	6
Task 5: Download and explore the Contoso Sports League sample.....	6
Task 6: Create a new Azure Resource Group	7
Exercise 1: Proof of concept deployment.....	8
Task 1: Deploy the e-commerce website, SQL Database, and storage	8
Task 2: Setup SQL Database Geo-Replication	18
Task 3: Deploying the call center admin website	31
Task 4: Deploying the payment gateway	35
Task 5: Deploying the offers Web API.....	39
Task 6: Update and deploy the e-commerce website	44
Exercise 2: Identity and security.....	49
Task 1: Enable Azure AD Premium Trial.....	49
Task 2: Create a new Contoso user	51
Task 3: Configure access control for the call center administration Web Application.....	52
Task 4: Apply custom branding for the Azure Active Directory logon page	56
Task 5: Verify the branding has been successfully applied to the Azure Active Directory logon page.....	58
Exercise 3: Enable Azure B2C for customer site	60
Task 1: Create a new directory.....	60
Task 2: Add a new application.....	62
Task 3: Create Policies, Sign up	63
Task 4: Create a sign-in policy	65
Task 5: Create a profile editing policy.....	66
Task 6: Modify the Contoso.App.SportsLeague.Web	67
Task 7: Send authentication requests to Azure AD.....	73
Task 8: Display user information.....	75
Task 9: Run the sample app.....	78
Exercise 4: Enabling Telemetry with Application Insights	80
Task 1: Configure the application for telemetry	81

Task 2: Creating the web performance test and load test.....	88
Exercise 5: Automating backend processes with Azure Functions and Logic Apps	94
Task 1: Create an Azure Function to Generate PDF Receipts	94
Task 2: Create an Azure Logic App to Process Orders.....	100
Task 3: Use Twilio to send SMS Order Notifications.....	115
After the hands-on lab	131
Task 1: Delete resources	131

Modern cloud apps hands-on lab step-by-step

Abstract and learning objectives

In this Microsoft Cloud Workshop, attendees will implement an end-to-end solution for e-commerce that is based on Azure App Services, Azure Active Directory, and Visual Studio Online. Attendees will ensure the solution is PCI-compliant, and appropriate security measures are put into place for both on-prem and public access scenarios.

Attendees will be better able to deploy and configure Azure Web Apps and associated services. In addition,

- Configure Web Apps for authentication with Azure AD
- Instrument and load-test the application with App Insights
- Automate backend services using Cloud Services and Logic Apps

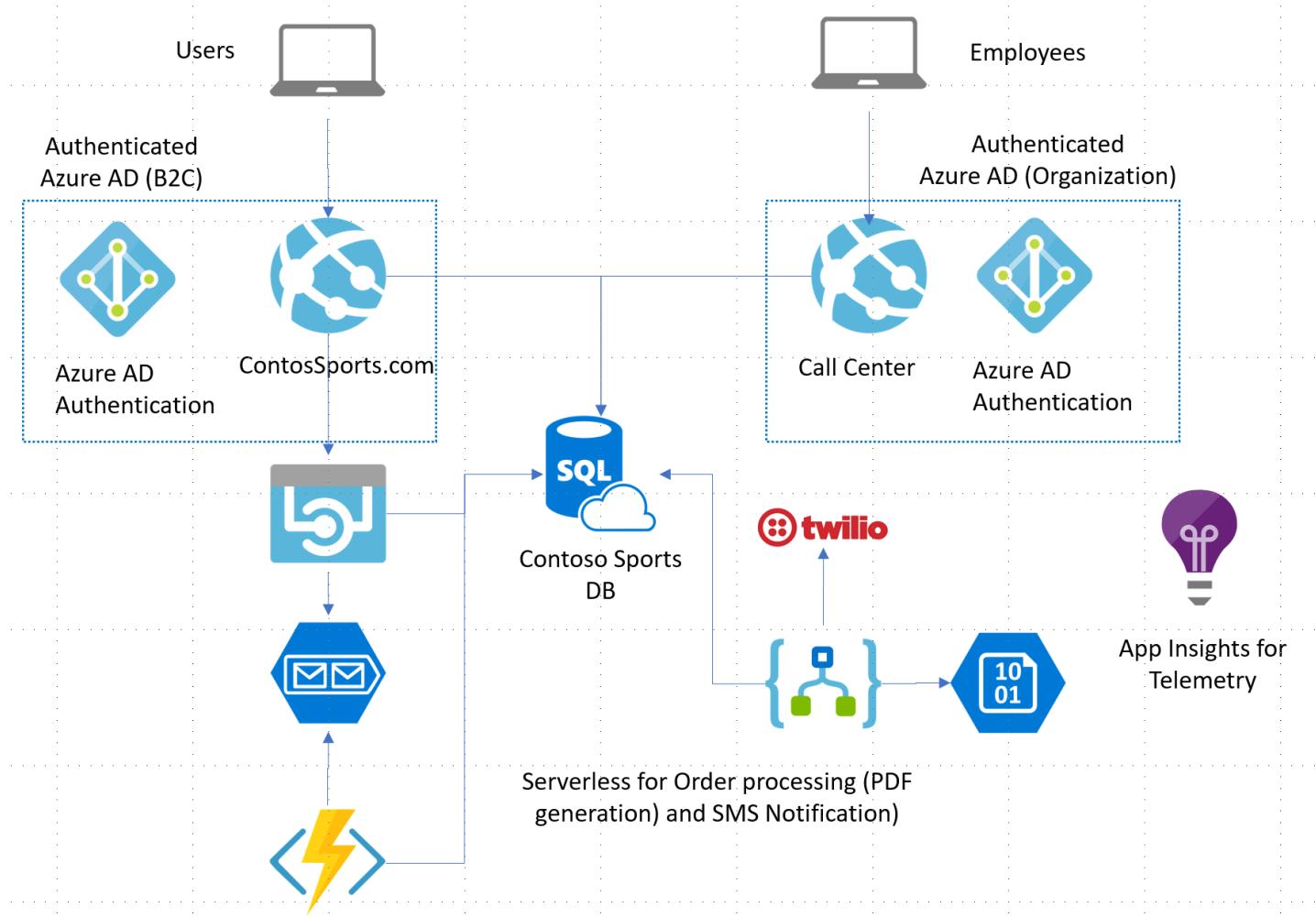
Overview

The Modern cloud apps Hackathon is a hands-on exercise that will challenge you to implement an end-to-end scenario using a supplied sample that is based on Microsoft Azure App Services and related services. The scenario will include implementing compute, storage, security, and scale using various components of Microsoft Azure. The Hackathon can be implemented on your own, but it is highly recommended to pair up with other members at the Hackathon to model a real-world experience much closer and to allow each member to share their expertise for the overall solution.

Requirements

1. Microsoft Azure subscription
2. Local machine or a virtual machine configured with:
 - a. Visual Studio 2017 Community Edition

Solution architecture



Help references

Description	Links
SQL firewall	https://azure.microsoft.com/en-us/documentation/articles/sql-database-configure-firewall-settings/
Deploying a Web App	https://azure.microsoft.com/en-us/documentation/articles/web-sites-deploy/
Deploying an API app	https://azure.microsoft.com/en-us/documentation/articles/app-service-dotnet-deploy-api-app/
Accessing an API app from a JavaScript client	https://azure.microsoft.com/en-us/documentation/articles/app-service-api-javascript-client/
SQL Database Geo-Replication overview	https://azure.microsoft.com/en-us/documentation/articles/sql-database-geo-replication-overview/
What is Azure AD?	https://azure.microsoft.com/en-us/documentation/articles/active-directory-whatis/
Azure Web Apps authentication	http://azure.microsoft.com/blog/2014/11/13/azure-websites-authentication-authorization/
View your access and usage reports	https://msdn.microsoft.com/en-us/library/azure/dn283934.aspx
Custom branding an Azure AD Tenant	https://msdn.microsoft.com/en-us/library/azure/Dn532270.aspx
Service Principal Authentication	https://docs.microsoft.com/en-us/azure/app-service-api/app-service-api-dotnet-service-principal-auth
Consumer Site B2C	https://docs.microsoft.com/en-us/azure/active-directory-b2c/active-directory-b2c-devquickstarts-web-dotnet
Getting Started with Active Directory B2C	https://azure.microsoft.com/en-us/trial/get-started-active-directory-b2c/
How to Delete an Azure Active Directory	https://blog.nicholasrogoff.com/2017/01/20/how-to-delete-an-azure-active-directory-add-tenant/
Run performance tests on your app	http://blogs.msdn.com/b/visualstudioalm/archive/2015/09/15/announcing-public-preview-for-performance-load-testing-of-azure-webapp.aspx
Application Insights Custom Events	https://azure.microsoft.com/en-us/documentation/articles/app-insights-api-custom-events-metrics/
Enabling Application Insights	https://azure.microsoft.com/en-us/documentation/articles/app-insights-start-monitoring-app-health-usage/
Detect failures	https://azure.microsoft.com/en-us/documentation/articles/app-insights-asp-net-exceptions/

Monitor performance problems	https://azure.microsoft.com/en-us/documentation/articles/app-insights-web-monitor-performance/
Creating a Logic App	https://azure.microsoft.com/en-us/documentation/articles/app-service-logic-create-a-logic-app/
Logic app connectors	https://azure.microsoft.com/en-us/documentation/articles/app-service-logic-connectors-list/
Logic Apps Docs	https://docs.microsoft.com/en-us/azure/logic-apps/logic-apps-what-are-logic-apps
Azure Functions – create first function	https://docs.microsoft.com/en-us/azure/azure-functions/functions-create-first-azure-function
Azure Functions docs	https://docs.microsoft.com/en-us/azure/logic-apps/logic-apps-azure-functions

Before the hands-on lab

Duration: 30 minutes

Before initiating the hands-on lab, you will setup an environment to use for the rest of the exercises.

Task 1: Setup a development environment

1. Create a virtual machine in Azure using the Visual Studio Community 2017 with the latest release on Windows Server 2016 image.

NAME	PUBLISHER	CATEGORY
Visual Studio Community 2017 on Windows Server 2016 (x64)	Microsoft	Compute
Visual Studio Community 2017 (version 15.1) on Windows Server 2016 (x64)	Microsoft	Compute

Note: It is **highly** recommended to use a DS2 or D2 instance size for this VM.

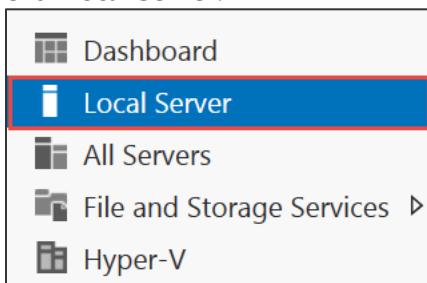
Task 2: Disable IE Enhanced Security

Note: Sometimes this image has IE ESC disabled, and sometimes it does not.

1. On the new VM you just created, select the Server Manager icon.



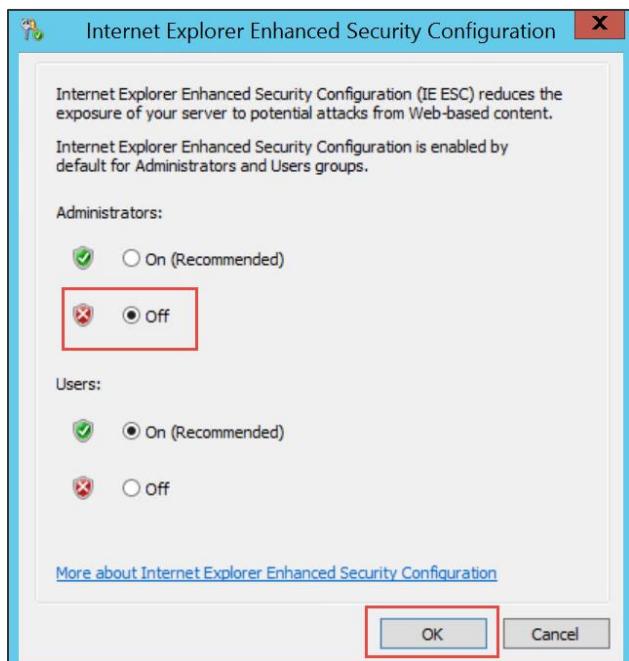
2. Click **Local Server**.



3. On the right side of the pane, click **On** by IE Enhanced Security Configuration.



4. Change to **Off** for Administrators, and click **OK**.

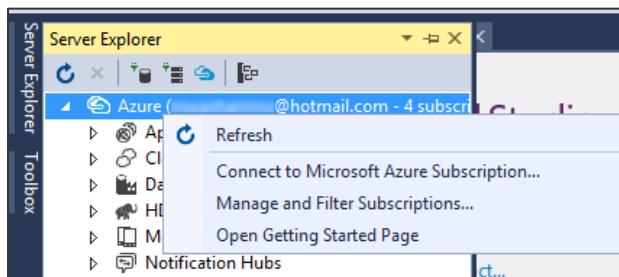


Task 3: Install SQL Server Management Studio

1. On the new VM, download and install SQL Server Management Studio.
<https://msdn.microsoft.com/en-us/library/mt238290.aspx>

Task 4: Validate connectivity to Azure

1. Within the virtual machine, launch Visual Studio, and validate you can login with your Microsoft Account when prompted.
2. Validate connectivity to your Azure subscription. Launch Visual Studio, open Server Explorer from the View menu, and ensure that you can connect to your Azure subscription.



Task 5: Download and explore the Contoso Sports League sample

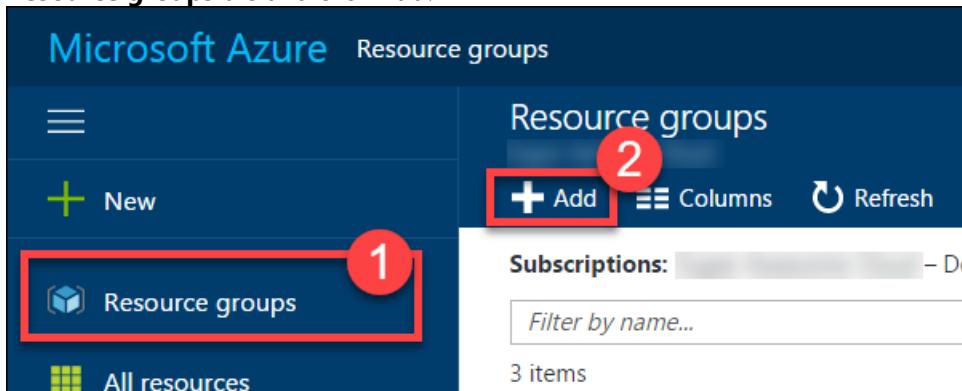
1. Create a new folder on your C: drive named **Hackathon**.
2. Download the sample application from here: <https://cloudworkshop.blob.core.windows.net/modern-cloud-apps/Modern%20Cloud%20Apps%20Student%20Files.zip> and extract to the **Hackathon** folder.

3. From the **Contoso Sports League** folder under **Hackathon**, open the Visual Studio Solution file: **Contoso.Apps.SportsLeague.sln**.
4. The solution contains the following projects:

Contoso.Apps.SportsLeague.Web	Contoso Sports League e-commerce application
Contoso.Apps.SportsLeague.Admin	Contoso Sports League call center admin application
Contoso.Apps.SportsLeague.Data	Data tier
Contoso.Apps.SportsLeague.Offers	API for returning list of available products
Contoso.Apps.PaymentGateway	API for payment processing

Task 6: Create a new Azure Resource Group

1. Create a new folder on your C: drive named **Hackathon**. Within the Azure Management Portal, open the **Resource groups** tile and click **Add**.



2. Specify the name of the resource group as **contososports**, and choose the Azure region you want to deploy the lab to. This resource group will be used throughout the rest of the lab. Click **Create** to create the resource group.

The screenshot shows the 'Resource group' creation dialog. It has fields for 'Resource group name' (contososports), 'Subscription' (selected), and 'Resource group location' (West US). The 'Create' button is at the bottom right.

You should follow all steps provided *before* attending the hands-on lab.

Exercise 1: Proof of concept deployment

Duration: 60 minutes

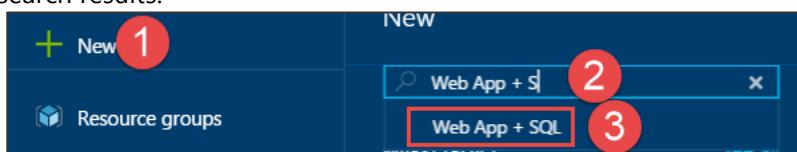
Contoso has asked you to create a proof of concept deployment in Microsoft Azure by deploying the web, database, and API applications for the solution as well as validating that the core functionality of the solution works. Ensure all resources use the same resource group previously created for the App Service Environment.

Task 1: Deploy the e-commerce website, SQL Database, and storage

In this exercise, you will provision a website via the Azure Web App + SQL template using the Microsoft Azure Portal. You will then edit the necessary configuration files in the starter project and deploy the e-commerce website.

Subtask 1: Create the Web App and SQL database instance

1. Navigate to the Azure Management portal, <http://portal.azure.com>, using a new tab or instance, navigate to create **Web App + SQL**.
2. Click **New**, and in the Marketplace search text box, enter “**Web App + S.**” Click the **Web App + SQL** item in the search results.



3. On the **Everything** blade, select **Web App + SQL**.

NAME	PUBLISHER	CATEGORY
Web App + SQL	Microsoft	Web + Mobile

4. Click **Create**.

Create and deploy web sites in seconds, as powerful as you need them

Leverage your existing tools to create and deploy applications without the hassle of managing infrastructure. Microsoft Azure Web Sites offers secure and flexible development, deployment, and scaling options for any sized web application. Use frameworks and templates to create web sites in seconds. Choose from source control options like TFS, GitHub, and BitBucket. Use any tool or OS to develop your site with .NET, PHP, Node.js or Python.

Use this Azure template to create a Website and SQL Database together to start developing even faster.

- Fastest way to build for the cloud
- Provision and deploy fast
- Secure platform that scales automatically
- Great experience for Visual Studio developers
- Open and flexible for everyone
- Monitor, alert, and auto scale (preview)

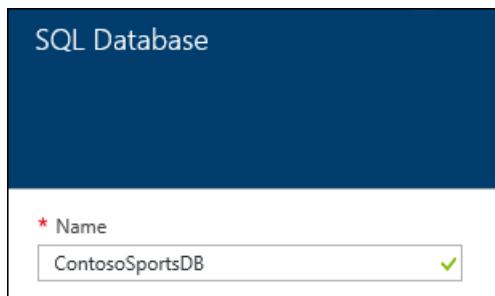
5. On the **Web App + SQL** blade, select **App Service plan/Location, Configure required settings**.



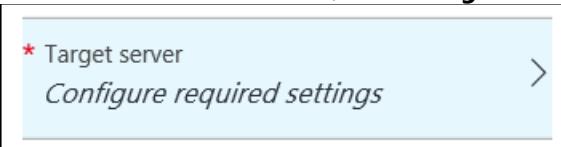
6. Create a new App Service plan called **ContosoSportsPlan** in the same region and with the **S1 Standard** pricing tier.
7. On the Web App blade, specify the following configuration:
- Specify a unique and valid name (until the green check mark appears)
 - Specify the **contososports** resource group.
 - Specify the name **ContosoSportsPlan** as the App Service plan and choose the same location as the Resource Group.

8. Select **SQL Database Configure required settings**, and click **Create a new database**.

9. On the **SQL Database** blade, specify **ContosSportsDB** as the database name.

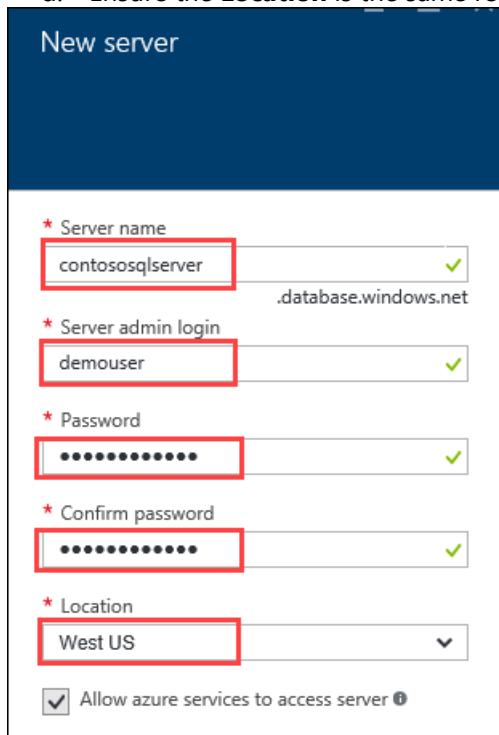


10. On the **SQL Database** blade, select **Target Server Configure required settings**.



11. On the **New server** blade, specify the following configuration:

- Server name: a unique value (ensure the green checkmark appears)
- Server admin login: **demouser**
- Password** and **Confirm Password**: demo@pass123
- Ensure the **Location** is the same region as the Web App.



12. Once the values are accepted in the **New server** blade, click **Select**.



13. On the **SQL Database** blade, click **Select**.



14. After the values are accepted, click **Create**.

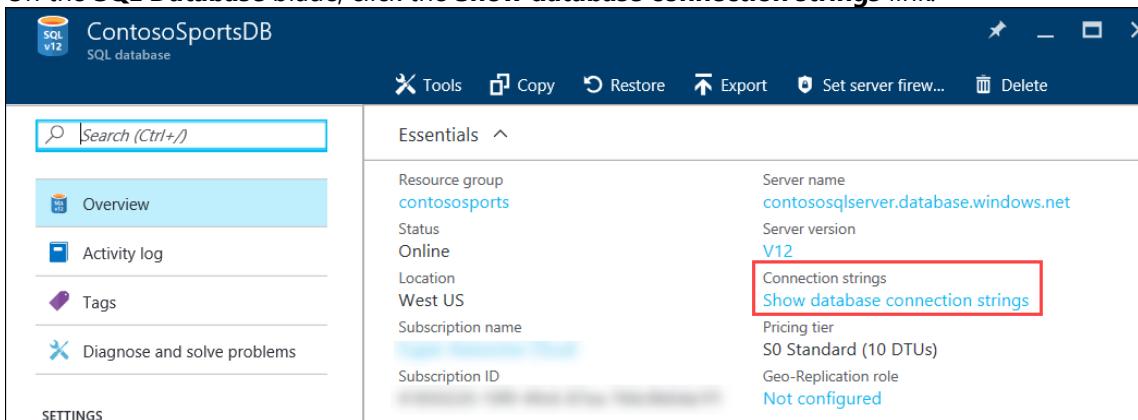


This may take a couple minutes to provision the Web App and SQL Database resources.

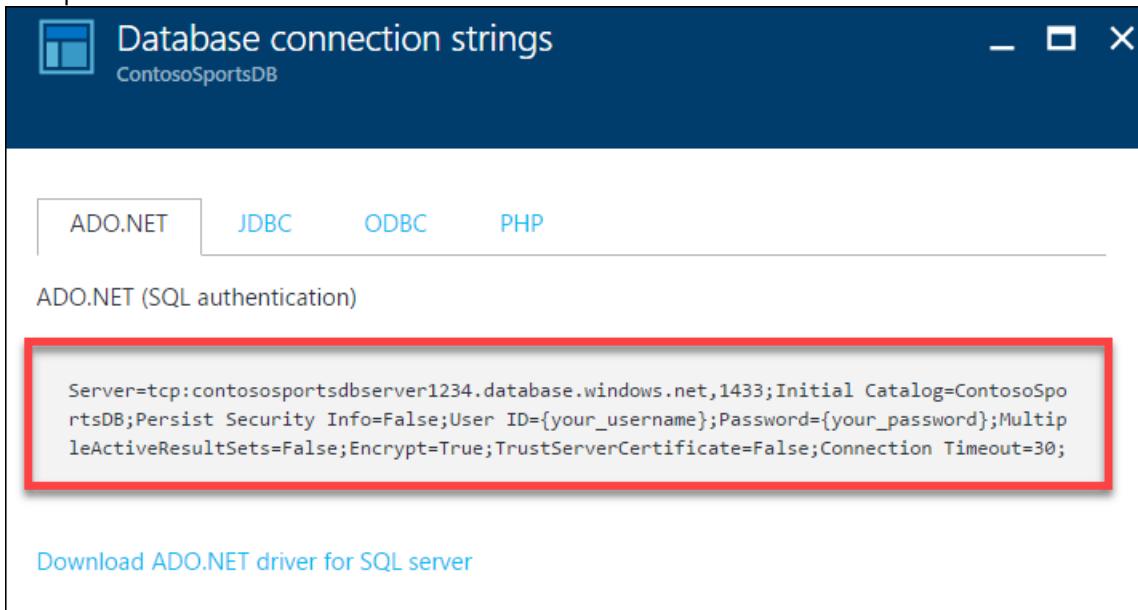
15. After the Web App and SQL Database are provisioned, click **More services > SQL databases** followed by the name of the SQL Database you just created.



16. On the **SQL Database** blade, click the **Show database connection strings** link.



17. On the **Database connection strings** blade, select and copy the **ADO.NET** connection string. Then, save it in Notepad for use later.



18. Click the SQL Database server name link.

Essentials ^

Resource group
contososports

Status
Online

Location
West US

Subscription name

Subscription ID

Server name
contososqlserver.database.windows.net

Server version
V12

Connection strings
[Show database connection strings](#)

Pricing tier
S0 Standard (10 DTUs)

Geo-Replication role
Not configured

19. On the **SQL Server** blade, under **Firewall**, click **Show firewall settings**.

contososqlserver SQL server

+ New pool Import database Reset password Delete Move

Search (Ctrl+ /)

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

Essentials ^

Resource group
contososports

Status
Available

Location
West US

Subscription name

Subscription ID

Server version
V12

Auditing
Not configured

Server admin
demouser

Active Directory admin
Not configured

Firewall
Show firewall settings

20. On the **Firewall Settings** blade, specify a new rule named **ALL**, with START IP **0.0.0.0**, and END IP **255.255.255.255**.

RULE NAME	START IP	END IP
ALL	0.0.0.0	255.255.255.255

This is only done to make the lab easier to do. In Production, you do NOT want to open up your SQL Database to all IP Addresses this way. In Production you will want to specify just the IP Addresses you wish to allow through the Firewall.

21. Click **Save**.

Firewall settings
contososqlserver (SQL server)

Save Discard Add client IP

22. On the **Success!** dialog box, click **OK**.

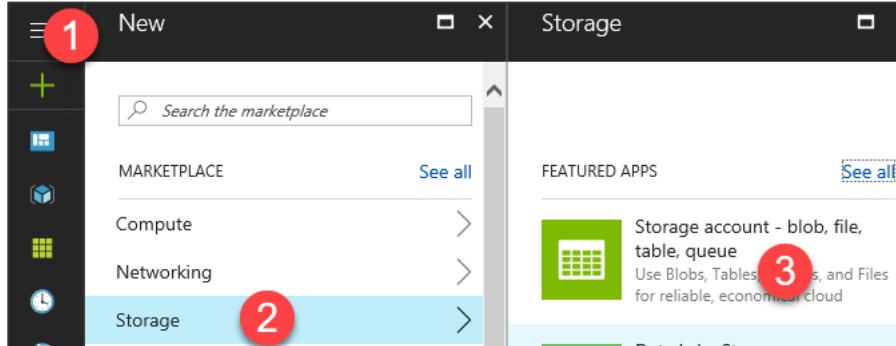
Success!
Successfully updated server firewall rules

Ok

23. Close all configuration blades.

Subtask 2: Provision the storage account

1. Using a new tab or instance of your browser, navigate to the Azure Management portal <http://portal.azure.com>.
2. Click **+New, Storage, and Storage account.**



3. On the Create storage account blade specify the following configuration options:
 - a. Name: unique value for the storage account (ensure the green check mark appears)
 - b. Specify the Resource Group **contososports**
 - c. Specify the same **Location** as the resource group.

The screenshot shows the 'Create storage account' blade. It includes fields for Name (contososportsstorag011), Deployment model (Resource manager), Account kind (General purpose), Performance (Standard), Replication (Read-access geo-redundant storage), Storage service encryption (Enabled), Secure transfer required (Enabled), Subscription (selected), Resource group (contososports), and Location (South Central US).

4. Click **Create**.



5. Once the storage account has completed provisioning, open the storage account by clicking **More services > Storage accounts** and clicking on the storage account name.

The screenshot shows the Azure portal's sidebar. The 'More services' section is expanded, showing 'Storage accounts'. The 'Storage accounts' item is highlighted with a blue background and white text. To the right of the list is a small star icon.

6. On the **Storage account** blade, click **All settings**.



7. On the **Storage** account blade, scroll down, and select the **Access keys** option.

The screenshot shows the 'contososports01' Storage account blade. In the left sidebar, there is a 'SETTINGS' section with a red box around the 'Access keys' option. In the main content area, there is an 'Essentials' panel on the right with various account details. The 'Access keys' option is highlighted with a red box.

8. On the **Access keys** blade, click the copy button by **key1** on the Connection string. Put the value in notepad for later reference.

contoso

The screenshot shows the 'Access keys' blade for the 'contoso' storage account. It lists two keys: 'key1' and 'key2'. The 'key1' row shows a connection string: 'DefaultEndpointsProtocol=https;AccountName=contoso;...'. A red box highlights the copy icon (a clipboard with a plus sign) next to the connection string for 'key1'.

Subtask 3: Update the configuration in the starter project

1. In the Azure Portal, click on **Resource Groups**. Then, click on the **contososports** resource group.

The screenshot shows the Azure Resource groups blade. On the left, there's a sidebar with links: 'Resource groups' (highlighted with a red box and number 1), 'All resources', 'Recent', and 'App Services'. The main area shows a list of resource groups under 'Subscriptions: Super Awesome Cloud'. One resource group, 'contososports', is highlighted with a red box and number 2.

2. Click on the **Web App** just created in a previous step.
3. On the **App Service** blade, scroll down in the left pane, and click on **Application settings**.

The screenshot shows the Azure App Service blade for the app 'contososportsweb4'. The left sidebar has sections: 'SETTINGS' (with 'Application settings' highlighted with a red box and number 4), 'Authentication / Authorization', 'Backups', and 'Custom domains'.

4. Scroll down, and locate the **App settings** section.

Key	Value	Slot setting	...
WEBSITE_NODE_DEFAULT_V...	6.9.1	<input type="checkbox"/>	...
AzureQueueConnectionString	(empty)	<input type="checkbox"/>	...

5. Add a new **App setting** with the following values:
 - a. Key: **AzureQueueConnectionString**

- b. Value: **enter the Connection String for the Azure Account just created**

App settings	
WEBSITE_NODE_DEFAULT_V...	6.9.1
AzureQueueConnectionString	DefaultEndpointsProtocol=https;AccountName=...;AccountKey=...

6. Locate **Connection Strings** below App settings.

Connection strings	
defaultConnection	< Hidden for Securi... SQL Database

7. Add a new **Connection String** with the following values:

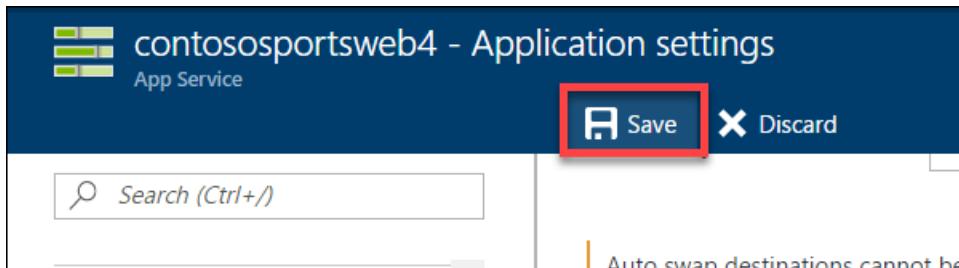
- Name: **ContosoSportsLeague**
- Value: **enter the Connection String for the SQL Database just created**
- Type: **SQL Database**

Connection strings	
defaultConnection	< Hidden for Securi... SQL Database
ContosoSportsLeague	Server=tcp:contoso... SQL Database

Ensure you replace the string placeholder values **{your_username}** **{your_password_here}** with the username and password you respectively setup during creation (demouser AND demo@pass123).

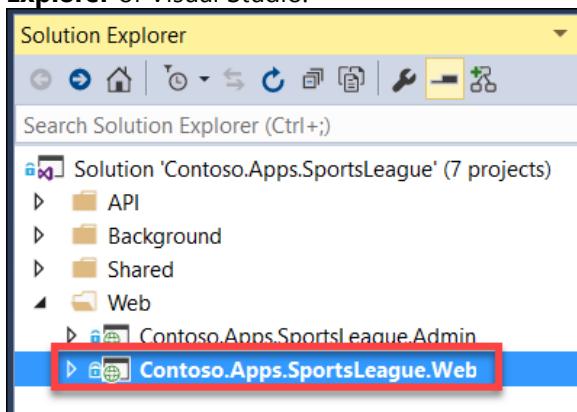
;Password={your_password_here};

8. Click Save.

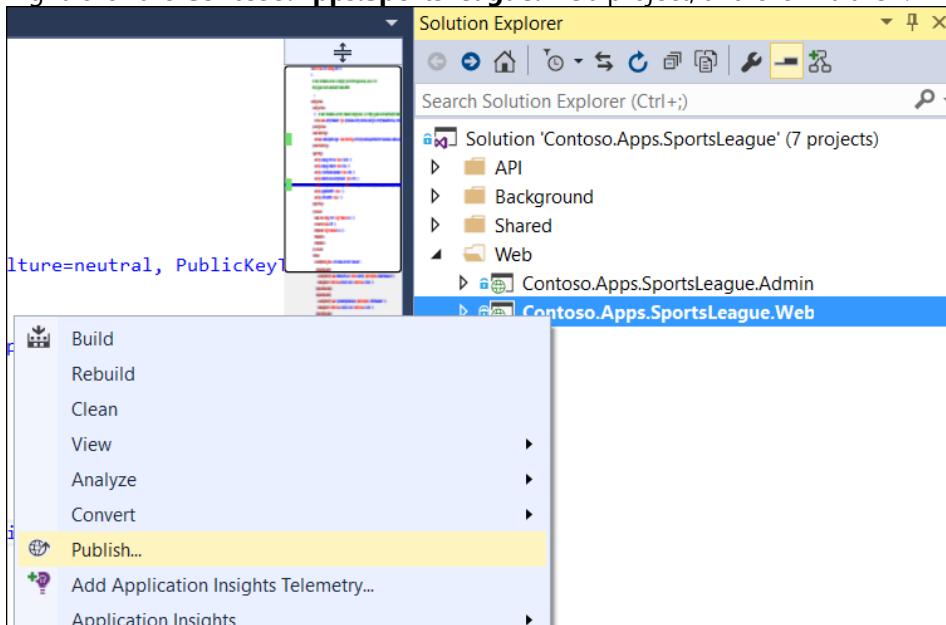


Subtask 4: Deploy the e-commerce Web App from Visual Studio

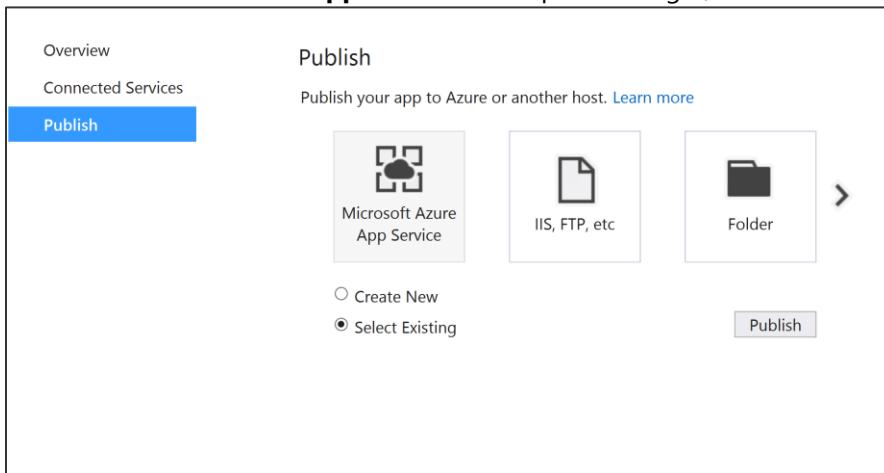
1. Navigate to the **Contoso.Apps.SportsLeague.Web** project located in the **Web** folder using the **Solution Explorer** of Visual Studio.



2. Right-click the **Contoso.Apps.SportsLeague.Web** project, and click **Publish**.



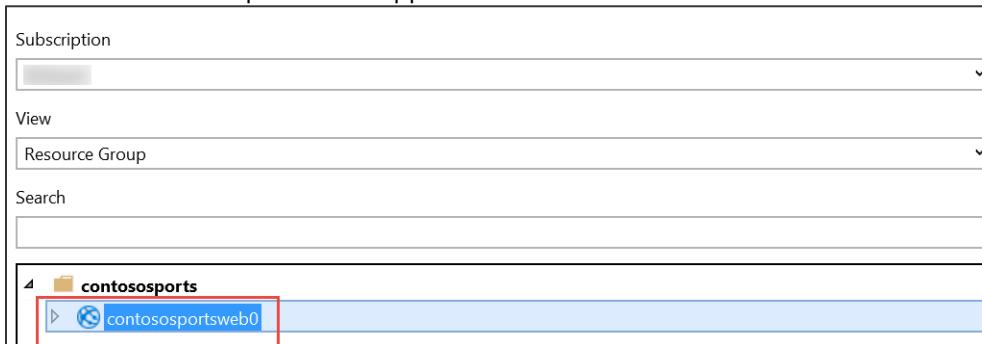
3. Choose **Microsoft Azure App Service** as the publish target, and choose **Select Existing**.



4. If prompted, log on with your credentials, and ensure the subscription you published earlier are selected.



5. Select the Contoso Sports Web App.



6. Click **OK**, and click **Publish** to publish the Web Application.

7. In the Visual Studio **Output** view, you will see a status that indicates the Web App was published successfully.

```
3>Adding ACL's for path (contososportsleagueweb)
3>Adding ACL's for path (contososportsleagueweb)
3>Adding ACL's for path (contososportsleagueweb/App_Data)
3>Publish Succeeded.
```

8. Validate the website by clicking the **Store** link on the menu. As long as products return, the connection to the database is successful.



Task 2: Setup SQL Database Geo-Replication

In this exercise, the attendee will provision a secondary SQL Database and configure Geo-Replication using the Microsoft Azure Portal.

Subtask 1: Add secondary database

1. Using a new tab or instance of your browser, navigate to the Azure Management Portal <http://portal.azure.com>

2. Click **More services > SQL databases**, and click the name of the SQL Database you created previously.

The screenshot shows the 'SQL databases' blade in the Azure Management Portal. On the left, there's a sidebar with 'More services >' and a star icon. The main area is titled 'DATABASES' and lists 'SQL databases'. A yellow star icon is located in the top right corner of the main content area.

3. Under **Settings**, click on **Geo-Replication**.

The screenshot shows the 'SETTINGS' blade in the Azure Management Portal. It includes options like 'Quick start', 'Pricing tier (scale DTUs)', and 'Geo-Replication'. The 'Geo-Replication' option is highlighted with a red box.

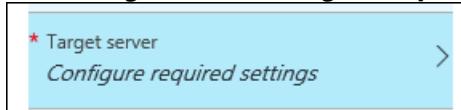
4. Select the Azure Region to place the Secondary within.

The screenshot shows the 'Geo-Replication' configuration blade. It displays a world map with various regions marked by colored circles. Below the map, under 'PRIMARY', it shows 'West US' with the status 'Online'. In the 'SECONDARIES' section, it says 'Geo-Replication is not configured'. Under 'TARGET REGIONS', 'East US' is listed with the status 'Recommended' (indicated by a red box), while 'West US' and 'Central US' are also listed.

The Secondary Azure Region should be the Region Pair for the region the SQL Database is hosted in. The portal suggests the Region Pair to use by labeling it as "Recommended."

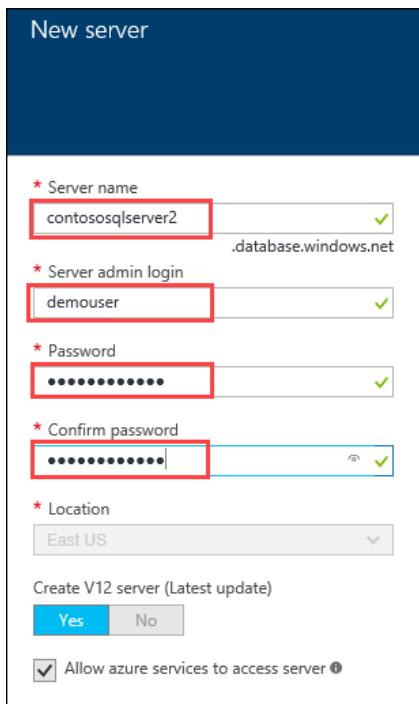
5. On the **Create secondary** blade, select **Secondary Type as Readable**.

6. Select **Target server Configure required settings.**



7. On the **New server** blade, specify the following configuration:

- Server name: a unique value (ensure the green checkmark appears)
- Server admin login: **demouser**
- Password and Confirm Password: **demo@pass123**



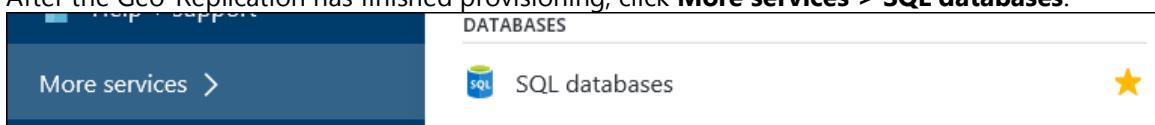
8. Once the values are accepted in the **New server** blade, click **Select**.



9. On the **Create secondary** blade, click **OK**.



10. After the Geo-Replication has finished provisioning, click **More services > SQL databases**.



11. Click the name of the Secondary SQL Database you just created.

NAME	STATUS	REPLICATION ROLE	SERVER
ContosoSportsDB	Online	Primary	contososqlserver
ContosoSportsDB	Online	Secondary	contososqlserver2

12. On the **SQL Database** blade, click the **Show database connection strings** link.

The screenshot shows the 'ContosoSportsDB' blade in the Azure portal. The left sidebar has links for Overview, Activity log, Tags, and Diagnose and solve problems. The main area shows the 'Essentials' section with details like Resource group (contososports), Status (Online), Location (East US), Subscription name, and Subscription ID. A red box highlights the 'Connection strings' link under the 'Show database connection strings' heading.

13. On the **Database connection strings** blade, select and copy the **ADO.NET** connection string, and save it in Notepad for use later.

The screenshot shows the 'Database connection strings' blade for 'ContosoSportsDB'. It has tabs for ADO.NET, JDBC, ODBC, and PHP. The ADO.NET tab is selected, showing the 'ADO.NET (SQL authentication)' section with a connection string. A red box highlights the connection string text:

```
Server=tcp:contososqlserver2.database.windows.net,1433;Initial Catalog=ContosoSportsDB;Persist Security Info=False;User ID={your_username};Password={your_password};MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate=False;Connection Timeout=30;
```

Below the connection string is a link to 'Download ADO.NET driver for SQL server'.

14. Click the SQL Database Server name link.

The screenshot shows the 'Essentials' section of the SQL Database blade. It includes details like Resource group (contososports), Status (Online), Location (East US), Subscription name, and Subscription ID. A red box highlights the 'Server name' link under the 'contososqlserver2.database.windows.net' heading.

15. On the **SQL Server** blade, under **Firewall**, click **Show firewall settings**.

The screenshot shows the Azure portal interface for a SQL server named 'contososqlserver2'. The left sidebar has links for Overview, Activity log, Access control (IAM), Tags, and Diagnose and solve problems. The main content area is titled 'Essentials' and shows details like Resource group ('contosports'), Status ('Available'), Location ('East US'), Subscription name (''), and Subscription ID (''). On the right, there are sections for Server version ('V12'), Auditing ('Not configured'), Server admin ('demouser'), Active Directory admin ('Not configured'), and Firewall (''). A red box highlights the 'Show firewall settings' link under the Firewall section.

16. On the **Firewall Settings** blade, specify a new rule named **ALL**, with START IP **0.0.0.0** and END IP **255.255.255.255**.

RULE NAME	START IP	END IP
ALL	0.0.0.0	255.255.255.255

17. Click **Save**.

The screenshot shows the 'Firewall settings' blade for 'contososqlserver2'. It includes a shield icon, the title 'Firewall settings', and the server name 'contososqlserver2 (SQL server)'. Below are three buttons: 'Save' (highlighted with a red box), 'Discard', and '+ Add client IP'.

18. On the **Success!** Dialog box, click **OK**.

The screenshot shows a 'Success!' dialog box with the message 'Successfully updated server firewall rules' and an 'Ok' button at the bottom.

19. Close all configuration blades.

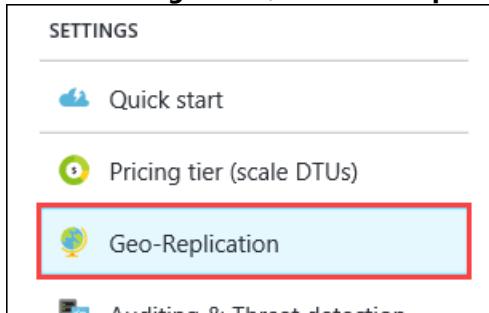
Subtask 2: Failover secondary SQL database – OPTIONAL

Since the Replication and Failover process can take anywhere from 10 – 30 minutes to complete, you have the choice to skip Subtask 2 through 5, and skip directly to Task 3. However, if you have the time, it is recommended that you complete these steps.

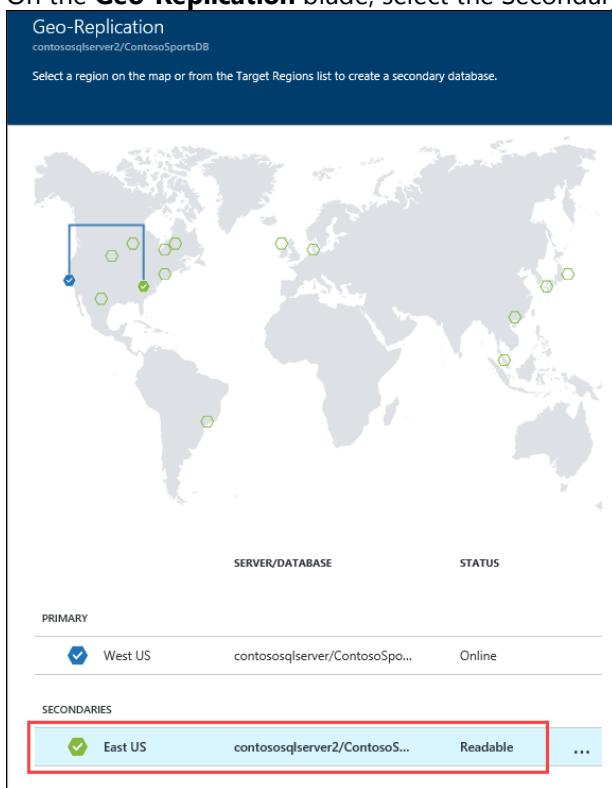
1. Using a new tab or instance of your browser, navigate to the Azure Management Portal <http://portal.azure.com>.
2. Click **More services > SQL databases**, and click the name of the SQL Database you created previously.

The screenshot shows the Azure portal navigation bar with 'More services >' and the 'SQL databases' option. There is also a yellow star icon.

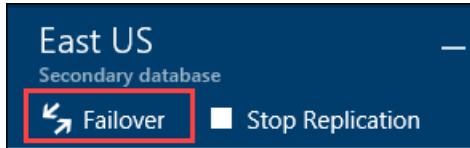
3. On the **Settings** blade, click **Geo-Replication**.



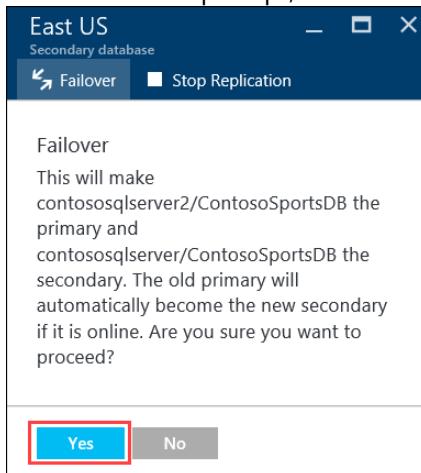
4. On the **Geo-Replication** blade, select the Secondary database.



5. Click the **Failover** button.



6. On the **Failover** prompt, click **Yes**.



The Failover may take a few minutes to complete. You can continue with the next Subtask modifying the Web App to point to the Secondary SQL Database while the Failover is pending.

Subtask 3: Test e-commerce Web App after Failover

1. Once completed, in the Azure Portal, click on **SQL databases**, and select the **ContosoSportsDB** secondary.

A screenshot of the Azure SQL Databases blade. The top navigation bar shows "SQL databases" and "rfustinohotmail (Default Directory)". Below the navigation are three buttons: "+ Add", "Columns", and "Refresh".
A message "Subscriptions: All 2 selected – Don't see a subscription? [Switch directories](#)" is displayed.
A search bar is followed by a link "All subscriptions".
The text "3 items" is shown.
A table lists the databases:

NAME	STATUS	REPLICATION ROLE
ContosoSportsDB	Online	Secondary
ContosoSportsDB	Online	Primary

2. Next, click on **Show database connection strings**, and copy it off thereby replacing the user and password.

The screenshot shows the Azure portal interface for a SQL database named 'ContosoSportsDB'. The left sidebar has options like 'Overview', 'Activity log', 'Tags', and 'Diagnose and solve problems'. The main pane shows 'Essentials' details including resource group ('contososports'), status ('Online'), location ('West US 2'), subscription name ('Developer Program Benefit'), and subscription ID ('3d49c824-2226-4b8d-bd19-d9e3bc11e656'). On the right, there's a section for 'Connection strings' with a link 'Show database connection strings' which is highlighted with a red box.

- From the Azure portal, click on resource groups, and select contososports.

The screenshot shows the 'Resource groups' blade in the Azure portal. The left sidebar has 'Resource groups' (marked with a red box and number 1) selected, 'All resources', 'Recent', and 'App Services'. The main pane lists 'Resource groups' under 'Subscriptions: Super Awesome Cloud' with 4 items. One item, 'contososports', is highlighted with a red box and number 2.

- Click on the **Web App** just created in a previous step.
- On the **App Service** blade, scroll down in the left pane, and click on **Application settings**.

The screenshot shows the 'App Service' blade for 'contososportsweb4'. The left sidebar has 'Application settings' (marked with a red box and number 1) selected, followed by 'Authentication / Authorization', 'Backups', and 'Custom domains'.

- Scroll down, and locate the **Connection strings** section.

7. Update the **ContosoSportsLeague** Connection String to the value of the Connection String for the **Secondary SQL Database**.

The screenshot shows the 'Connection strings' blade in the Azure portal. It lists a single entry: 'defaultConnection' with a value of 'ContosoSportsLe...'. The 'Value' field for 'ContosoSportsLe...' is highlighted with a red box. Below it, there's a table with columns 'Name' and 'Value'.

Ensure you replace the string placeholder values **{your_username}** **{your_password_here}** with the username and password you respectively setup during creation (demouser AND demo@pass123).

`;Password={your_password_here};`

8. Click **Save**.

The screenshot shows the 'Application settings' blade for the 'contososportsweb4' app service. It has a search bar and two buttons: 'Save' (highlighted with a red box) and 'Discard'.

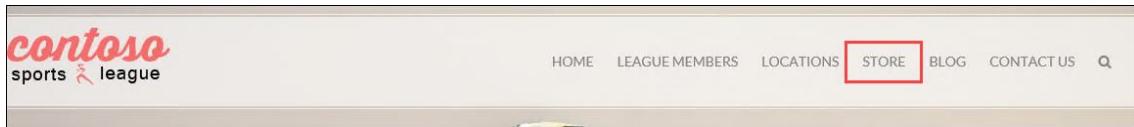
9. On the **App Service** blade, click on **Overview**.

The screenshot shows the 'App Service' blade for the 'contososportsweb4' app service. It has a search bar and a sidebar with links: 'Overview' (highlighted with a red box), 'Activity log', 'Save', 'Debugging', 'Remote de...', and 'Remote Vis...'.

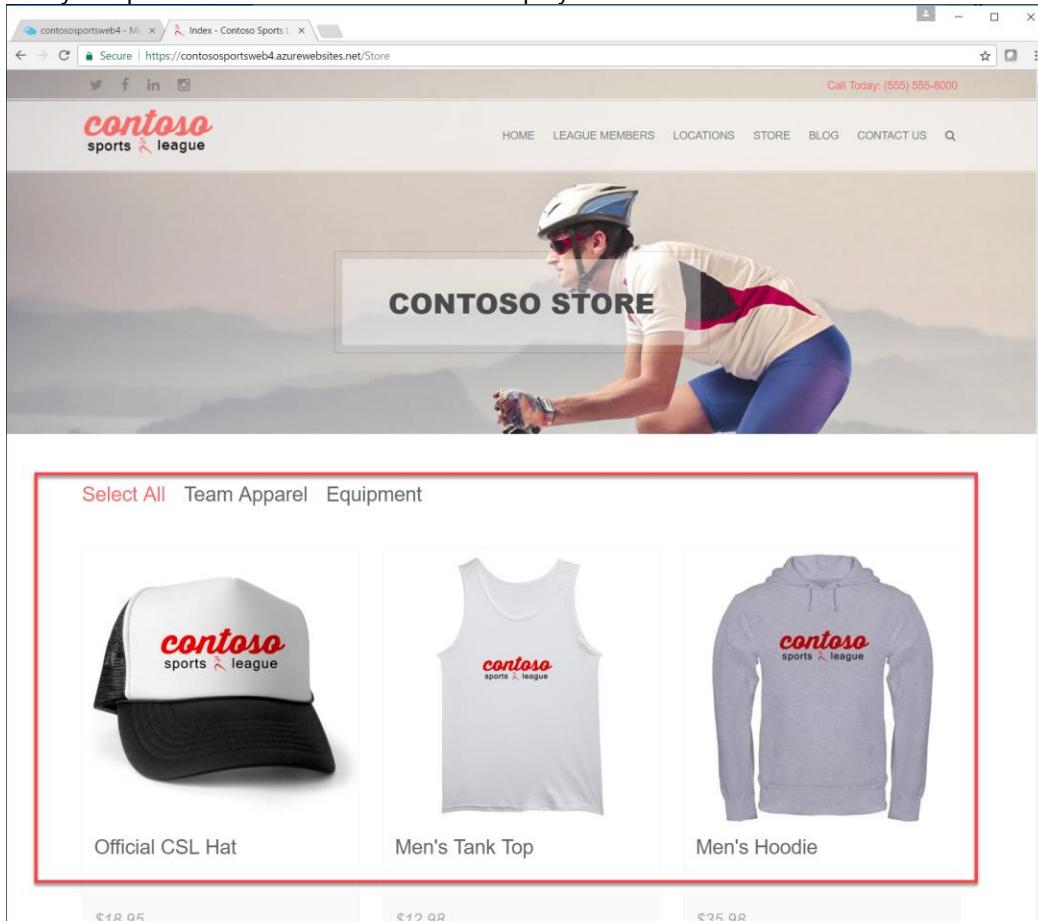
10. On the **Overview** pane, click on the **URL** for the Web App to open it in a new browser tab.

The screenshot shows the 'Overview' pane for the 'contososportsweb4' app service. It has a search bar and a table with columns: 'Resource group', 'Status', 'Location', and 'URL'. The 'URL' row contains the value 'http://contososportsweb4.azurewebsites.net' (highlighted with a red box).

11. After the e-commerce Web App loads in Internet Explorer, click on **STORE** in the top navigation bar of the website.

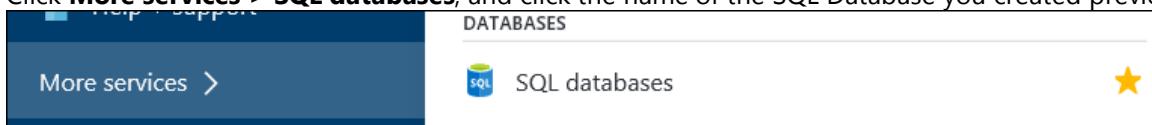


12. Verify the product list from the database displays.

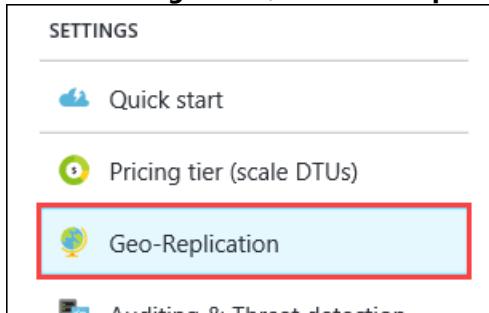


Subtask 4: Revert Failover back to Primary database

1. Using a new tab or instance of your browser, navigate to the Azure Management Portal <http://portal.azure.com>.
2. Click **More services > SQL databases**, and click the name of the SQL Database you created previously.



3. On the **Settings** blade, click **Geo-Replication**.

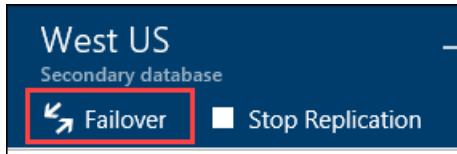


4. On the **Geo-Replication** blade, select the Secondary database.

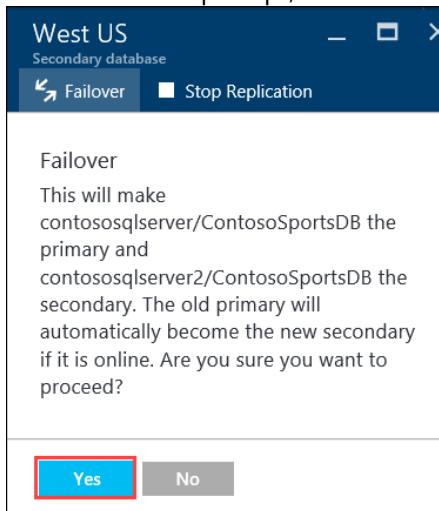
The screenshot shows the 'Geo-Replication' blade for the 'contososqlserver/ContosoSportsDB' database. It features a world map with various regions marked by green circles. Below the map, the 'PRIMARY' section shows 'East US' as the primary server. The 'SECONDARIES' section shows 'West US' as a secondary server, which is highlighted with a red box. The table below lists the server details:

SERVER/DATABASE	STATUS
PRIMARY East US contososqlserver2/ContosoSp...	Online
SECONDARIES West US contososqlserver/ContosoSp... Readable	...

5. Click the **Failover** button.



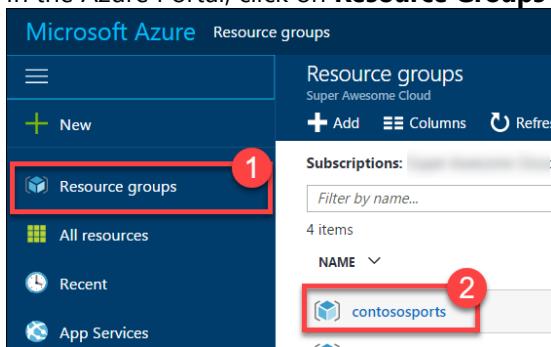
6. On the **Failover** prompt, click **Yes**.



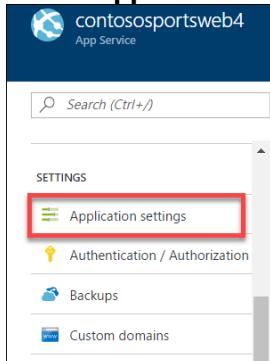
The Failover may take a few minutes to complete. You can continue with the next Subtask modifying the Web App to point back to the Primary SQL Database while the Failover is pending.

Subtask 5: Test e-commerce Web App after reverting Failover

1. In the Azure Portal, click on **Resource Groups > contososports** resource group.



2. Click on the **Web App** just created in a previous step.
3. On the **App Service** blade, scroll down in the left pane, and click on **Application settings**.



4. Scroll down, and locate the **Connection strings** section.

5. Update the **ContosoSportsLeague** Connection String to the value of the Connection String for the **Primary SQL Database**.

Name	Value	Type	Slot setting	...
ContosoSportsLe...	Server=tcp:cont...	SQL Database	<input type="checkbox"/>	...
		SQL Database	<input type="checkbox"/>	...

Ensure you replace the string placeholder values **{your_username}** **{your_password_here}** with the username and password you respectively setup during creation (demouser AND demo@pass123).

;Password={your_password_here};

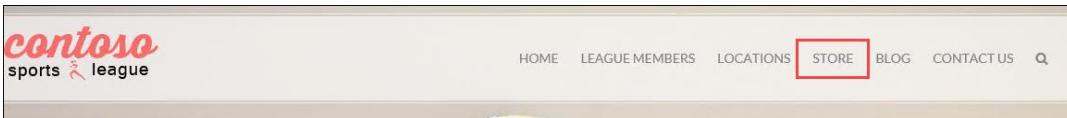
6. Click **Save**.

7. On the **App Service** blade, click on **Overview**.

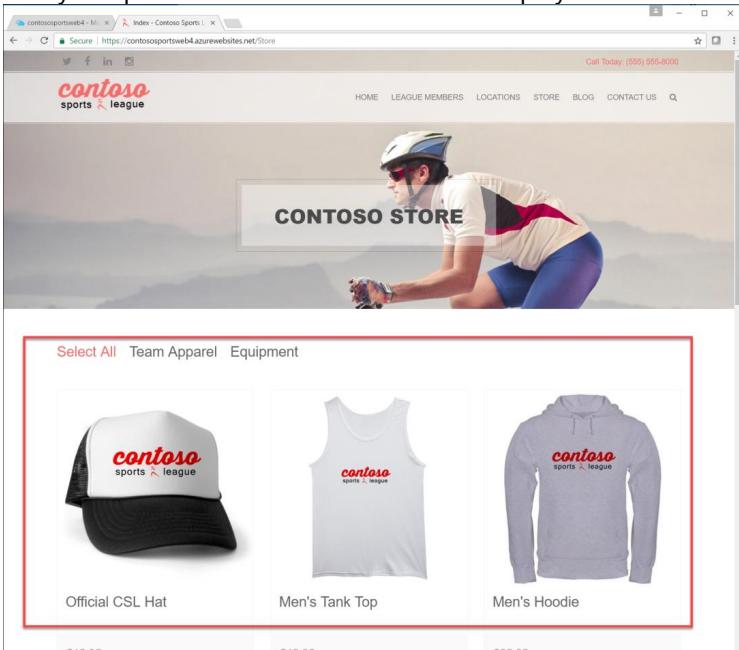
8. On the **Overview** pane, click on the **URL** for the Web App to open it in a new browser tab.

Resource group contososports	URL http://contososportsweb4.azurewebsites.net
Status Running	App Service plan/pricing tier ContosoSportsPlan (Standard: 1 Small)
Location	FTP/deployment username

9. After the e-commerce Web App loads in Internet Explorer, click on **STORE** in the top navigation bar of the website.



10. Verify the product list from the database displays.

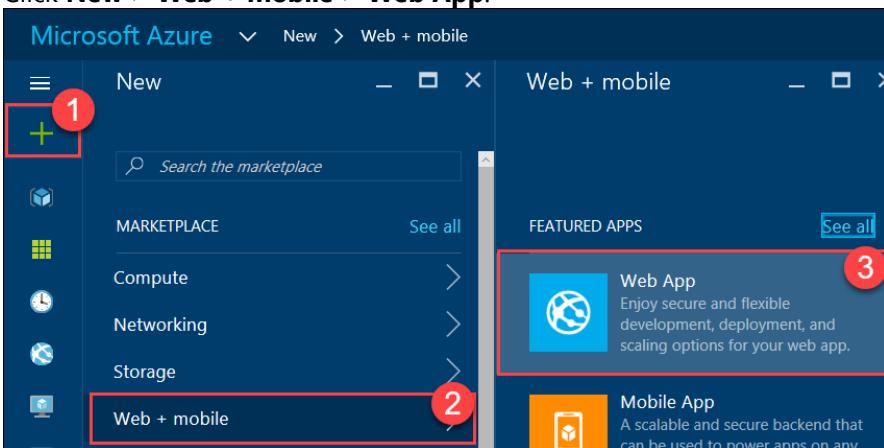


Task 3: Deploying the call center admin website

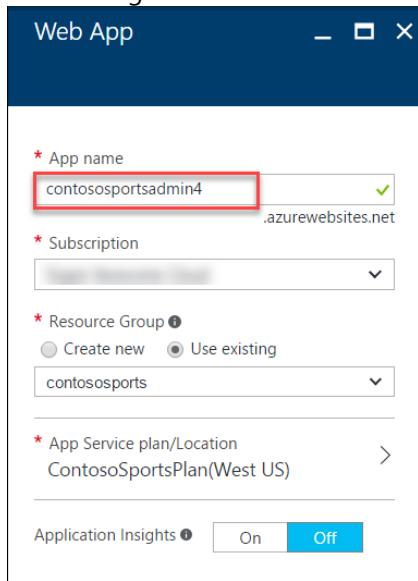
In this exercise, you will provision a website via the Azure Web App template using the Microsoft Azure Portal. You will then edit the necessary configuration files in the Starter Project and deploy the call center admin website.

Subtask 1: Provision the call center admin Web App

1. Using a new tab or instance of your browser, navigate to the Azure Management portal <http://portal.azure.com>.
2. Click **New > Web + mobile > Web App**.



3. Specify a **unique URL** for the Web App, and ensure the **same App Service Plan** and **resource group** you have used throughout the lab are selected.



4. Click on **App Service plan/Location**, and select the **ContosoSportsPlan** used by the front-end Web App.
 5. After the values are accepted, click **Create**.



Subtask 2: Update the configuration in the starter project

1. Navigate to the **App Service** blade for the Call Center Admin App just provisioned.

Setting	Value
Resource group	contosports
Status	Running
Location	West US
Subscription name	
Subscription ID	
URL	http://contosportsadmin4.azurewebsites....
App Service plan/pricing tier	ContosoSportsPlan (Standard: 1 Small)
FTP/deployment username	contosportsadmin4\demouser42
FTP hostname	ftp://waws-prod-bay-071.ftp.azurewebsites...
FTPS hostname	ftps://waws-prod-bay-071.ftp.azurewebsite...

2. On the **App Service** blade, click on **Application settings** in the left pane.

- SETTINGS
- Application settings
- Authentication / Authorization

3. Scroll down, and locate the **Connection strings** section.

The screenshot shows the 'Application settings' blade for an Azure App Service named 'contososportsadmin4'. On the left, there's a sidebar with 'SETTINGS' expanded, showing 'Application settings' selected. The main area has a heading 'Connection strings' with a sub-section 'No results'. Below this are two input fields: 'Name' and 'Value'. A dropdown menu shows 'SQL Database' selected. There are also 'Slot setting' and '...' buttons.

4. Add a new **Connection string** with the following values:
 - a. Name: **ContosoSportsLeague**
 - b. Value: **enter the Connection String for the SQL Database that was created**
 - c. Type: **SQL Database**

A modal dialog titled 'Connection strings' is shown. It contains a table with one row. The first column is 'Name' with value 'ContosoSportsLeague', the second is 'Value' with value 'Server=tcp:co...', and the third is a dropdown set to 'SQL Database'. Below the table are two empty input fields for 'Name' and 'Value', and another dropdown set to 'SQL Database'. At the bottom right are 'Slot setting' and '...' buttons.

Ensure you replace the string placeholder values **{your_username}** **{your_password_here}** with the username and password you respectively setup during creation (demouser AND demo@pass123).

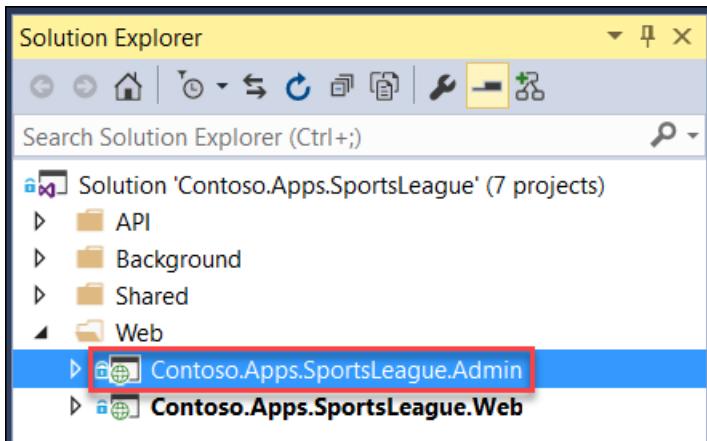
;Password={your_password_here};

5. Click **Save**.

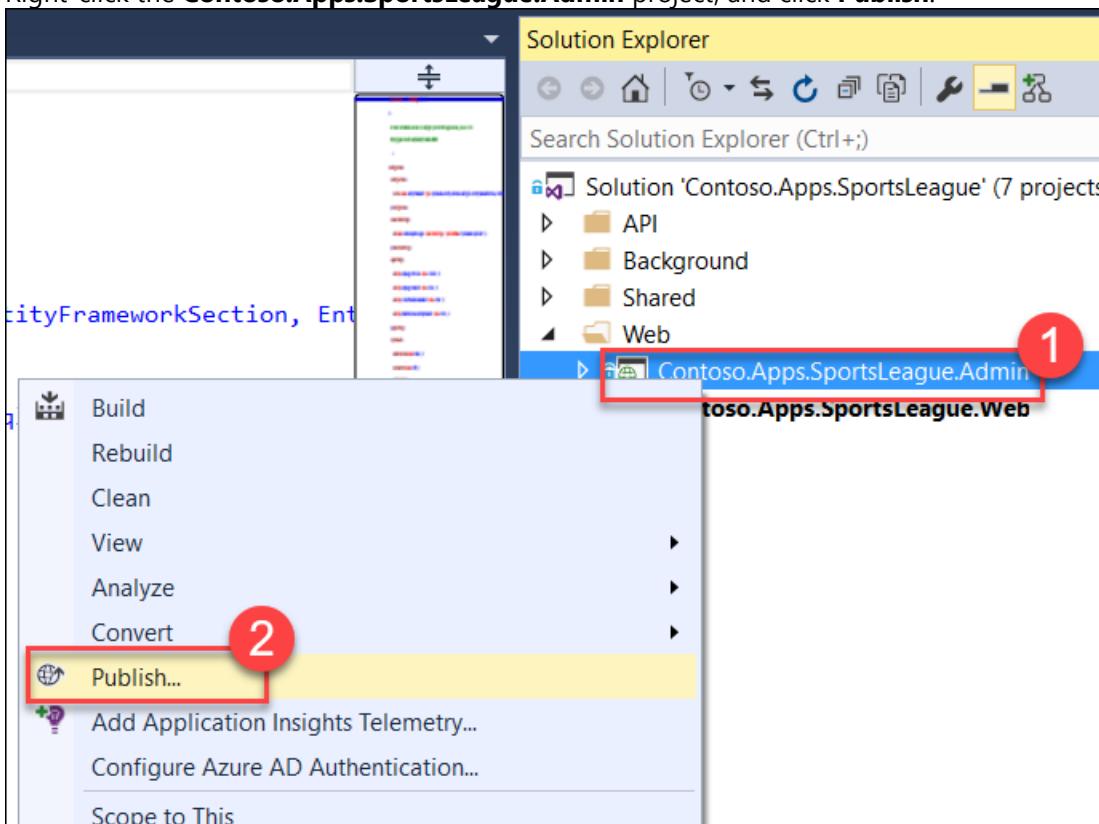
The screenshot shows the 'Application settings' blade again. The 'Save' button at the top right is highlighted with a red box. In the main area, under 'WEBSITE_NODE_DEFAULT_V...', there's a table with 'Key' and 'Value' columns. Below this is a section for 'Connection strings' containing a table with a single row: 'ContosoSportsLeague' and 'Server=tcp:co...'. The 'Application settings' item in the sidebar is now highlighted.

Subtask 3: Deploy the call center admin Web App from Visual Studio

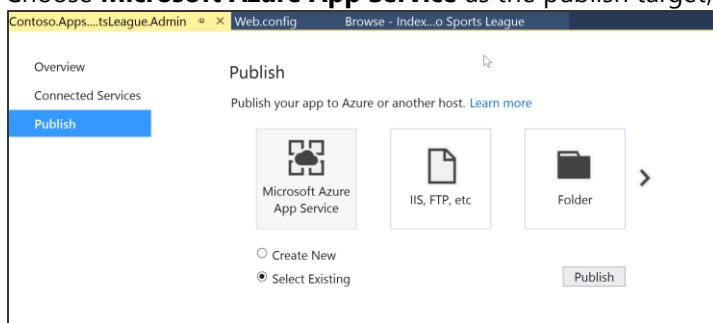
1. Navigate to the **Contoso.Apps.SportsLeague.Admin** project located in the **Web** folder using the **Solution Explorer** in Visual Studio.



2. Right-click the **Contoso.Apps.SportsLeague.Admin** project, and click **Publish**.



3. Choose **Microsoft Azure App Service** as the publish target, and choose **Select Existing**.



4. Select the **Web App** for the Call Center Admin App.

The screenshot shows the 'App Service' blade in the Azure portal. At the top, it says 'Host your web and mobile applications,'. Below that is a 'Subscription' dropdown set to 'Super Awesome Cloud'. Under 'View', there's a 'Resource Group' dropdown set to 'Resource Group'. A 'Search' input field is present. In the main area, there's a tree view of resources under 'contosports'. The 'contosportsadmin4' item is highlighted with a red box. Other items in the tree include 'contosportsweb4' and several other icons.

5. Click **OK**, and click **Publish** to deploy the site.
6. The website should load / display the following:

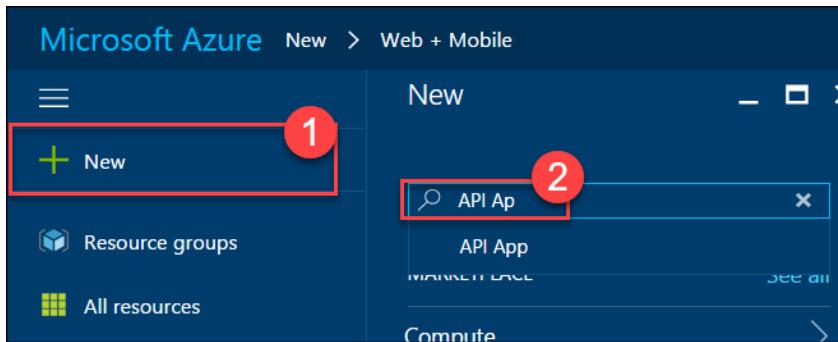
The screenshot shows the 'Contoso Sports League Admin' website. The header includes a logo, 'Home', 'About', and 'Sign in'. The main content area has a heading 'Contoso Sports League Admin' and a paragraph: 'Below is a list of orders, sorted by reverse order date. You may click on an order to see its details and to obtain the generated PDF receipt.' A 'Learn more »' button is visible. Below this is a section titled 'Completed Orders' with the message 'There is no data available.' At the bottom, a copyright notice reads '© 2015 - Contoso Sports League Admin'.

Task 4: Deploying the payment gateway

In this exercise, the attendee will provision an Azure API app template using the Microsoft Azure Portal. The attendee will then deploy the payment gateway API to the API app.

Subtask 1: Provision the payment gateway API app

1. Using a new tab or instance of your browser, navigate to the Azure Management Portal <http://portal.azure.com>.
2. Click **+New**, type **API App** into the Search the marketplace box, and press **Enter**.



3. Click on **API App** in the search results list.

NAME	PUBLISHER	CATEGORY
API App	Microsoft	Web + Mobile
Cognitive Services APIs (preview)	Microsoft	Intelligence + analytics

4. Click on **Create**.

API App
Microsoft

Create and deploy RESTful APIs in seconds, as powerful as you need them

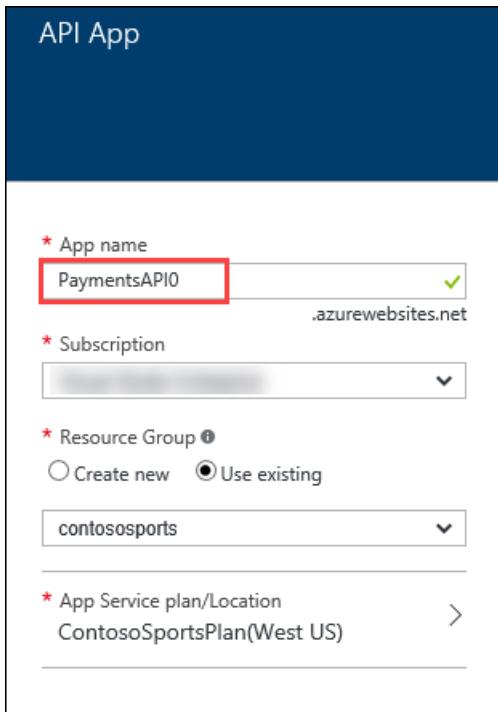
Leverage your existing tools to create and deploy RESTful APIs without the hassle of managing infrastructure. Microsoft Azure App Service API Apps offers secure and flexible development, deployment, and scaling options for any sized RESTful API application. Use frameworks and templates to create RESTful APIs in seconds. Choose from source control options like TFS, GitHub, and BitBucket. Use any tool or OS to develop your RESTful API with .NET, Java, PHP, Node.js or Python.

- Fastest way to build for the cloud
- Provision and deploy fast
- Simple access control and authentication
- Secure platform that scales automatically
- Great experience for Visual Studio developers with automatic SDK generation
- Open and flexible for everyone
- Monitor, alert, and auto scale (preview)

[Documentation](#) [Solution Overview](#) [Pricing Details](#)

Create

5. On the new **API App** blade, specify a unique name for the App Name, and ensure the previously used Resource Group and App Service Plan are selected.



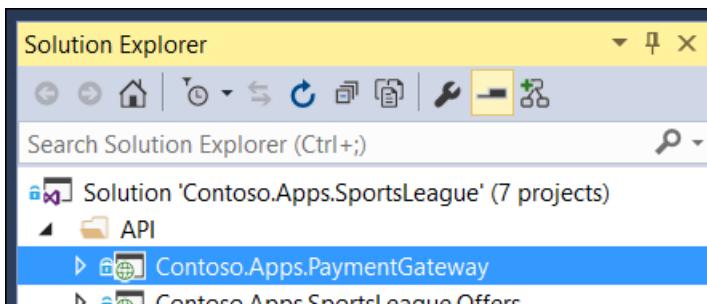
6. Click on **App Service plan/Location**, and select the same App Service Plan used for the other Web App services.

7. After the values are accepted, click **Create**.

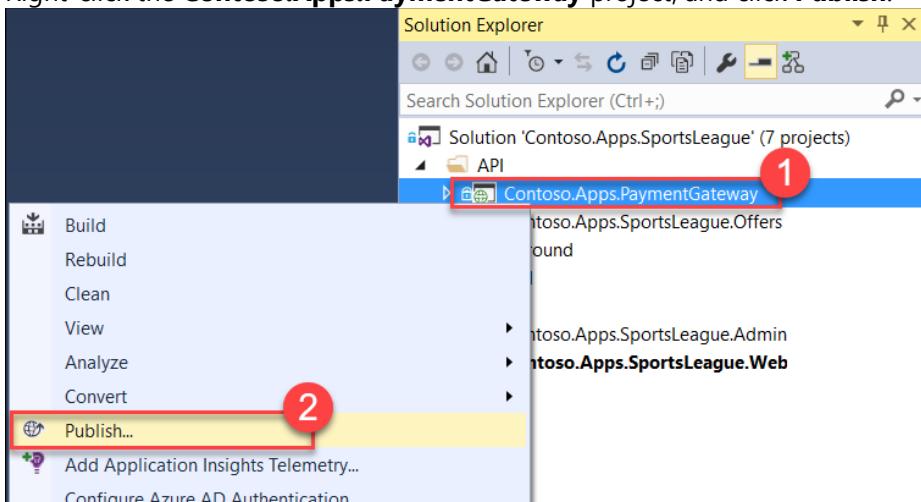


Subtask 2: Deploy the Contoso.Apps.PaymentGateway project in Visual Studio

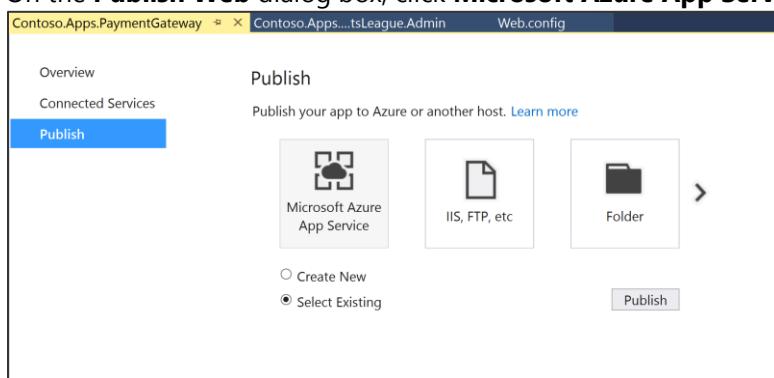
1. Navigate to the **Contoso.Apps.PaymentGateway** project located in the **APIs** folder using the **Solution Explorer** in Visual Studio.



2. Right-click the **Contoso.Apps.PaymentGateway** project, and click **Publish**.



3. On the **Publish Web** dialog box, click **Microsoft Azure App Service**, and choose **Select Existing**.



4. Select the Payment Gateway API app created earlier, click **OK > Publish**.

App Service

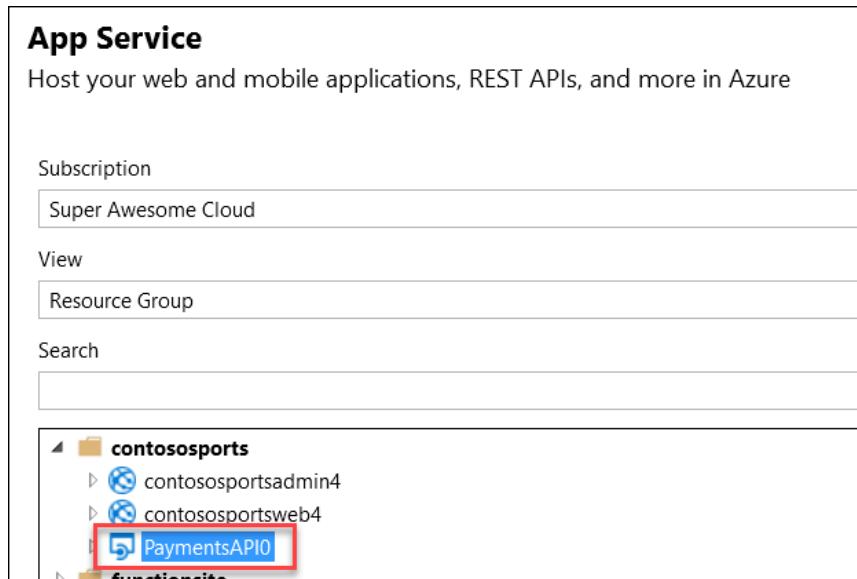
Host your web and mobile applications, REST APIs, and more in Azure

Subscription
Super Awesome Cloud

View
Resource Group

Search

contosospports
contosospportsadmin4
contososportsweb4
PaymentsAPI0



5. In the Visual Studio **Output** view, you will see a status indicating the Web App was published successfully.

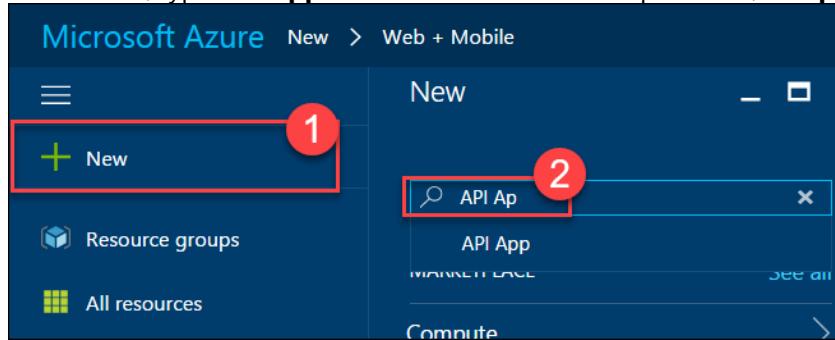
```
2>Publish Succeeded.  
2>Web App was published successfully http://paymentsapi0.azurewebsites.net/  
===== Build: 1 succeeded, 0 failed, 1 up-to-date, 0 skipped =====
```
6. Record the value of the deployed **API App URL** for later use.

Task 5: Deploying the offers Web API

In this exercise, the attendee will provision an Azure API app template using the Microsoft Azure Portal. The attendee will then deploy the offers Web API.

Subtask 1: Provision the offers Web API app

1. Using a new tab or instance of your browser, navigate to the Azure Management Portal (<http://portal.azure.com>).
2. Click **+New**, type **API App** into the Search the marketplace box, and **press Enter**.



3. Click on **API App** in the search results list.

The screenshot shows a search results page with a filter bar at the top. A search bar contains the text 'API App'. Below it, a table lists two items:

NAME	PUBLISHER	CATEGORY
API App	Microsoft	Web + Mobile
Cognitive Services APIs (preview)	Microsoft	Intelligence + analytics

4. Click on **Create**.

The screenshot shows a product page for 'API App' by Microsoft. It includes a brief description, a list of benefits, social sharing links, publisher information, useful links, and a prominent 'Create' button.

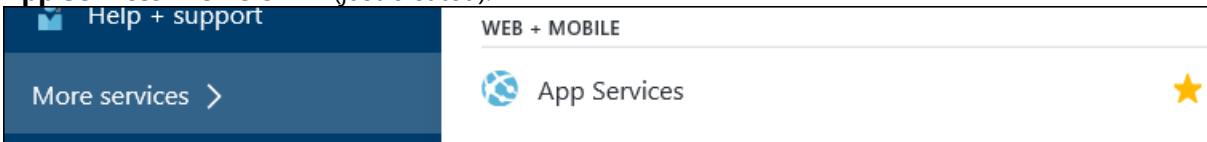
5. On the new **API App** blade, specify a unique name for the App Service Name, and ensure the previously used Resource Group and App Service Plan are selected.

The screenshot shows the 'API App' creation blade. The 'App name' field is filled with 'OffersAPI4'. Other fields include 'Subscription' (selected), 'Resource Group' (set to 'contososports'), 'App Service plan/Location' (set to 'ContosoSportsPlan(West US)'), and 'Application Insights' (set to 'Off').

6. After the values are accepted, click **Create**.

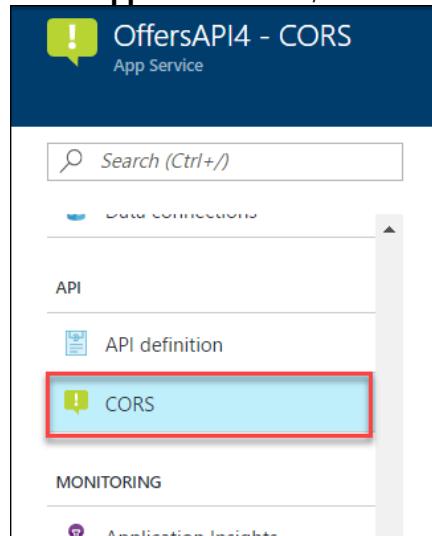


7. When the Web App template has completed provisioning, open the new API App by clicking **More services > App Services > Offers API** (just created).



Subtask 2: Configure cross-origin resource sharing (CORS)

1. On the **App Service** blade, scroll down, and click on **CORS** within the API section of the left pane.



2. In the **ALLOWED ORIGINS** text box, specify *, and click **Save**.

Subtask 3: Update the configuration in the starter project

1. On the **App Service** blade for the Offers API, click on **Application settings**

The screenshot shows the Azure portal interface for an App Service named 'OffersAPI4'. The left sidebar has a 'SETTINGS' section with three items: 'Application settings' (highlighted with a red box), 'Authentication / Authorization', and 'Backups'. A search bar at the top is labeled 'Search (Ctrl+ /)'. A 'Continuous Delivery (Preview)' section is also visible.

2. Scroll down, and locate the **Connection strings** section.

The screenshot shows the 'Connection strings' section in the Azure portal. It displays a table with the following columns: 'Name', 'Value', 'SQL Database', 'Slot setting', and an ellipsis button. The 'Name' column currently contains 'Name' and 'Value' placeholders.

3. Add a new **Connection string** with the following values:

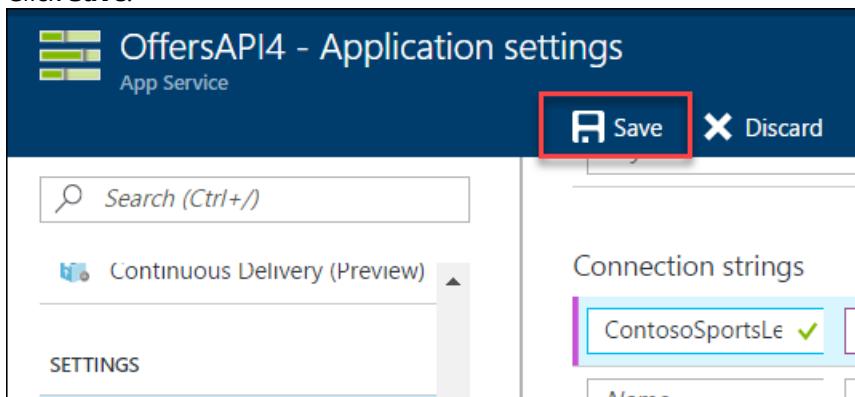
- a. Name: **ContosoSportsLeague**
- b. Value: **enter the Connection String for the SQL Database that was created**
- c. Type: **SQL Database**

The screenshot shows the 'Connection strings' section with a new entry added. The 'Name' field is populated with 'ContosoSportsLeague', the 'Value' field contains a placeholder connection string, and the 'Type' dropdown is set to 'SQL Database'.

Ensure you replace the string placeholder values **{your_username}** **{your_password_here}** with the username and password you respectively setup during creation (demouser AND demo@pass123).

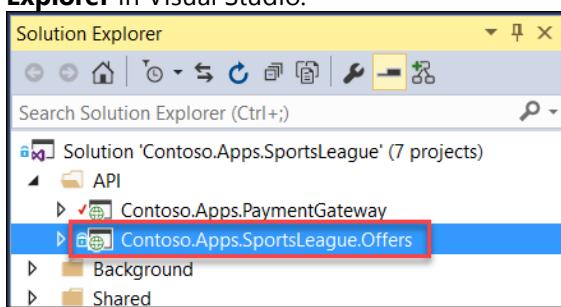
;Password={your_password_here};

4. Click **Save**.

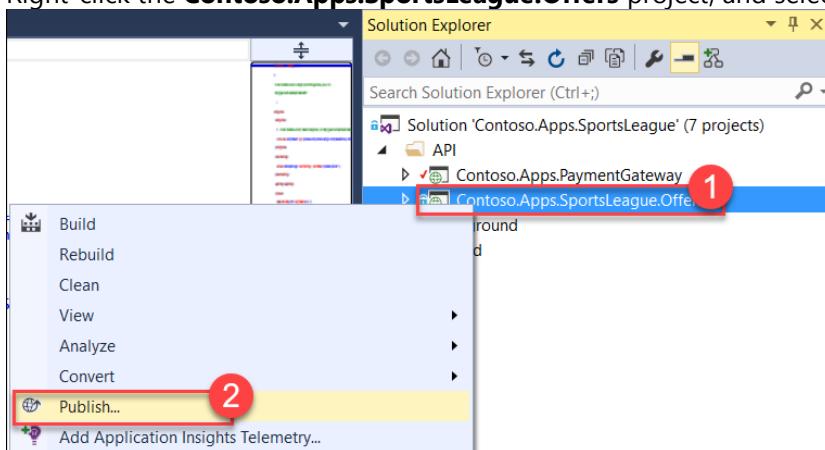


Subtask 4: Deploy the Contoso.Apps.SportsLeague.Offers project in Visual Studio

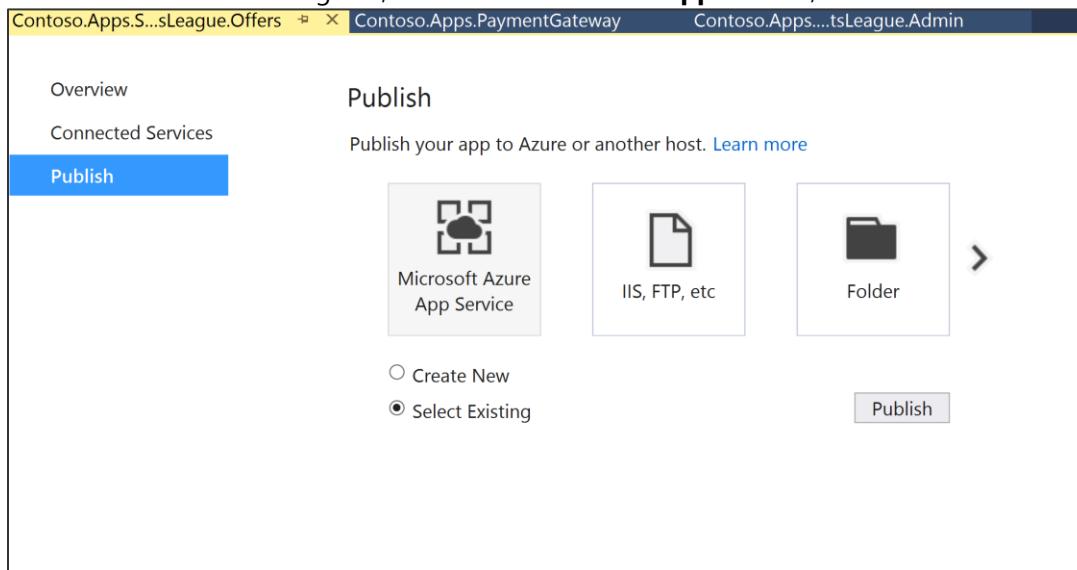
1. Navigate to the **Contoso.Apps.SportsLeague.Offers** project located in the **APIs** folder using the **Solution Explorer** in Visual Studio.



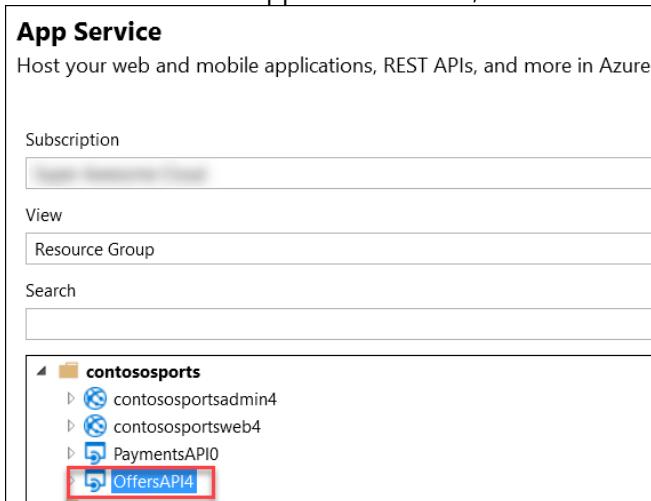
2. Right-click the **Contoso.Apps.SportsLeague.Offers** project, and select **Publish**.



3. On the **Publish Web** dialog box, click **Microsoft Azure App Service**, and choose **Select Existing**.



4. Select the Offers API app created earlier, and click **OK > Publish**.



5. In the Visual Studio **Output** view, you will see a status the API app was published successfully.

6. Record the value of the deployed API app URL for later use.

```
2>Publish Succeeded.  
2>Web App was published successfully: http://offersapi4.azurewebsites.net/
```

Task 6: Update and deploy the e-commerce website

Subtask 1: Update the Application Settings for the Web App that hosts the Contoso.Apps.SportsLeague.Web project

1. Using a new tab or instance of your browser, navigate to the Azure Management Portal <http://portal.azure.com>.

2. Click on **Resource groups > contososports** resource group.

The screenshot shows the Microsoft Azure Resource Groups blade. On the left sidebar, the 'Resource groups' item is highlighted with a red box and a red number '1'. In the main pane, a list of resource groups is displayed, with 'contososports' highlighted with a red box and a red number '2'.

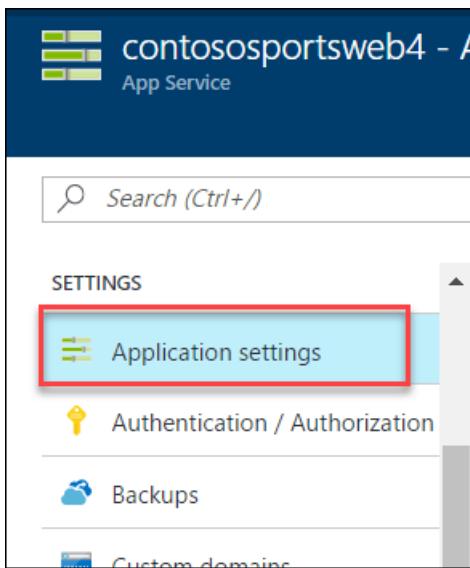
NAME
contososports

3. Click on the **App Service Web App** for the front-end Web Application.

The screenshot shows the Microsoft Azure Resource Group blade for the 'contososports' resource group. The left sidebar has 'Overview' selected. In the main pane, under the 'Essentials' section, there is a list of resources. The 'contososportsweb4' app service is highlighted with a red box and a red number '2'.

NAME
contososports01
contososportsadmin4
contososportsdbservice1234
ContosoSportsDB
ContosoSportsPlan
contososportsweb4
OffersAPI4
PaymentsAPI0

4. On the **App Service** blade, scroll down, and click on **Application settings** in the left pane.



5. Scroll down, and locate the **App settings** section.

A screenshot of the 'App settings' section in the Azure portal. It shows two existing app settings: 'WEBSITE_NODE_DEFAULT_V...' with value '6.9.1' and 'AzureQueueConnectionString' with value 'DefaultEndpointsProtocol=...'. Below these is a table with columns 'Key' and 'Value'. A new row is being added, with 'paymentsAPIUrl' in the 'Key' column and 'https://paymentsapi0.azurewebsites.net/api/nvp' in the 'Value' column. There are checkboxes for 'Slot setting' and a '...' button next to each row.

6. Add a new **App Setting** with the following values:

- Key: **paymentsAPIUrl**
- Value: enter the **HTTPS URL** for the Payments API App with **/api/nvp** appended to the end. Ex: <https://paymentsapi0.azurewebsites.net/api/nvp>

A screenshot of the 'App settings' section in the Azure portal. It shows the same two existing app settings as before. A new row is being added, with 'paymentsAPIUrl' in the 'Key' column and 'https://paymentsapi0.azurewebsites.net/api/nvp' in the 'Value' column. Both columns have a blue border, indicating they are selected or being edited. There are checkboxes for 'Slot setting' and a '...' button next to each row.

7. Add a new **App Setting** with the following values:

- Key: **offersAPIUrl**
- Value: enter the **HTTPS URL** for the Offers API App with **/api/get** appended to the end. Ex: <https://offersapi4.azurewebsites.net/api/get>

The screenshot shows the 'App settings' section of the Azure portal. It lists several environment variables:

- WEBSITE_NODE_DEFAULT_V... 6.9.1
- AzureQueueConnectionString DefaultEndpointsProtocol=...
- paymentsAPIUrl https://paymentsapi0.azurewebsites.net/api/get
- offersAPIUrl** https://contososportsweb4.azurewebsites.net/api/get

The 'offersAPIUrl' row is highlighted with a red box. Below the table, there are 'Key' and 'Value' input fields, both of which are currently empty.

- Click on **Save**.

The screenshot shows the 'Application settings' blade for the app service 'contososportsweb4'. The left sidebar shows 'SETTINGS' with 'Application settings' selected. The main area displays the following application settings:

- WEBSITE_NODE_DEFAULT_V... 6.9.1
- AzureQueueConnectionString Defa...
- paymentsAPIUrl https://...
- offersAPIUrl** https://...

The 'offersAPIUrl' row is highlighted with a red box. At the top right, there are 'Save' and 'Discard' buttons, with 'Save' being highlighted by a red box.

Note: Ensure both of the API URLs are using **SSL** (<https://>), or you will see a CORS errors.

Subtask 2: Validate App Settings are correct

- On the **App Service** blade, click on **Overview**.

The screenshot shows the 'Overview' blade for the app service 'contososportsweb4'. The left sidebar has 'Overview' selected and is highlighted with a red box. The main area displays the following details:

Essentials	
Resource group	contososports
Status	Running
Location	West US
Subscription name	Super Awesome Cloud
URL	http://contososportsweb4.azurewebsites.net
App Service plan/pricing tier	ContosoSportsPlan (Standard: 1 Small)
FTP/deployment username	contososportsweb4\demouser42
FTP hostname	feurluaue arad bay 071 for ourourshitee

2. In the **Overview** pane, click on the **URL** for the Web App to open it in a new browser tab.

The screenshot shows the Azure portal's 'Overview' page for an App Service named 'contososportsweb4'. On the left is a navigation sidebar with 'Overview' selected. The main panel displays 'Essentials' information: Resource group 'contososports', Status 'Running', Location 'West US', Subscription name 'Contoso Azure Cloud', and URLs. The 'URL' field is highlighted with a red box and contains the value 'http://contososportsweb4.azurewebsites.net'. Other details include 'App Service plan/pricing tier ContosoSportsPlan (Standard: 1 Small)' and 'FTP/deployment username contososportsweb4\demouser42'.

3. On the homepage, you should see the latest offers populated from the Offers API.

The screenshot shows the homepage of the 'contoso sports league' website. At the top, there is a navigation bar with links for HOME, LEAGUE MEMBERS, LOCATIONS, STORE, and BLOG. Below the navigation, a section titled 'Today's Offers' displays three products: 'Baseball Socks' (\$16.99), 'Road Bike' (\$1,250.99), and 'Baseball Mitt' (\$45.50). Each product has a 'PRODUCT DETAILS' button.

4. Submit several test orders to ensure all pieces of the site are functional.

The screenshot shows the 'Checkout/Complete' page of the website. It features a large image of a baseball player in action. A central box displays the message 'ORDER COMPLETED'. Below the image, a green 'Success!' bar contains the text: 'Thank you for your business! Your order has been completed. Here's your transaction id for reference: fNzY5hVwYAF'. At the bottom is a red 'RETURN HOME' button.

Leader Note: If the attendee is still experiencing CORS errors ensure the URLs to the Web App in Azure local host are exact.

Exercise 2: Identity and security

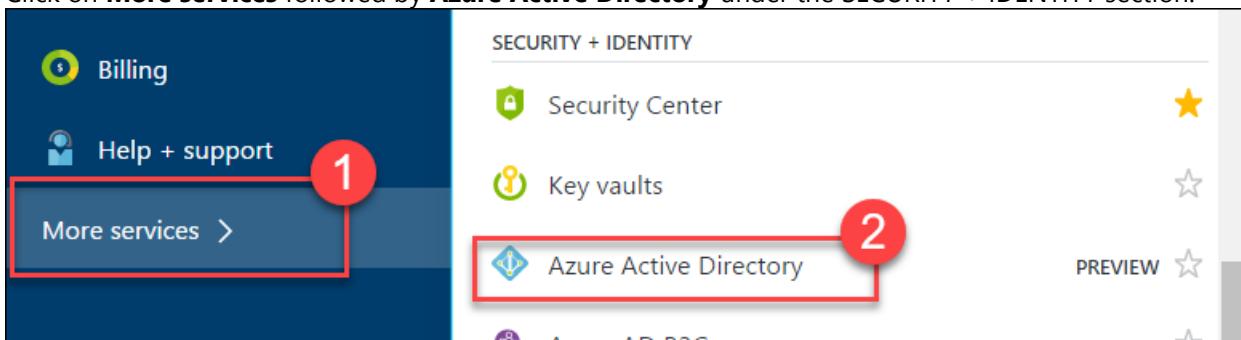
Duration: 75 Minutes

The Contoso call center admin application will only be accessible by users of the Contoso Active Directory environment. You have been asked to create a new Azure AD Tenant and secure the application so only users from the tenant can log on.

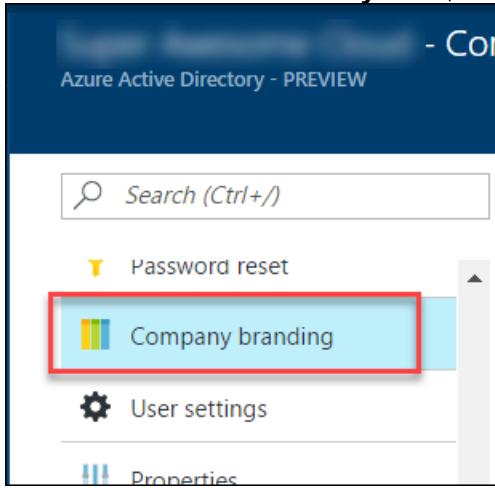
Task 1: Enable Azure AD Premium Trial

Note: this task is **optional**, and it is valid only if you are a global administrator on the Azure AD tenant associated with your subscription.

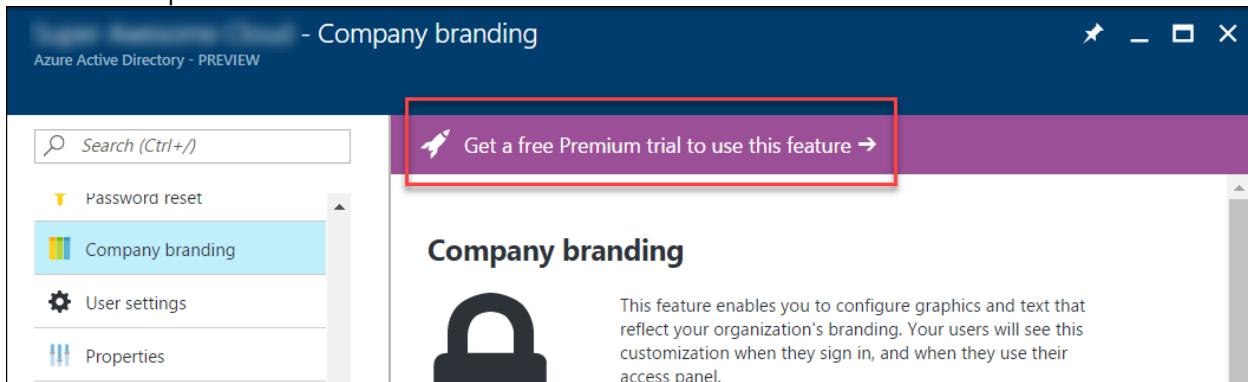
1. Navigate to the Azure Management portal, <http://portal.azure.com>, using a new tab or instance.
2. Click on **More services** followed by **Azure Active Directory** under the SECURITY + IDENTITY section.



3. On the **Azure Active Directory** blade, locate and click on the **Company branding** option.

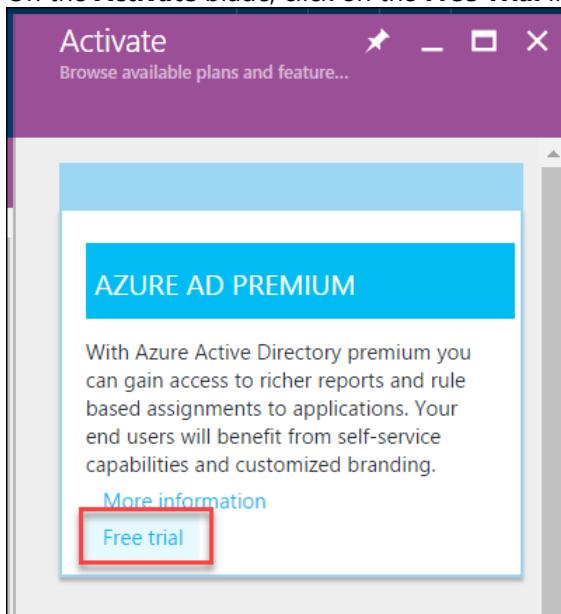


4. Click on the option to **Get a free Premium trial**.

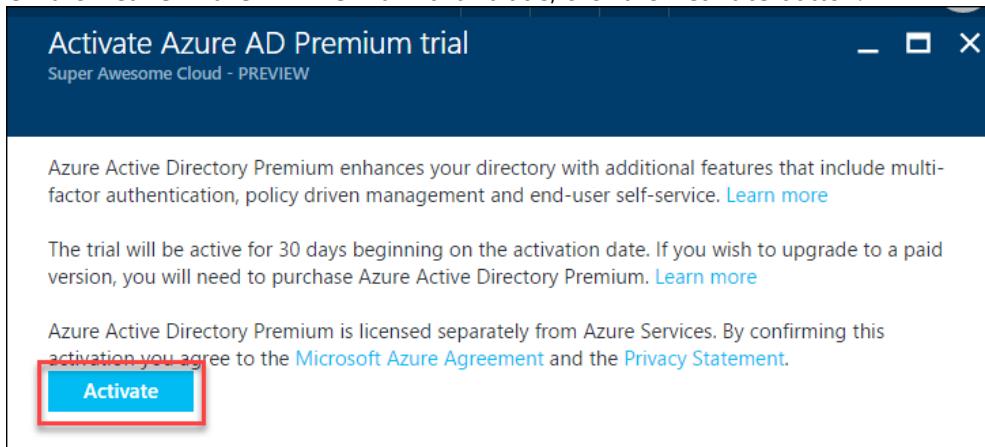


If you already have a Premium Azure Active Directory, skip to Task 2.

5. On the **Activate** blade, click on the **Free Trial** link within the AZURE AD PREMIUM box.



6. On the **Active Azure AD Premium trial** blade, click the **Activate** button.

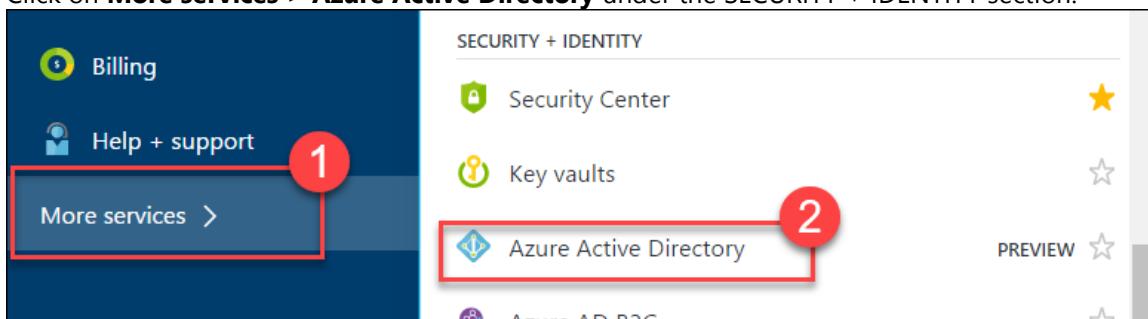


7. Close the **Azure Active Directory** blades.

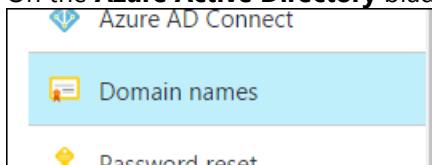
Task 2: Create a new Contoso user

Note: this task is **optional**, and it is valid only if you are a global administrator on the Azure AD tenant associated with your subscription.

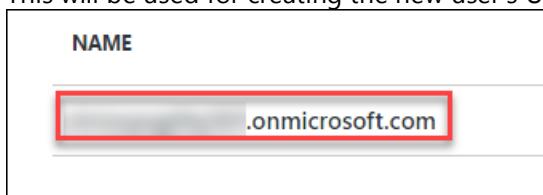
1. Navigate to the Azure Management portal, <http://portal.azure.com>, using a new tab or instance.
2. Click on **More services > Azure Active Directory** under the SECURITY + IDENTITY section.



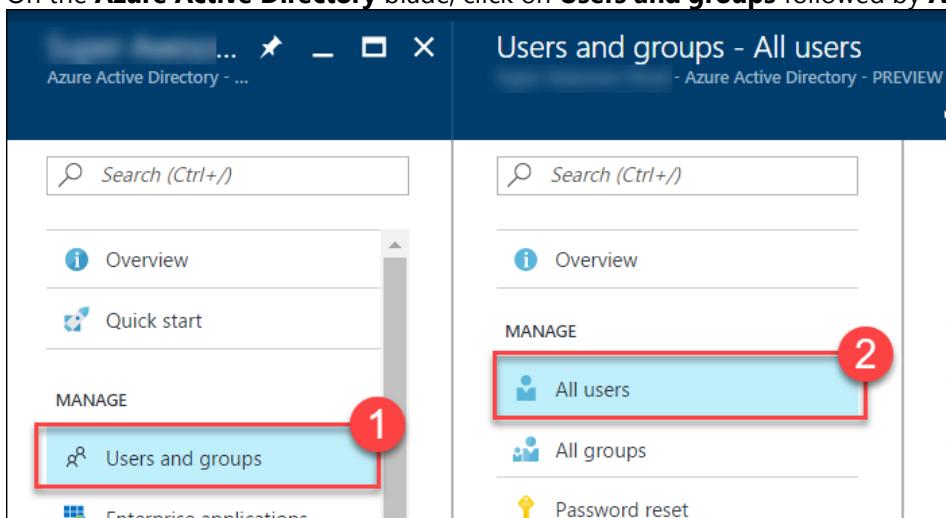
3. On the **Azure Active Directory** blade, click on **Domain names**.



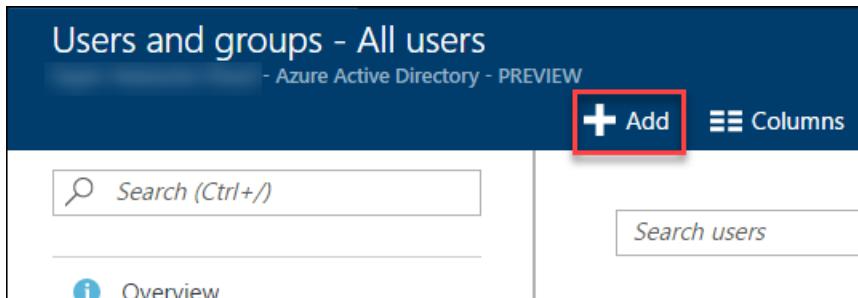
4. Copy the **Domain Name** for your Azure AD Tenant. It will be in the format: *[your tenant].onmicrosoft.com*. This will be used for creating the new user's Username.



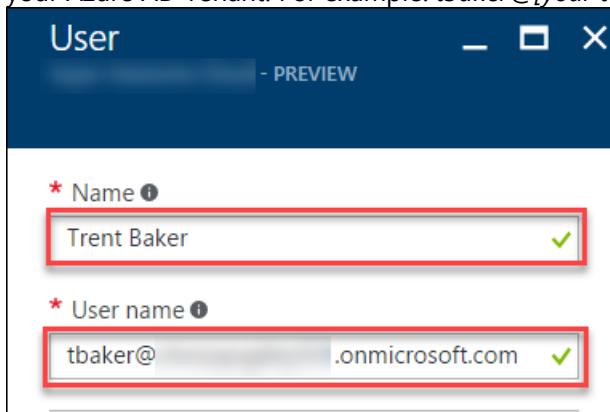
5. On the **Azure Active Directory** blade, click on **Users and groups** followed by **All users**.



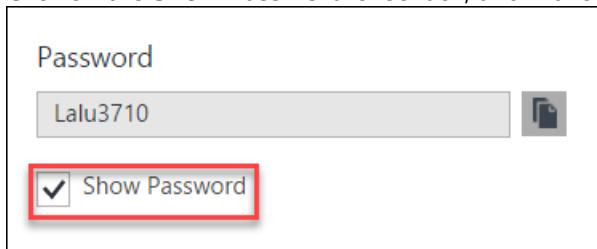
6. Click on **+Add** to add a new user.



7. On the **User** blade, specify a user's **Name** and **Username**. Specify the **Username** to be at the domain name for your Azure AD Tenant. For example: *tbaker@[your tenant].onmicrosoft.com*



8. Click on the **Show Password** checkbox, and make a note of the Password for use later.



9. Click **Create**.



Task 3: Configure access control for the call center administration Web Application

Note: this task is **optional**, and it is valid only if you have the right to create applications in your Azure AD Tenant.

Subtask 1: Enable Azure AD Authentication

1. On the left navigation of the Azure Portal, select **App Services** (or click **More services > App Services**).



2. On the **Web Apps** page, select the **call center administration Web App**.

The screenshot shows the 'App Services' blade in the Azure portal. At the top, there are buttons for 'Add', 'Columns', and 'Refresh'. Below that, a section for 'Subscriptions' is shown with a filter input field. A message indicates there are '6 items'. A dropdown menu is open, showing 'NAME' as the sorting option. A list of app services is displayed, with each item having a small icon and a name. The first item, 'contososportsadmin4', has a red box drawn around it, indicating it is the selected item.

NAME
contososportsadmin4
contososportsweb4
[redacted]
OffersAPI4
PaymentsAPI0
[redacted]

3. Click the **Authentication / Authorization** tile.

The screenshot shows the 'contososportsadmin4' App Service blade. At the top, there's a search bar and a 'Continuous Delivery (Preview)' button. Below that, a 'SETTINGS' section is visible with a list of tiles. One tile, 'Authentication / Authorization', is highlighted with a red box. Other visible tiles include 'Application settings' and 'Backups'.

SETTINGS
Application settings
Authentication / Authorization
Backups

4. Change **App Service Authentication** to **On**, and change the dropdown to **Log in with Azure Active Directory**.

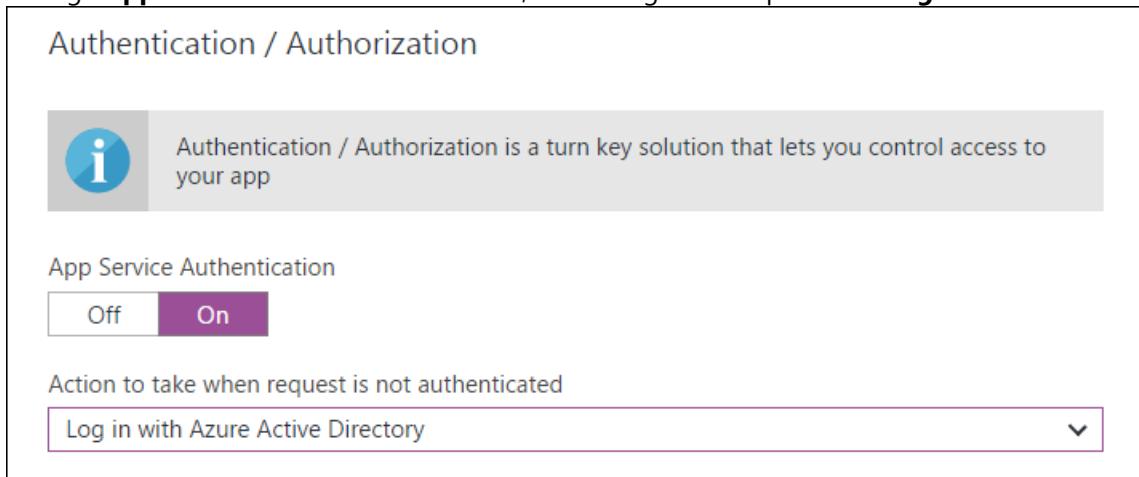
Authentication / Authorization

App Service Authentication

Off **On**

Action to take when request is not authenticated

Log in with Azure Active Directory



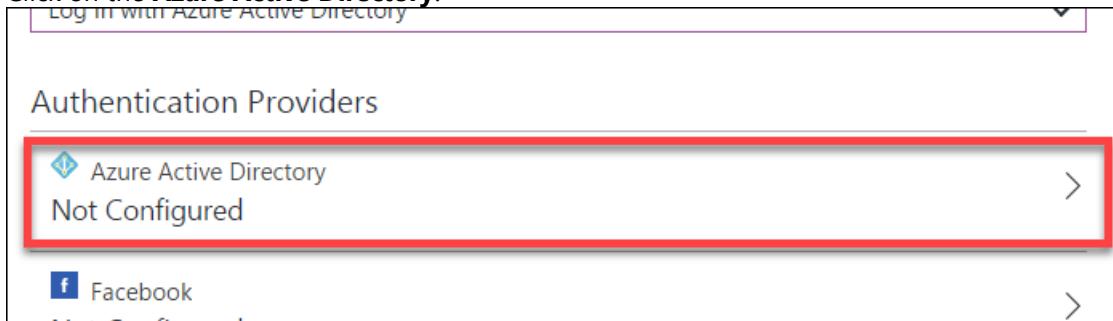
5. Click on the **Azure Active Directory**.

Log in with Azure Active Directory

Authentication Providers

Azure Active Directory
Not Configured

Facebook

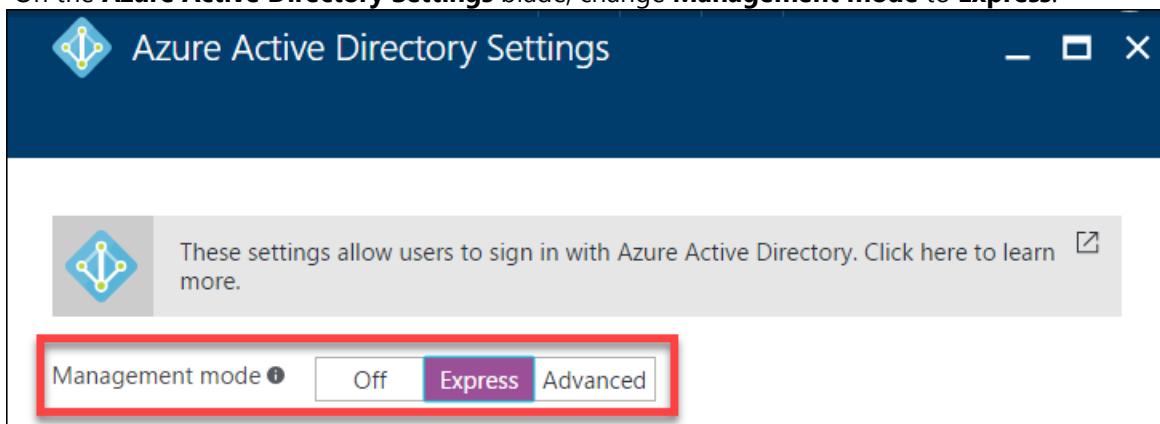


6. On the **Azure Active Directory Settings** blade, change **Management mode** to **Express**.

Azure Active Directory Settings

These settings allow users to sign in with Azure Active Directory. Click here to learn more.

Management mode **Express**



7. Click **OK**.

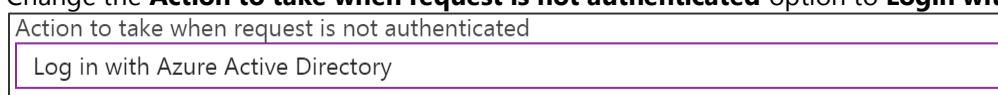
OK



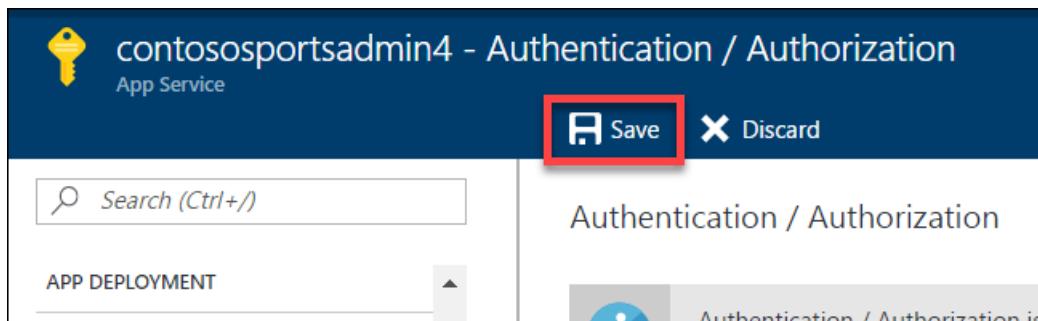
8. Change the **Action to take when request is not authenticated** option to **Login with Azure Active Directory**.

Action to take when request is not authenticated

Log in with Azure Active Directory

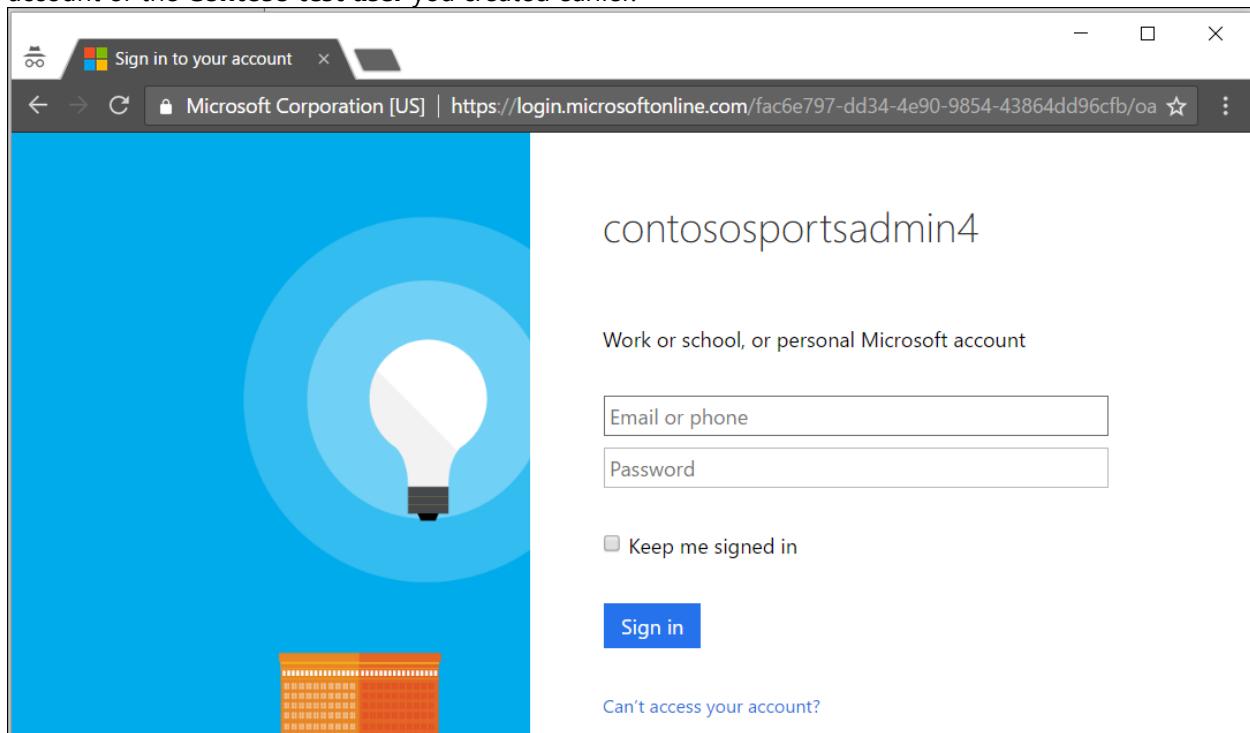


9. In the **Authentication / Authorization** blade, click **Save**.

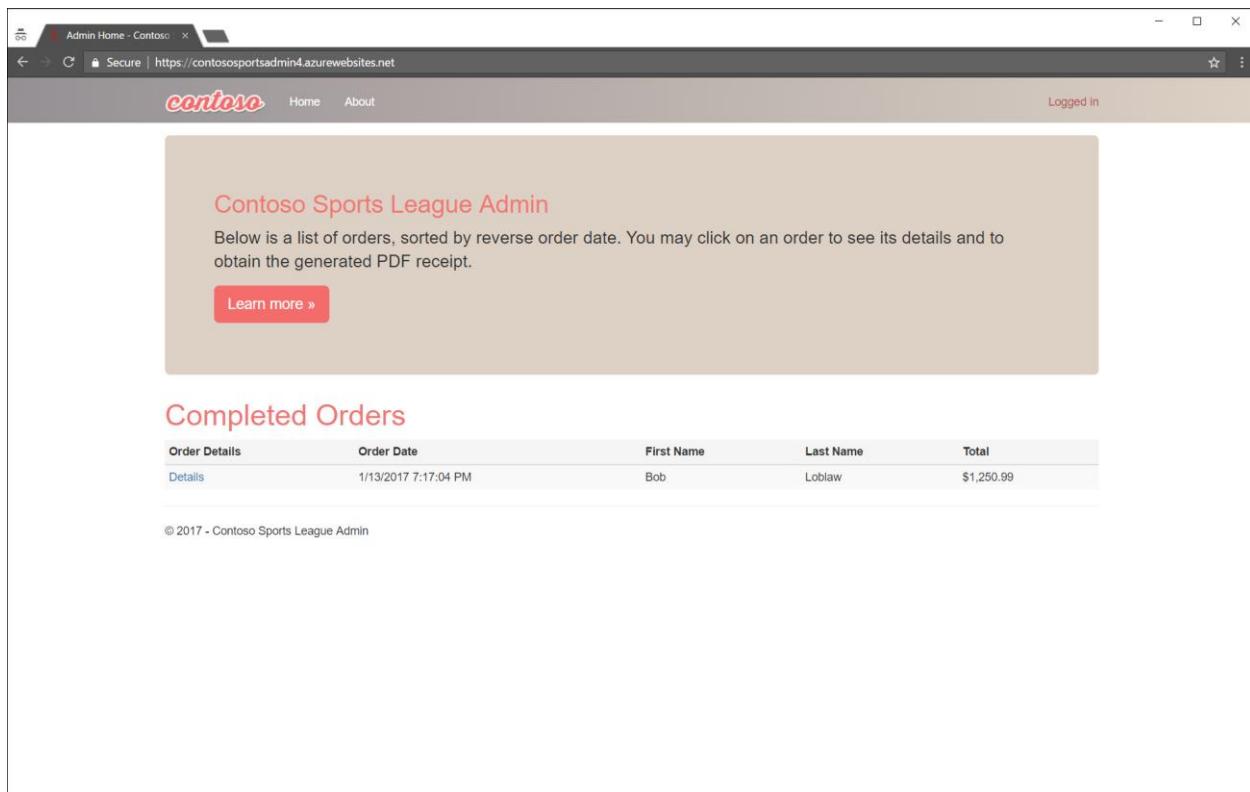


Subtask 2: Verify the call center administration website uses the access control logon

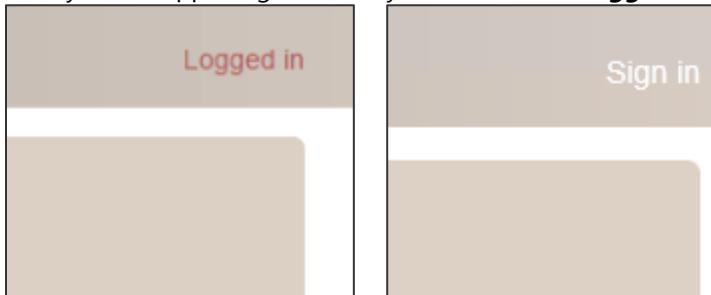
1. Close your browser (or use an alternative), and launch a browser in InPrivate or Incognito mode. Navigate to the **call center administration** website.
2. The browser will redirect to the non-branded Access Control logon URL. You can log on with your Microsoft account or the **Contoso test user** you created earlier.



3. After you log on and **accept the consent**, your browser will be redirected to the Contoso Sports League Admin webpage.



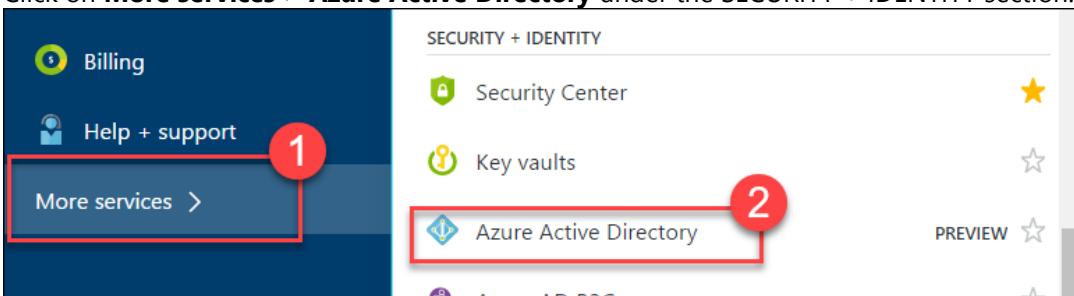
4. Verify in the upper-right corner you see the link **Logged In**. If it is not configured, you will see **Sign in**.



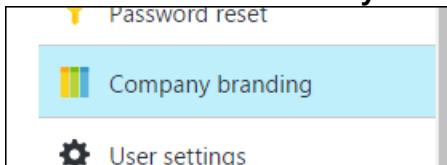
Task 4: Apply custom branding for the Azure Active Directory logon page

Note: this task is **optional**, and it is valid only if you are a global administrator on the Azure AD tenant associated with your subscription, and you completed the Enabling Azure AD Premium exercise.

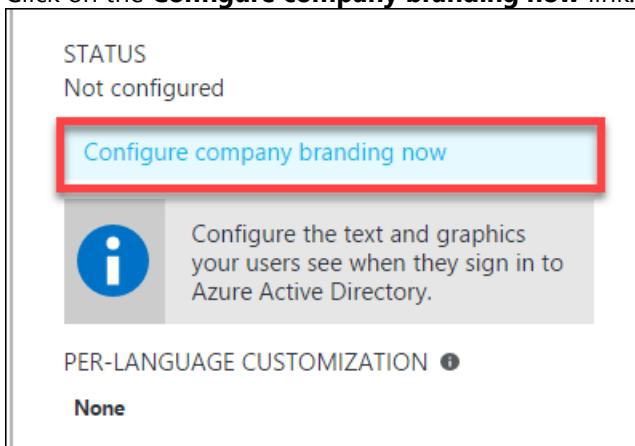
1. Navigate to the Azure Management portal, <http://portal.azure.com>, using a new tab or instance.
2. Click on **More services > Azure Active Directory** under the SECURITY + IDENTITY section.



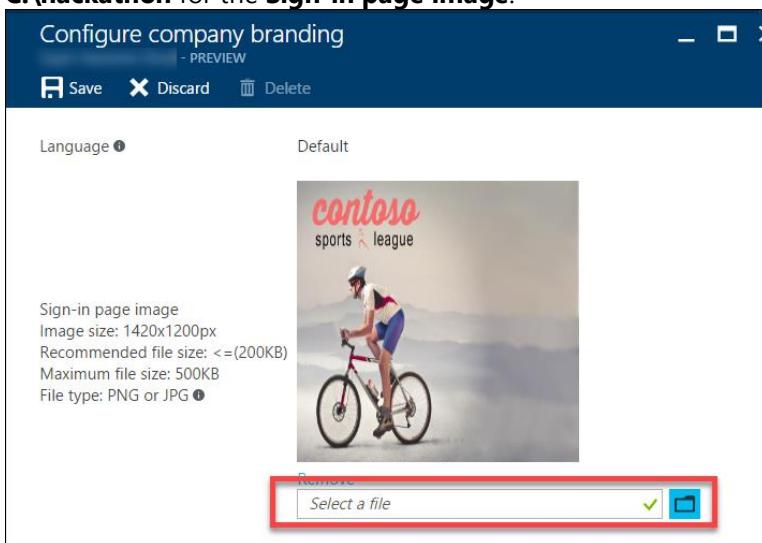
3. On the **Azure Active Directory** blade, click on **Company branding**.



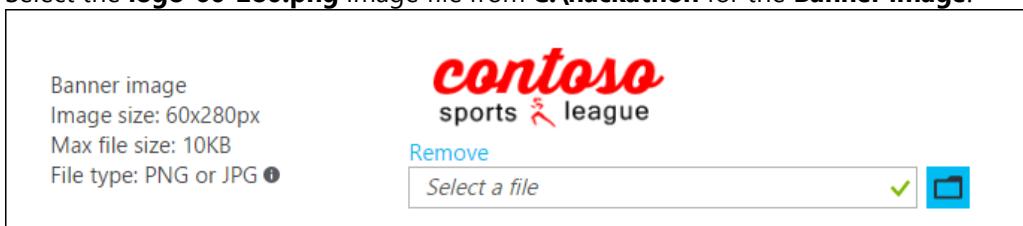
4. Click on the **Configure company branding now** link.



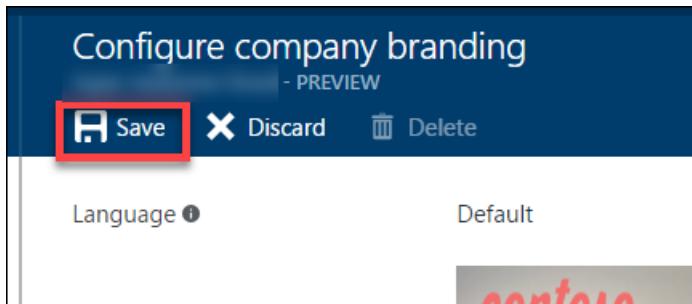
5. On the **Configure company branding** blade, select the **default_signin_illustration.jpg** image file from **C:\hackathon** for the **Sign-in page image**.



6. Select the **logo-60-280.png** image file from **C:\hackathon** for the **Banner image**.

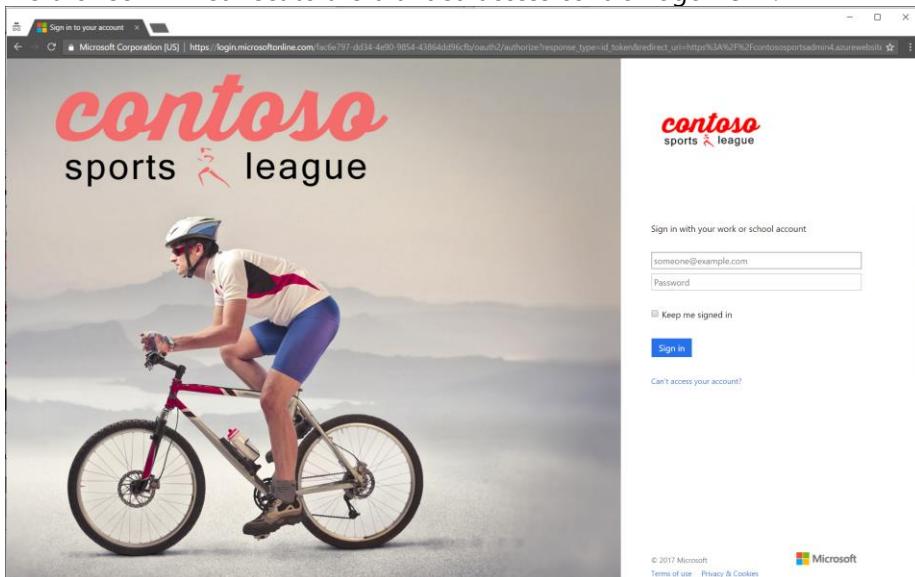


Click **Save**.

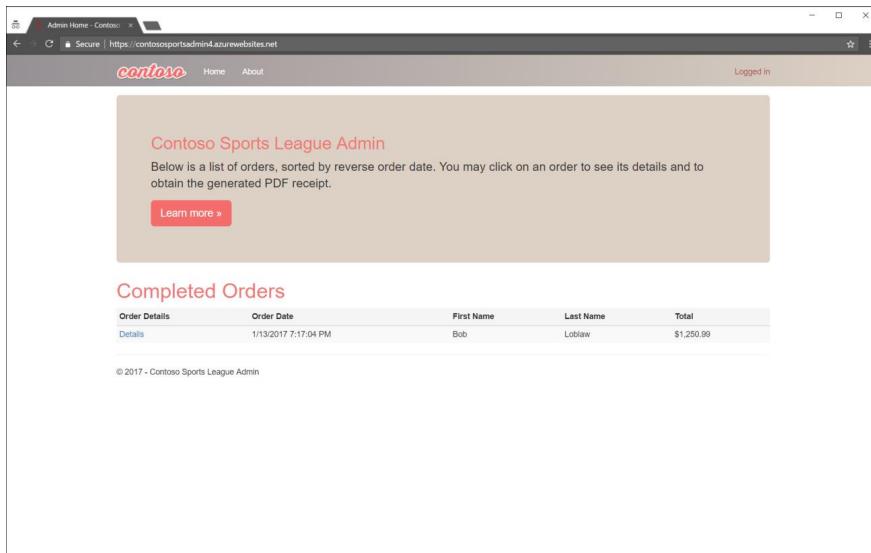


Task 5: Verify the branding has been successfully applied to the Azure Active Directory logon page

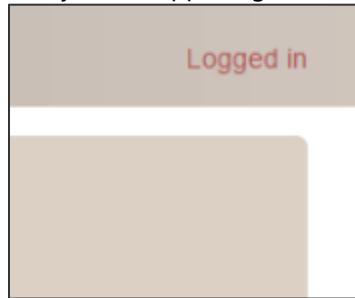
1. Close any previously authenticated browser sessions to the call center administration website, reopen using InPrivate or Incognito mode, and navigate to the **call center administration** website.
2. The browser will redirect to the branded access control logon URL.



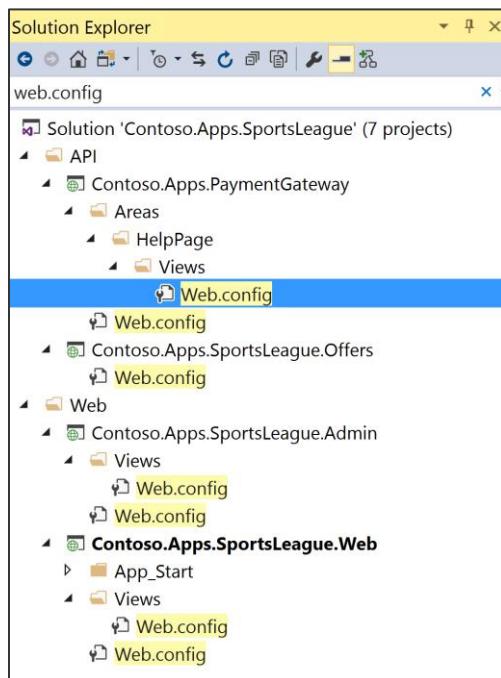
3. After you log on, your browser will be redirected to the Contoso Sports League Admin webpage.



4. Verify in the upper-right corner you see the link **Logged in**.



5. If you run the app using localhost, ensure connection strings for all of the web.config files in the solution have the placeholders removed with actual values. Search on web.config in the solution explorer to come up with the list.



Exercise 3: Enable Azure B2C for customer site

Duration: 75 minutes

In this exercise, you will configure an Azure AD Business to Consumer (B2C) instance to enable authentication and policies for sign-in, sign-out and profile policies for the Contoso E-Commerce site.

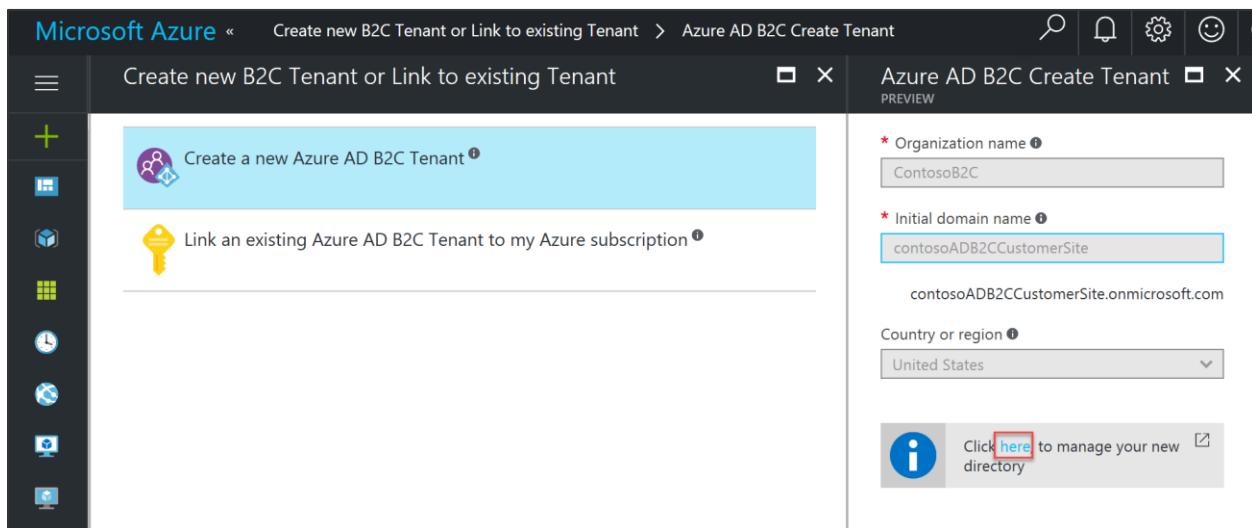
Task 1: Create a new directory

1. Log in to the Azure portal by using your existing Azure subscription or by starting a free trial. At the left bottom of the screen, click **New > Azure Active Directory B2C**.

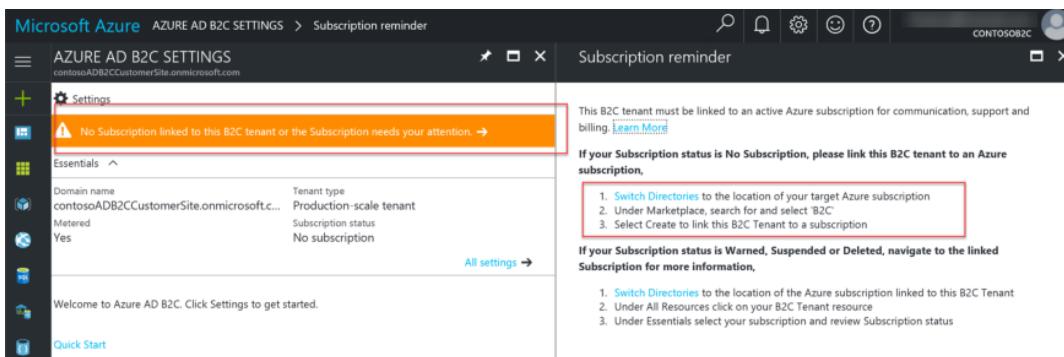
The image consists of two screenshots from the Microsoft Azure portal. The top screenshot shows the 'New' blade with a search bar containing 'active directory B2C'. Below the search bar, 'Azure Active Directory B2C' is listed under the 'MARKETPLACE' section. The bottom screenshot shows the search results for 'Azure Active Directory B2C' in the Marketplace. The results table has columns for NAME, PUBLISHER, and CATEGORY. One result is shown: 'Azure Active Directory B2C' by Microsoft, categorized under 'Security + Identity'.

NAME	PUBLISHER	CATEGORY
Azure Active Directory B2C	Microsoft	Security + Identity

2. Enter for the name, **ContosoB2C** and a unique domain name and region. Click **Create a new Azure AD B2C Tenant**, and it will take a minute to complete. Click the link to manage your new B2C Directory.

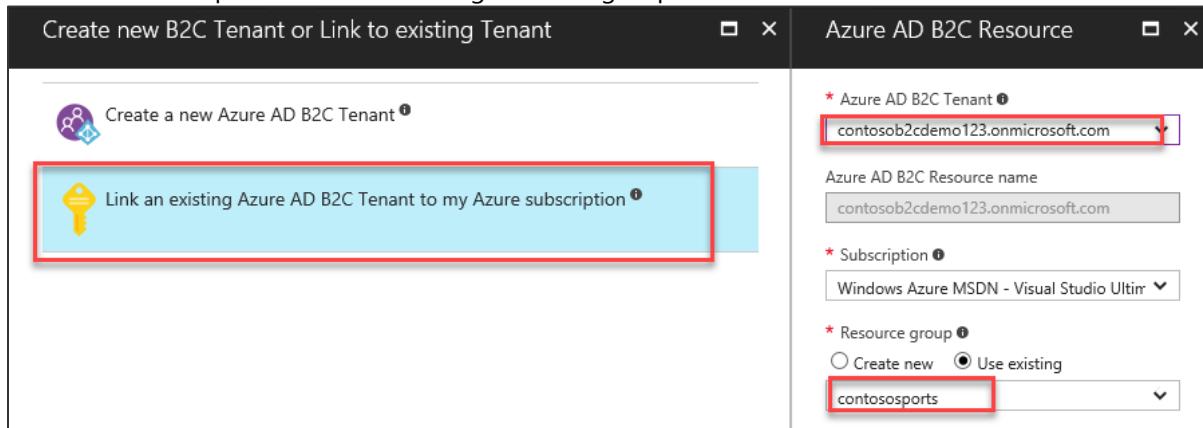


3. Click on the orange No Subscription message for instructions on how to link to an active subscription.



Note: Essentially, you will need to switch back to your previous Azure AD tenant, and then launch the Azure AD B2C creation wizard again.

4. Click on **Link an existing Azure AD B2C Tenant to my Azure subscription**, and select the Tenant you just created in the dropdown list and existing resource group. Press **Create**.



5. Open the new Azure AD B2C tenant.
6. Click on **All Settings > Applications > +Add**.

AZURE AD B2C SETTINGS > Settings > Applications

AZURE AD B2C SETTINGS
contosoADB2CCustomerSite.onmicrosoft.com

Essentials

- Domain name: contosoADB2CCustomerSite.onmicrosoft.com
- Tenant type: Production-scale tenant
- Metered: Yes
- Subscription ID: da487eb6-8aaf-45ce-ab90-e4767fc8f646
- Resource name: contosoADB2CCustomerSite.onmicrosoft.com

All settings →

Welcome to Azure AD B2C. Click Settings to get started.

Quick Start

Submit support request

Settings
contosoADB2CCustomerSite.onmicrosoft.com

Search resources

MANAGE

- Applications** > (Red box)
- Identity providers
- User attributes
- Users and groups

POLICIES

- Sign-up policies
- Sign-in policies
- Sign-up or sign-in policies
- Profile editing policies
- Password reset policies
- All policies

NAME

No applications found in the tenant

+ Add

Task 2: Add a new application

1. Specify the following configuration options for the Web App:
 - Name: Contoso B2C Application
 - Reply URL: [https://\[your web url\].azurewebsites.net](https://[your web url].azurewebsites.net) <- this should be the HTTPS URL to the Contoso E-Commerce Site.
 - Include Web App / web API: Yes

New application

* Name ⓘ
Contoso B2C Application ✓

Web App / Web API
Include web app / web API ⓘ
Yes No

Allow implicit flow ⓘ
Yes No

Redirect URIs must all belong to the same domain

Reply URL ⓘ

2. Click **Create**.
3. Click the application you just created, and copy down the globally unique **Application ID** you will use later in your code.

Task 3: Create Policies, Sign up

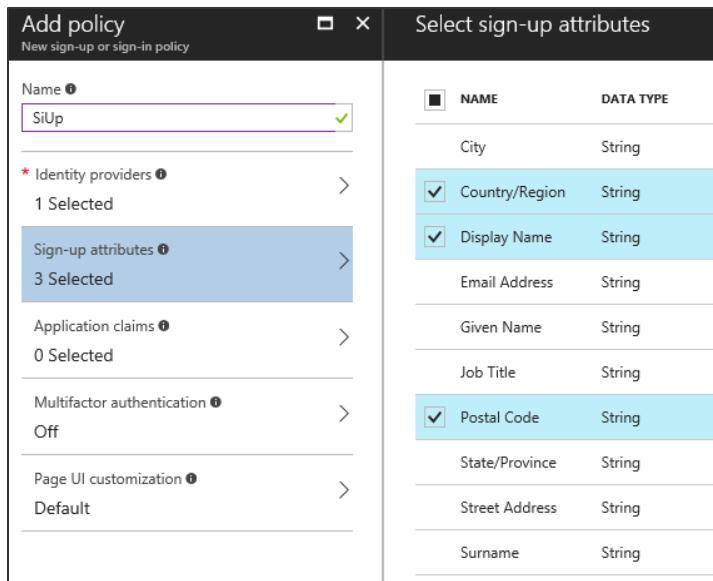
1. Open your Azure AD B2C Tenant in the Azure management portal.
2. To enable sign-up on your application, you will need to create a sign-up policy. This policy describes the experiences consumers will go through during sign-up and the contents of tokens the application will receive on successful sign-ups. Click **Sign-up or sign-in policies** as well as **+Add** at the top of the blade.

The screenshot shows the 'Manage' section of the Azure AD B2C portal. On the left, there's a sidebar with 'Filter settings' and sections for 'MANAGE' (Applications, Identity providers, User attributes, Users and groups) and 'POLICIES' (Sign-up or sign-in policies). The 'Sign-up or sign-in policies' link is highlighted with a red box. On the right, a blade titled 'No policies' has a '+ Add' button at the top, also highlighted with a red box.

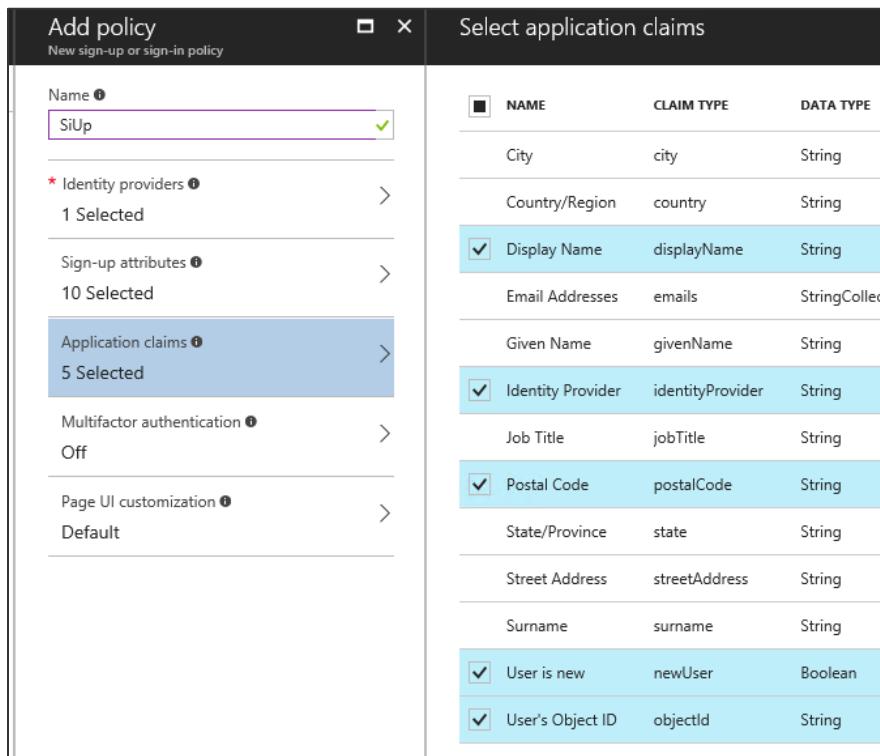
3. The **Name** determines the sign-up policy name used by your application. For example, enter "SiUp."
4. Click **Identity providers**, and select "Email signup." Optionally, you can also select social identity providers (if previously configured for the tenant). Click **OK**.

The screenshot shows the 'Add policy' dialog. It has fields for 'Name' (set to 'SiUp') and 'Identity providers' (set to '0 Selected'). A 'Select identity providers' blade is open, listing 'NAME' and 'Email signup' under 'IDENTITY PROVIDER' (Local Account).

5. Click **Sign-up attributes**. Here, you choose attributes you want to collect from the consumer during sign-up. For example, select "Country/Region," "Display Name" and "Postal Code." Click **OK**.



- Click **Application claims**. Here, you choose claims you want returned in the tokens sent back to your application after a successful sign-up experience. For example, select "Display Name," "Identity Provider," "Postal Code," "User is new" and "User's Object ID." Click **OK**.



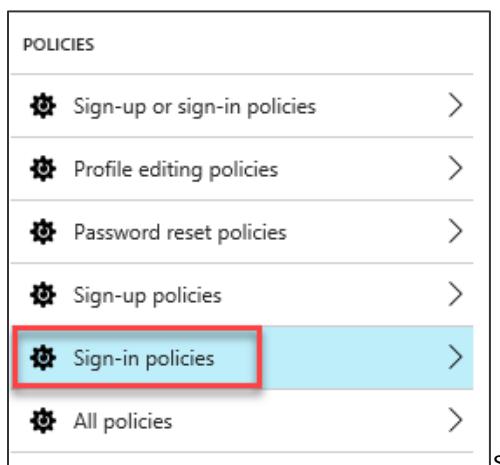
- Click **Create**. Observe the policy just created appears as "**B2C_1_SiUp**" (the **B2C_1_** fragment is automatically added) in the **Sign-up policies** blade.
- Open the policy by clicking "**B2C_1_SiUp**".

9. Select "**Contoso B2C app**" in the **Applications** drop-down.
10. Click **Run now**. A new browser tab opens, and you can run through the consumer experience of signing up for your application.

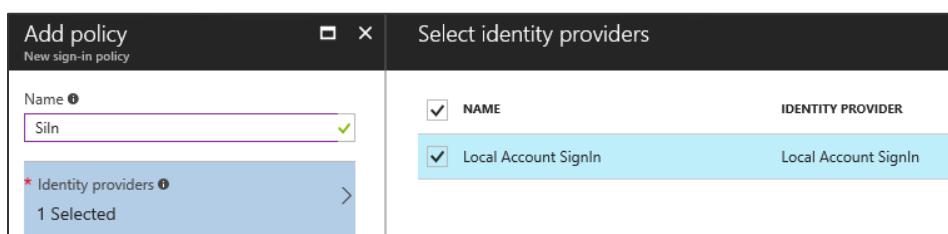
Task 4: Create a sign-in policy

To enable sign-in on your application, you will need to create a sign-in policy. This policy describes the experiences consumers will go through during sign-in and the contents of tokens the application will receive on successful sign-ins.

1. Click **Sign-in policies**.



2. Click **+Add** at the top of the blade.
3. The **Name** determines the sign-in policy name used by your application. For example, enter "**SiIn**" <the 3rd letter is an upper case i>.
4. Click **Identity providers** and select "**Local Account SignIn**." Optionally, you can also select social identity providers, if already configured. Click **OK**.



5. Click **Application claims**. Here you choose claims that you want returned in the tokens sent back to your application after a successful sign-in experience. For example, select "Display Name," "Identity Provider," "Postal Code," and "User's Object ID." Click **OK**.

6. Click **Create**. Observe the policy just created appears as "**B2C_1_Siln**" (the **B2C_1_** fragment is automatically added) in the **Sign-in policies** blade.
7. Open the policy by clicking "**B2C_1_Siln**".
8. Select "Contoso B2C app" in the **Applications** drop-down.
9. Click **Run now**. A new browser tab opens, and you can run through the consumer experience of signing into your application.

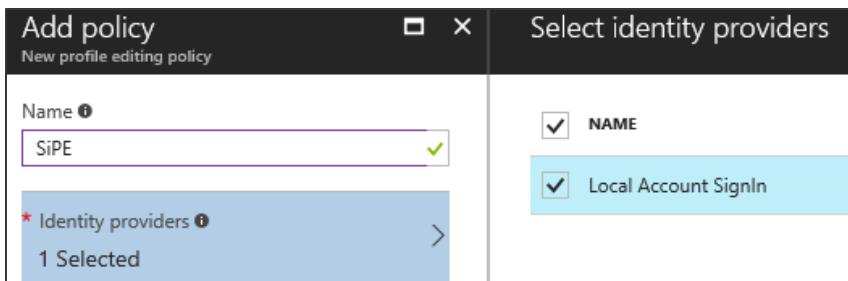
Task 5: Create a profile editing policy

To enable profile editing on your application, you will need to create a profile editing policy. This policy describes the experiences that consumers will go through during profile editing and the contents of tokens that the application will receive on successful completion.

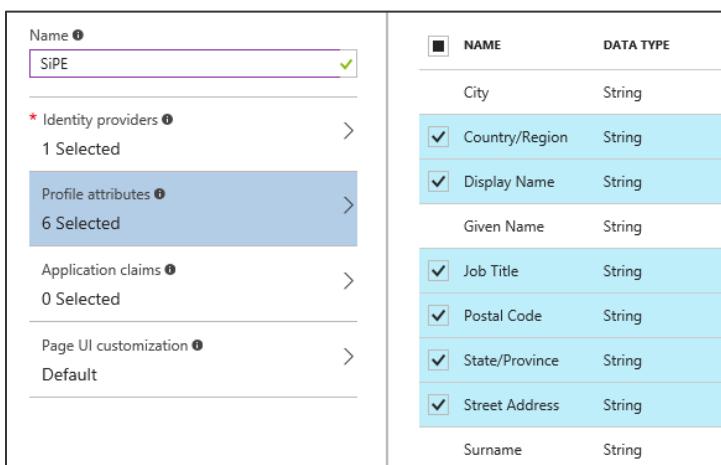
1. Click **Profile editing policies**.

2. Click **+Add** at the top of the blade.

3. The Name determines the profile editing policy name used by your application. For example, enter "SiPe."
4. Click Identity providers, and select "**Local Account SignIn**." Optionally, you can also select social identity providers, if already configured. Click **OK**.



5. Click **Profile attributes**. Here, you choose attributes the consumer can view and edit. For example, select "Country/Region," "Display Name," "Job Title," "Postal Code," "State/Province," and "Street Address." Click **OK**.



6. Click **Application claims**. Here, you choose claims you want returned in the tokens sent back to your application after a successful profile editing experience. For example, select "Display Name" and "Postal Code." Click **OK**
7. Click **Create**. Observe the policy just created appears as "**B2C_1_SiPe**" (the **B2C_1_** fragment is automatically added) in the **Profile editing policies** blade.
8. Open the policy by clicking "**B2C_1_SiPe**."
9. Select "Contoso B2C app" in the **Applications** drop-down.
10. Click **Run now**. A new browser tab opens, and you can run through the profile editing consumer experience in your application.

Task 6: Modify the Contoso.App.SportsLeague.Web

1. Within Visual Studio, click on **View -> Other Windows -> Package Manager Console**. Execute the following commands to install these the required NuGet Packages.

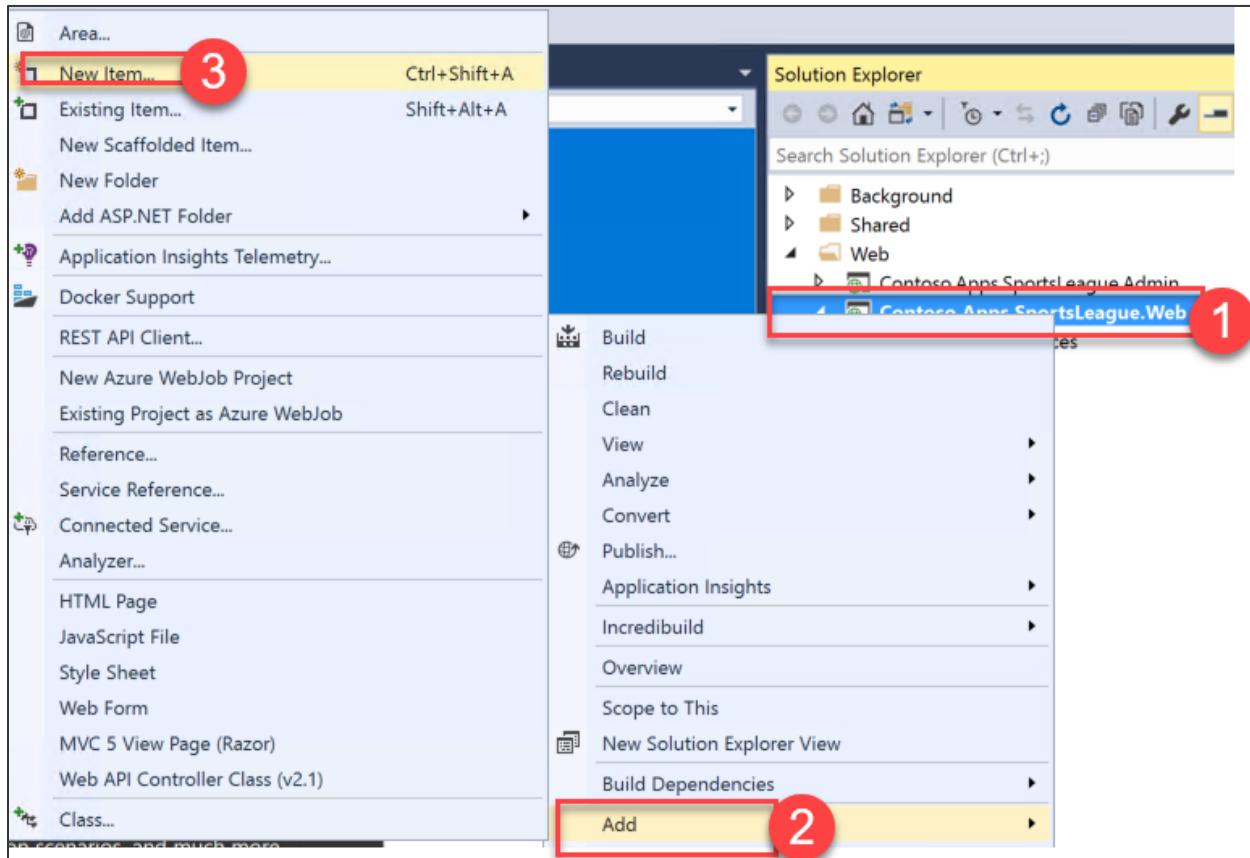
```
Install-Package Microsoft.Owin.Security.OpenIdConnect -Version 3.0.1
Install-Package Microsoft.Owin.Security.Cookies -Version 3.0.1
Install-Package Microsoft.Owin.Host.SystemWeb -Version 3.0.1
Install-Package Microsoft.IdentityModel.Protocol.Extensions -Version
1.0.4.403061554
```

2. Next, using the Azure Management Portal, open the Contoso E-Commerce Site, and click on App Settings.
3. Add the following settings:

- ida:Tenant - [your Azure AD B2C name].onmicrosoft.com
- ida:ClientId – [the client/app ID from your app]
- ida:RedirectUri - https://[your web url].azurewebsites.net
- ida:SignupPolicyId – B2C_1_SiUp
- ida:SignInPolicyId – B2C_1_Siln <the 3rd letter is an upper case i>
- ida:UserProfilePolicyId – B2C_1_SiPe
- ida:AadInstance - https://login.microsoftonline.com/{0}/v2.0/.well-known/openid-configuration?p={1}

App settings	
WEBSITE_NODE_DEFAULT_VERSION	6.9.1
AzureQueueConnectionString	DefaultEndpointsProtocol=https;AccountName=contosostorag...
paymentsAPIUrl	https://paymentsapi123.azurewebsites.net/api/nvp
offersAPIUrl	https://offersapi123.azurewebsites.net/api/get
ida:Tenant	contosob2cdemo123.onmicrosoft.com
ida:ClientId	15d29a82-334c-4470-9388-700a45b3be18
ida:RedirectUri	https://contososportsweb12312.azurewebsites.net/
ida:SignupPolicyId	B2C_1_SiUp
ida:SignInPolicyId	B2C_1_Siln
ida:UserProfilePolicyId	B2C_1_SiPe

4. Click **Save** when you are complete.
5. Within Visual Studio, **right** click on the **Contoso.Apps.SportsLeague.Web** project, and click **Add -> New Item**.



6. In the Search Installed Templates search box search for OWIN. Click the OWIN Startup class, change the name to **Startup.cs**, and then click **Add**.
7. In the new class, insert the word partial in between public and class to make this a partial class.

```
namespace Contoso.Apps.SportsLeague.Web
{
    public partial class Startup
    {
        public void Configuration(IAppBuilder app)
        {
            // For more information on how to config
        }
    }
}
```

8. Add the following code between the brackets of the Configuration method.

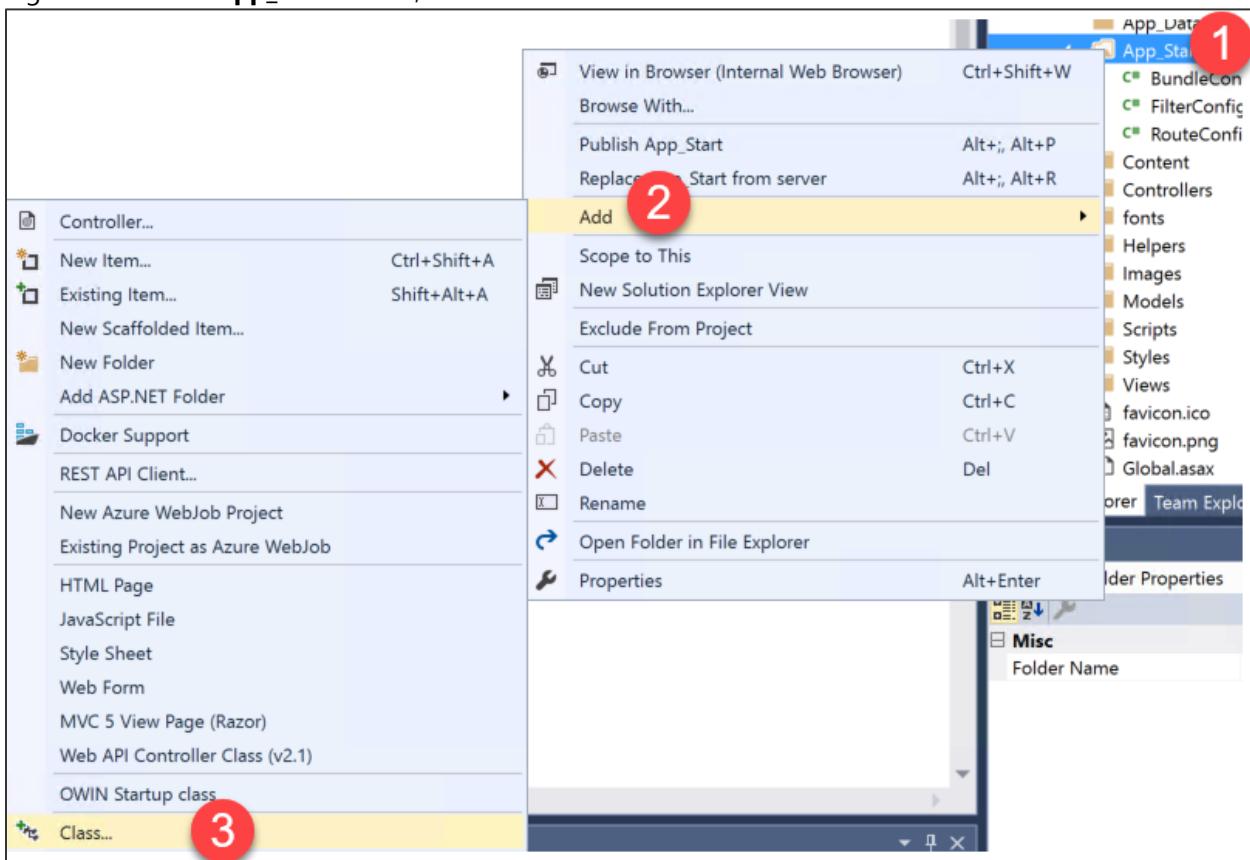
```
ConfigureAuth(app);
```

```
// Startup.cs

public partial class Startup
{
    public void Configuration(IAppBuilder app)
    {
        ConfigureAuth(app);
    }
}
```

Note: The OWIN middleware will invoke the Configuration(...) method when your app starts.

9. Right click on the **App_Start** folder, and click **Add -> Class**.



10. Select **Visual C# and Class**, and name the new file **Startup.Auth.cs**.



11. Replace the entire contents of Startup.Auth.cs with the following code:

```
// App_Start\Startup.Auth.cs
using System;
using Owin;
using Microsoft.Owin.Security;
using Microsoft.Owin.Security.Cookies;
using Microsoft.Owin.Security.OpenIdConnect;
using System.Threading.Tasks;
using Microsoft.Owin.Security.Notifications;
using Microsoft.IdentityModel.Protocols;

using System.Configuration;
using System.IdentityModel.Tokens;
using System.Web.Helpers;
using System.IdentityModel.Claims;

namespace Contoso.Apps.SportsLeague.Web
{
```

```
public partial class Startup
{
    // App config settings
    private static string clientId =
ConfigurationManager.AppSettings["ida:ClientId"];
    private static string aadInstance =
ConfigurationManager.AppSettings["ida:AadInstance"];
    private static string tenant =
ConfigurationManager.AppSettings["ida:Tenant"];
    private static string redirectUri =
ConfigurationManager.AppSettings["ida:RedirectUri"];

    // B2C policy identifiers
    public static string SignUpPolicyId =
ConfigurationManager.AppSettings["ida:SignUpPolicyId"];
    public static string SignInPolicyId =
ConfigurationManager.AppSettings["ida:SignInPolicyId"];
    public static string ProfilePolicyId =
ConfigurationManager.AppSettings["ida:UserProfilePolicyId"];

    public void ConfigureAuth(IAppBuilder app)
    {

        app.SetDefaultSignInAsAuthenticationType(CookieAuthenticationDefaults.AuthenticationType);

        app.UseCookieAuthentication(new CookieAuthenticationOptions());

        // Configure OpenID Connect middleware for each policy

        app.UseOpenIdConnectAuthentication(CreateOptionsFromPolicy(SignUpPolicyId));

        app.UseOpenIdConnectAuthentication(CreateOptionsFromPolicy(ProfilePolicyId));
    }

    app.UseOpenIdConnectAuthentication(CreateOptionsFromPolicy(SignInPolicyId));
    AntiForgeryConfig.UniqueClaimTypeIdentifier = ClaimTypes.NameIdentifier;
}

// Used for avoiding yellow-screen-of-death
private Task<AuthenticationFailedNotification<OpenIdConnectMessage, OpenIdConnectAuthenticationOptions>> AuthenticationFailed(AuthenticationFailedNotification<OpenIdConnectMessage, OpenIdConnectAuthenticationOptions> notification)
{
    notification.HandleResponse();
    if (notification.Exception.Message == "access_denied")
    {
        notification.Response.Redirect("/");
    }
}
```

```
        }

        else
        {
            notification.Response.Redirect("/Home/Error?message=" +
notification.Exception.Message);
        }

        return Task.FromResult(0);
    }

    private OpenIdConnectAuthenticationOptions
CreateOptionsFromPolicy(string policy)
{
    return new OpenIdConnectAuthenticationOptions
    {
        // For each policy, give OWIN the policy-specific metadata
address, and
        // set the authentication type to the id of the policy
        MetadataAddress = String.Format(aadInstance, tenant, policy),
        AuthenticationType = policy,

        // These are standard OpenID Connect parameters, with values
pulled from web.config
        ClientId = clientId,
        RedirectUri = redirectUri,
        PostLogoutRedirectUri = redirectUri,
        Notifications = new OpenIdConnectAuthenticationNotifications
        {
            AuthenticationFailed = AuthenticationFailed,
        },
        Scope = "openid",
        ResponseType = "id_token",

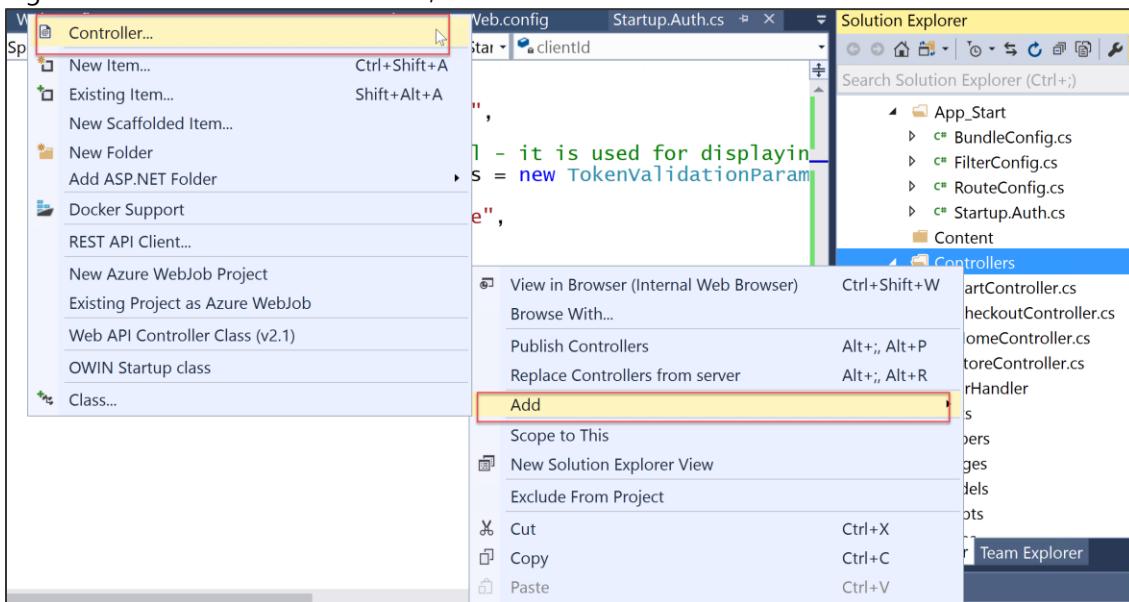
        // This piece is optional - it is used for displaying the user's
name in the navigation bar.
        TokenValidationParameters = new TokenValidationParameters
        {
            NameClaimType = "name",
        },
    };
}
}
```

Note: The parameters you provide in OpenIdConnectAuthenticationOptions serve as coordinates for your app to communicate with Azure AD. You also need to set up cookie authentication. The OpenID Connect middleware uses cookies to maintain user sessions, among other things.

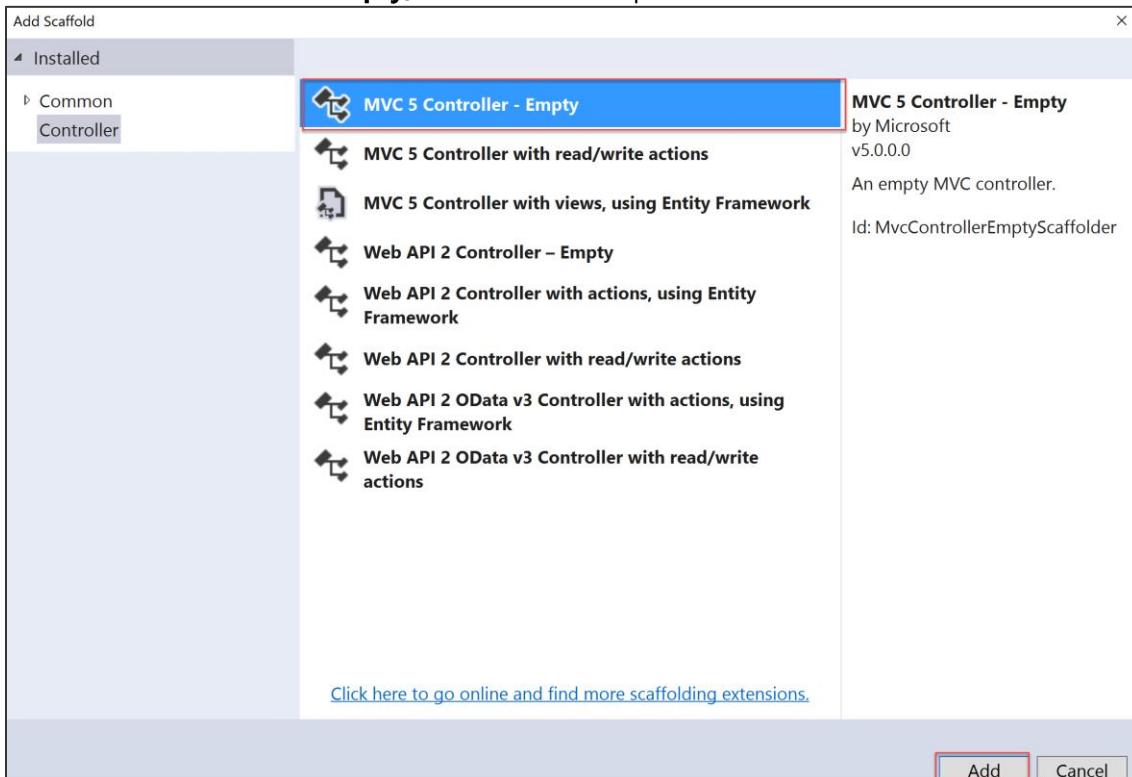
Task 7: Send authentication requests to Azure AD

Your app is now properly configured to communicate with Azure AD B2C by using the OpenID Connect authentication protocol. OWIN has taken care of all of the details of crafting authentication messages, validating tokens from Azure AD, and maintaining user session. All that remains is to initiate each user's flow.

- Right click on the **Controllers** folder, and click **Add -> Controller**.



- Select **MVC 5 Controller – Empty**, and click **Add**. Replace **Default** with **Account** for the controller name.



- Add the following using statement to the top of the controller:

```
using Microsoft.Owin.Security;
```

4. Replace the default controller method Index

```
public class AccountController : Controller
{
    // GET: Account
    public ActionResult Index()
    {
        return View();
    }
}
```

With the following code:

```
// Controllers\AccountController.cs

public void SignIn()
{
    if (!Request.IsAuthenticated)
    {
        // To execute a policy, you simply need to trigger an OWIN
challenge.
        // You can indicate which policy to use by specifying the policy id
as the AuthenticationType
        HttpContext.GetOwinContext().Authentication.Challenge(
            new AuthenticationProperties() { RedirectUri = "/" },
Startup.SignInPolicyId);
    }
}

public void SignUp()
{
    if (!Request.IsAuthenticated)
    {
        HttpContext.GetOwinContext().Authentication.Challenge(
            new AuthenticationProperties() { RedirectUri = "/" },
Startup.SignUpPolicyId);
    }
}

public void Profile()
{
    if (Request.IsAuthenticated)
    {
        HttpContext.GetOwinContext().Authentication.Challenge(
            new AuthenticationProperties() { RedirectUri = "/" },
Startup.ProfilePolicyId);
    }
}
```

5. You can also use OWIN to sign out the user from the app. Add the following method to the account controller (**Controllers\AccountController.cs**):

C# Copy

```
// Controllers\AccountController.cs

public void SignOut()
{
    // To sign out the user, you should issue an OpenIDConnect sign out request
    if (Request.IsAuthenticated)
    {
        IEnumerable<AuthenticationDescription> authTypes =
        HttpContext.GetOwinContext().Authentication.GetAuthenticationTypes();
        HttpContext.GetOwinContext().Authentication.SignOut(authTypes.Select(t =>
t.AuthenticationType).ToArray());
        Request.GetOwinContext().Authentication.GetAuthenticationTypes();
    }
}
```

Task 8: Display user information

When you authenticate users by using OpenID Connect, Azure AD returns an ID token to the app that contains **claims**. These are assertions about the user. You can use claims to personalize your app. You can access user claims in your controllers via the ClaimsPrincipal.Current security principal object.

1. Open the **Controllers\HomeController.cs** file and add the following using statements at the end of the other using statements.

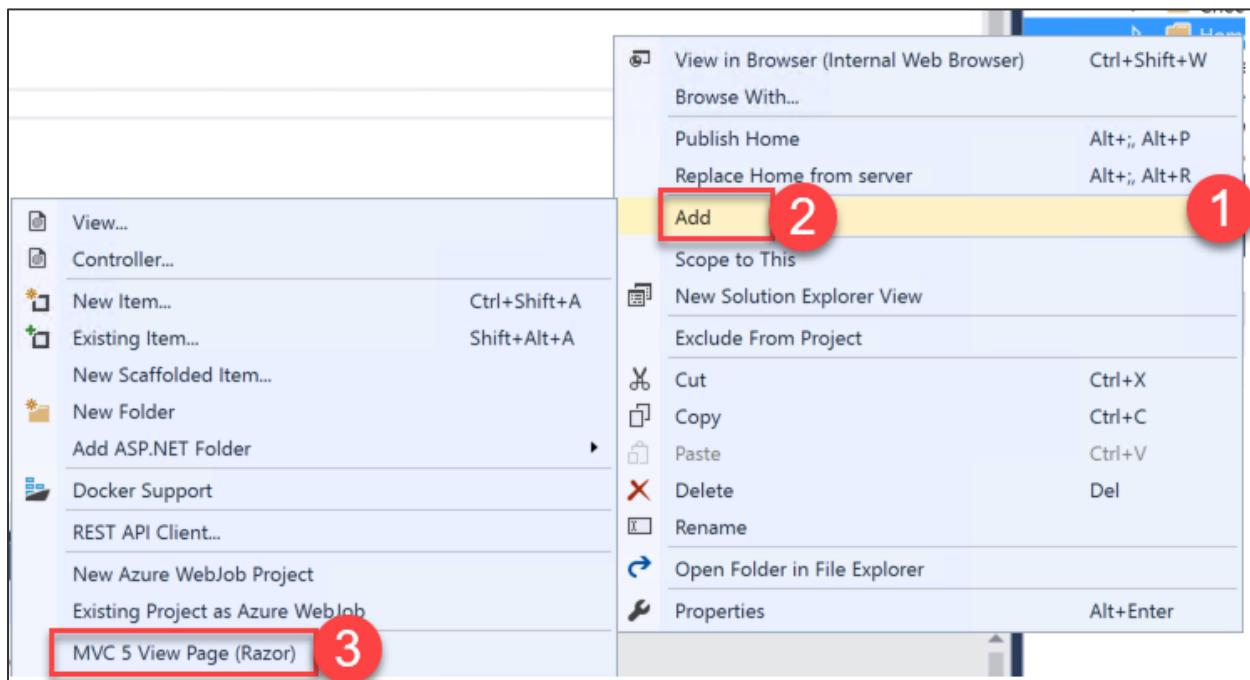
```
using System.Linq;
using System.Security.Claims;
```

2. Open the **Controllers\HomeController.cs** file and add the following method:

```
[Authorize]
public ActionResult Claims()
{
    Claim displayName =
    ClaimsPrincipal.Current.FindFirst(ClaimsPrincipal.Current.Identities.First()
    .NameClaimType);
    ViewBag.DisplayName = displayName != null ? displayName.Value :
    string.Empty;
    return View();
}
```

3. You can access any claim that your application receives in the same way. A list of all the claims the app receives is available for you on the **Claims** page. Right click on **Views -> Home**, click **Add -> MVC 5 View Page (Razor)** and

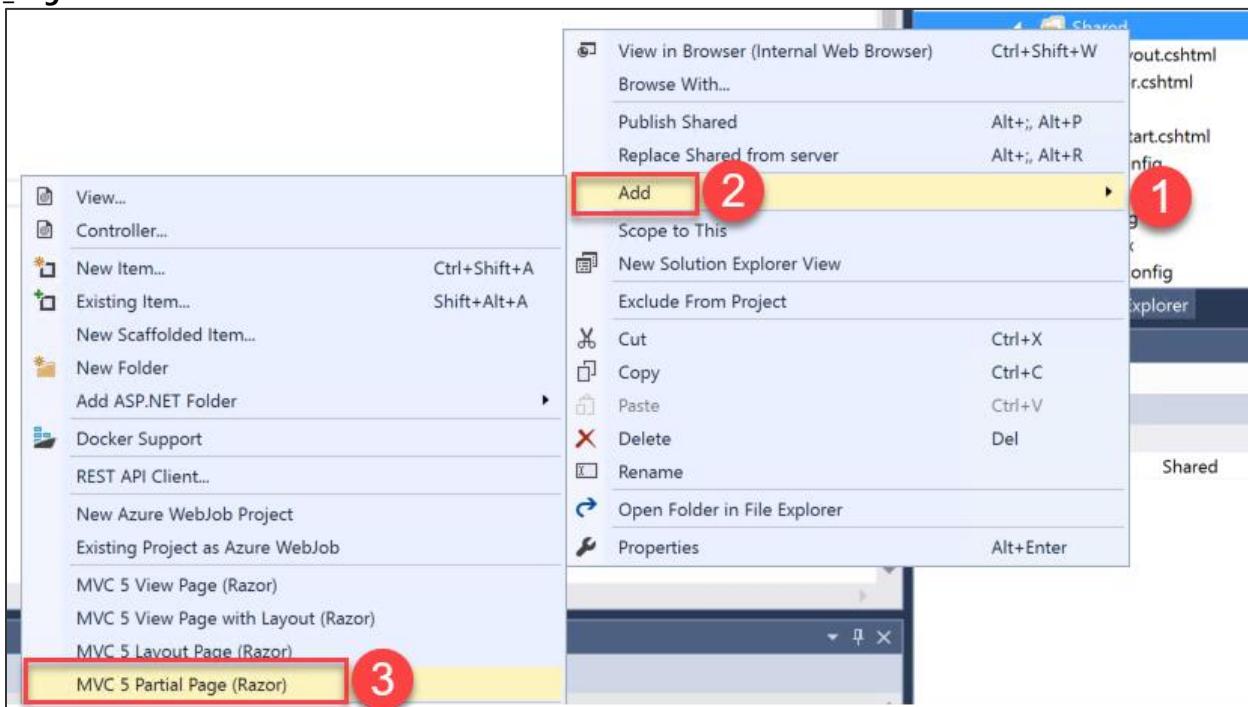
name it **Claims**.



4. Open the **Claims.cshtml** file and replace the code with the following:

```
@using System.Security.Claims  
{@  
    ViewBag.Title = "Claims";  
}  
<h2>@ViewBag.Title</h2>  
  
<h4>Claims Present in the Claims Identity: @ViewBag.DisplayName</h4>  
  
<table class="table-hover claim-table">  
    <tr>  
        <th class="claim-type claim-data claim-head">Claim Type</th>  
        <th class="claim-data claim-head">Claim Value</th>  
    </tr>  
  
    @foreach (Claim claim in ClaimsPrincipal.Current.Claims)  
    {  
        <tr>  
            <td class="claim-type claim-data">@claim.Type</td>  
            <td class="claim-data">@claim.Value</td>  
        </tr>  
    }  
</table>
```

5. Right click on the **Views -> Shared** folder, click **Add**, and add a new **MVC 5 Partial Page (Razor)**. Specify **_LoginPartial** for the name.



6. Add the following code to the razor partial view to provide a sign-in and sign-out link as well as a link to edit the user's profile.

```

@if (Request.IsAuthenticated)
{
    <text>
        <ul class="nav navbar-nav navbar-right">
            <li>
                <a id="profile-link">@User.Identity.Name</a>
                <div id="profile-options" class="nav navbar-nav navbar-right">
                    <ul class="profile-links">
                        <li class="profile-link">
                            @Html.ActionLink("Edit Profile", "Profile", "Account")
                        </li>
                    </ul>
                </div>
            </li>
            <li>
                @Html.ActionLink("Sign out", "SignOut", "Account")
            </li>
        </ul>
    </text>
}
else
{

```

```
<ul class="nav navbar-nav navbar-right">
    <li>@Html.ActionLink("Sign up", "SignUp", "Account", routeValues:
null, htmlAttributes: new { id = "signUpLink" })</li>
    <li>@Html.ActionLink("Sign in", "SignIn", "Account", routeValues:
null, htmlAttributes: new { id = "loginLink" })</li>
</ul>
}
```

7. Open Views\Shared_Layout.cshtml in Visual Studio. Locate the header-top div. and add the two lines highlighted.

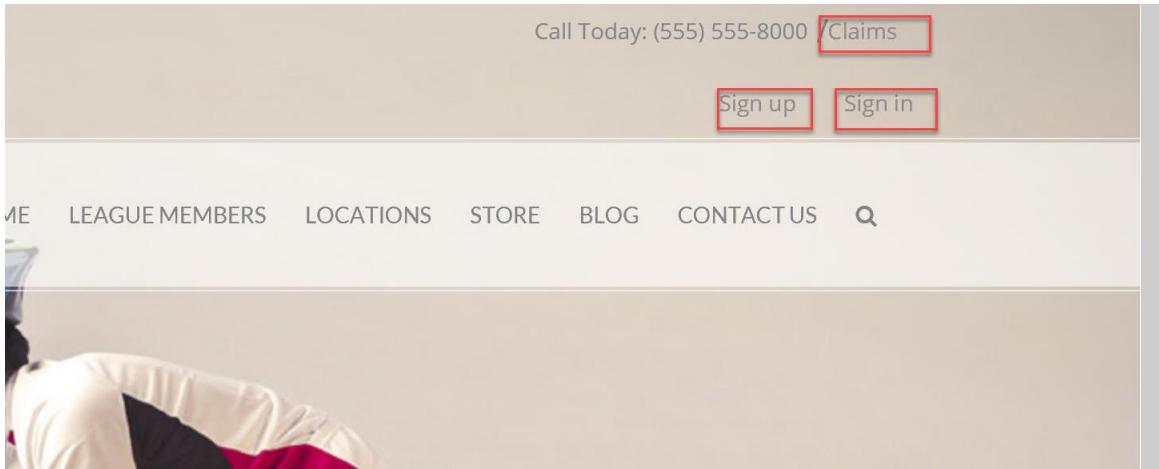
```
<div class="header-top">
<div class="container">
<div class="row">
    <div class="header-top-left">
        <a href="#"><i class="fa fa-twitter"></i></a>
        <a href="#"><i class="fa fa-facebook"></i></a>
        <a href="#"><i class="fa fa-linkedin"></i></a>
        <a href="#"><i class="fa fa-instagram"></i></a>
    </div>
    <div class="header-top-right">
        <a href="#" class="top-wrap"><span class="icon-phone">Call today: </span>
(555) 555-8000</a>
        @Html.ActionLink("Claims", "Claims", "Home")
    </div>
        @Html.Partial("_LoginPartial")
    </div>
</div>
</div>
```

Task 9: Run the sample app

1. Right click on the **Contoso.Apps.SportsLeague.Web** project, and click **Publish**. Follow the steps to deploy the updated application to the Microsoft Azure Web App.

Launch a browser outside of Visual Studio for testing if the page loads in Visual Studio.

2. Test out Sign up. Next, test Sign out.
3. When you click on Claims and are not signed in, it will bring you to the sign-in page and then display the claim information. Sign in, and test Edit Profile.



Claims information page

A screenshot of a claims information page. At the top, it shows the contoso sports league logo and a navigation bar with "HOME", "LEAGUE MEMBERS", "LOCATIONS", "STORE", "BLOG", "CONTACT US", and a search icon. On the far right, there are buttons for "Russell", "Sign out", and "Edit Profile", all enclosed in a red border. The main content area is titled "Claims" and displays a table of user attributes. The columns are "Claim Type" and "Claim Value".

Claim Type	Claim Value
exp	1493260122
nbf	1493256522
ver	1.0
iss	https://login.microsoftonline.com/8260bc5c-7dff-4121-b7ba-32019a3464f7/v2.0/
http://schemas.xmlsoap.org/ws/2005/05/identity/claims/nameidentifier	9ebd9420-e5ff-48b8-a29b-46c12f369a4e
aud	5e27a739-6550-40ea-8a08-77c3a24e7809
nonce	636298533166653432.MjBiZDQxMDktNjhjZl00OTY1LWlwNWUty2i4NDMxZDU4NGJhNzViYWExMTAtZdmOC00NWZmL Tg2NzEtYTlINDIyMmQ5OGQ0
iat	1493256522
http://schemas.microsoft.com/ws/2008/06/identity/claims/authenticationinstant	1493256522
http://schemas.microsoft.com/identity/claims/objectidentifier	9ebd9420-e5ff-48b8-a29b-46c12f369a4e
name	Russell
postalCode	34652

Exercise 4: Enabling Telemetry with Application Insights

To configure the application for logging and diagnostics, you have been asked to configure Microsoft Azure Application Insights and add some custom telemetry.

Note: You may need to create an Application Insights Resource in Azure portal depending on your subscription rights. After it is created, you can configure it and add to the project using the tasks below. To create a new Application Insights resource.

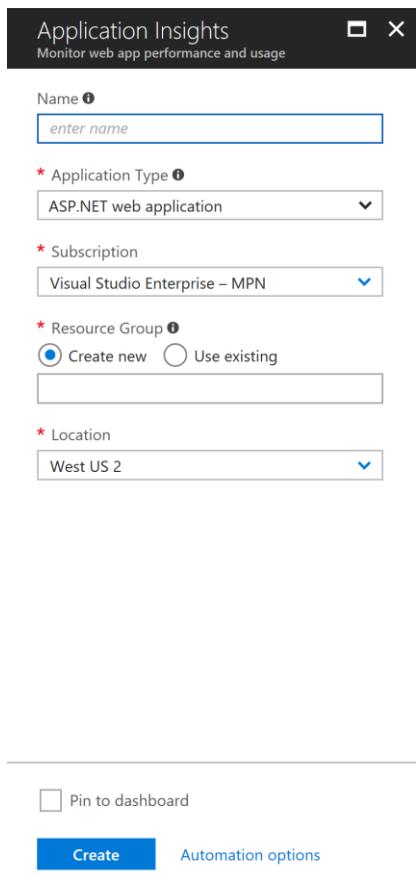
1. Click **Create** a resource. Search the marketplace for Application Insights. Select Application Insights.

The screenshot shows the Azure Marketplace search interface. On the left, there's a sidebar with options like 'Create a resource', 'All services', 'Dashboard', 'All resources', 'Resource groups', and 'App Center'. The main area has a search bar at the top with 'Application Insights' typed in. Below the search bar is a 'Filter' button. The results table has columns for 'NAME', 'PUBLISHER', and 'CATEGORY'. There is one result listed: 'Application Insights' by Microsoft, categorized under 'Web + Mobile'.

2. Click the **Create** button.

The screenshot shows the Microsoft Application Insights product page. It features a purple header with the Microsoft logo and the text 'Application Insights Microsoft'. Below the header is a section titled 'Application performance, availability and usage information at your fingertips' with three questions: 'Is my website available at all times?', 'Is the user experience responsive and usable?', and 'How are customers using my app/website?'. A paragraph below states 'Application Insights answers these questions and many more, with zero lines of code.' followed by a bulleted list of features: Availability and performance monitoring, Users and usage insights, Diagnostic logs and crash analytics, Seamless integration with Microsoft Azure and Visual Studio, and Supports ASP.NET websites (Azure or on-premises), Windows Phone, and Windows Store apps. At the bottom of this section are social sharing icons for Twitter, Facebook, LinkedIn, YouTube, Google+, and Email. Below this is a preview of the Application Insights dashboard, which includes a summary card for 'Northwind WebSite' and four charts: 'Average response time', 'Request rate', 'Median response time', and 'Session per browser'. At the very bottom is a large blue 'Create' button.

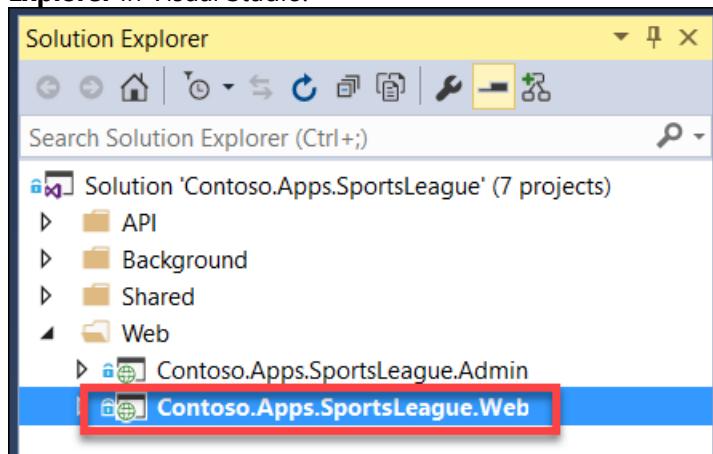
3. Enter the name as **Contoso.Apps.SportsLeague.Web**. Choose the existing resource group of **contoso**. Location should be the same location as your resource group.



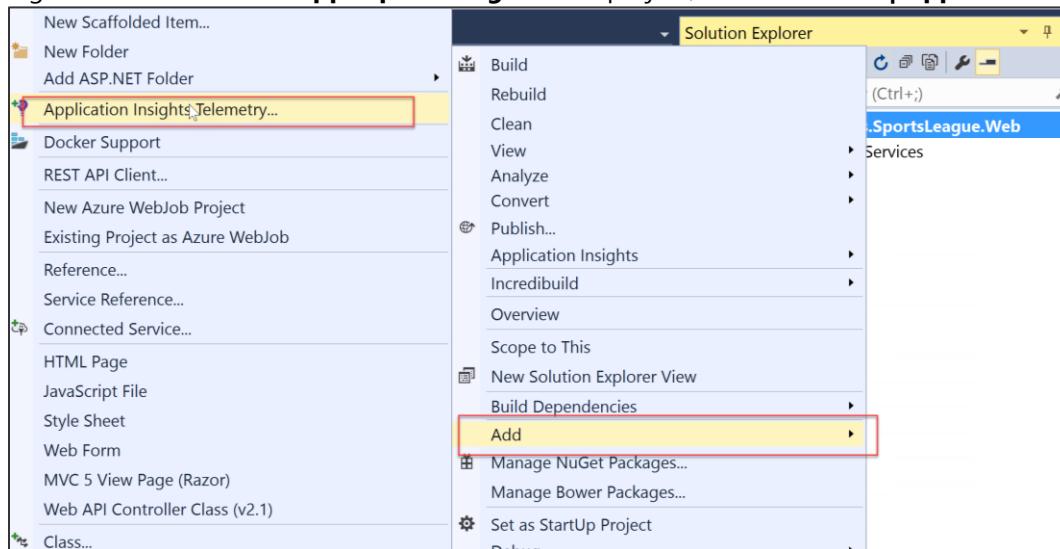
Task 1: Configure the application for telemetry

Subtask 1: Add Application Insights Telemetry to the e-commerce website project

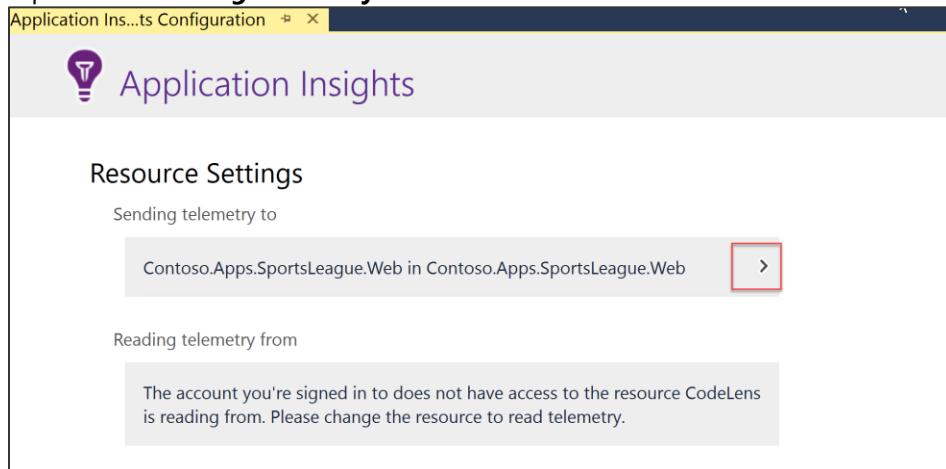
1. Open the Solution **Contoso.Apps.SportsLeague** in Visual Studio.
2. Navigate to the **Contoso.Apps.SportsLeague.Web** project located in the **Web** folder using the **Solution Explorer** in Visual Studio.



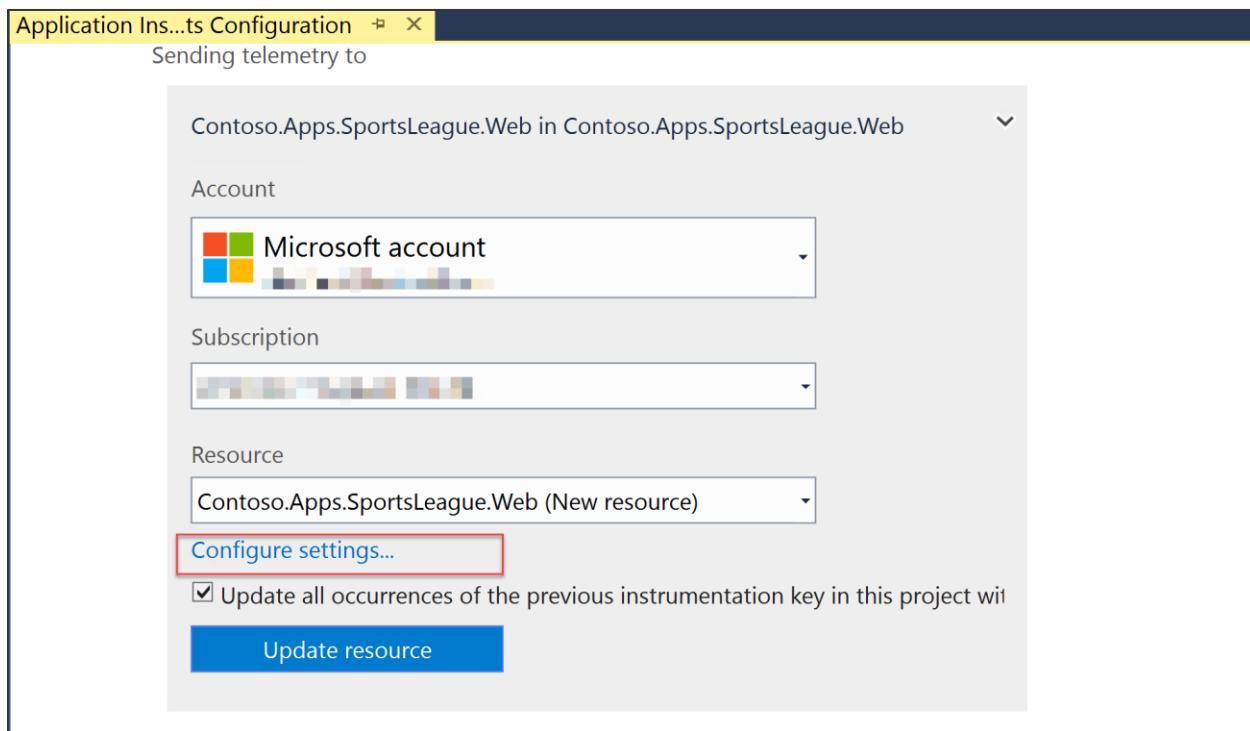
3. Right-click the **Contoso.Apps.SportsLeague.Web** project, and select **Add | Application Insights Telemetry...**



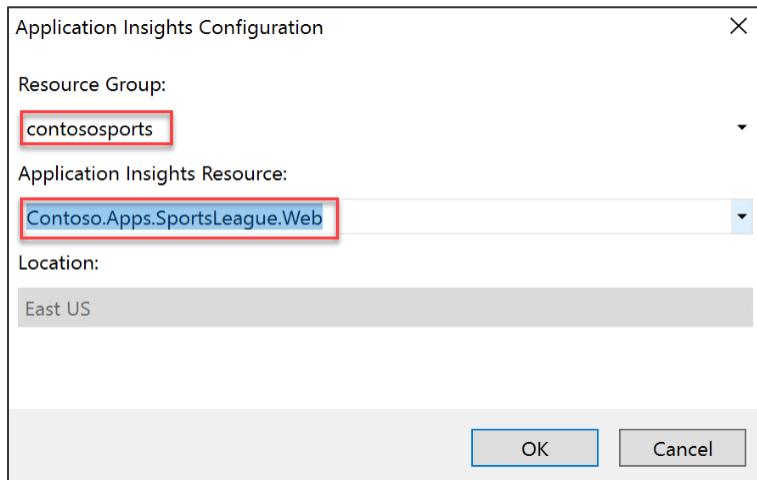
4. Expand the **Sending telemetry to** section.



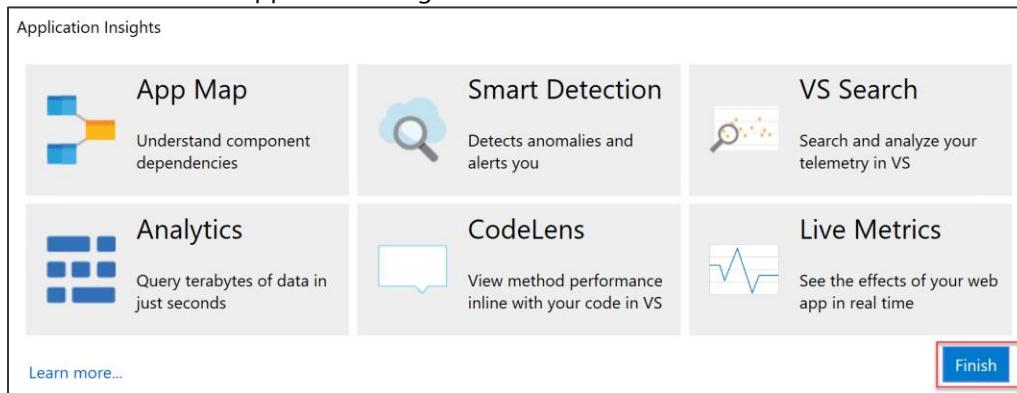
5. Click on the **Configure settings...** button.



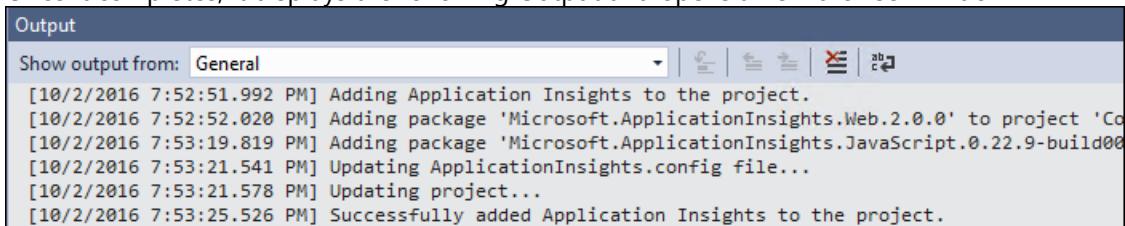
6. In the **Application Insights Configuration** dialog box, change the **Resource Group** to the **contososports** resource group used to host the Web App, and choose the New Application Insights Resource. Next, click **OK**, followed by **Update Resource**.



7. Press **Finish** on the Application Insights window.



8. Once it completes, it displays the following Output and opens a new browser window



9. Open the file **\Helpers\TelemetryHelper.cs** located in the **Contoso.Apps.SportsLeague.Web** project.

10. Add the following using statement to the top of the file:

```
using Microsoft.ApplicationInsights;
```

11. Add the following code to the **TrackException** method to instantiate the telemetry client and track exceptions.

```
var client = new TelemetryClient();  
client.TrackException(new  
Microsoft.ApplicationInsights.DataContracts.ExceptionTelemetry(exc));
```

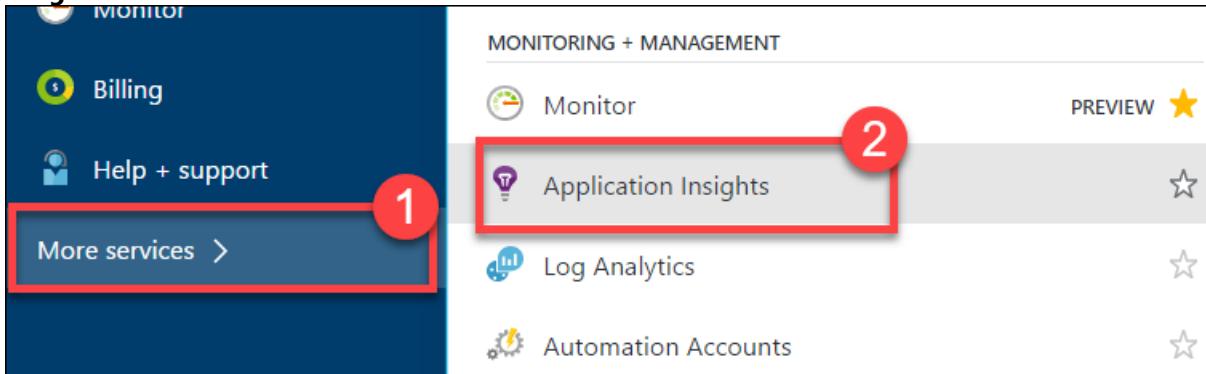
12. Add the following code to the **TrackEvent** method to instantiate the telemetry client and track event data.

```
var client = new TelemetryClient();
client.TrackEvent(eventName, properties);
```

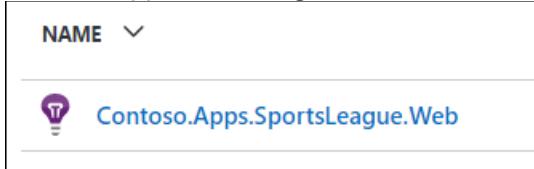
13. Save the **TelemetryHelper.cs** file.

Subtask 2: Enable client side telemetry

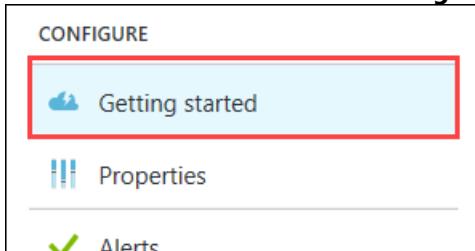
1. Open the Azure Management Portal (<http://portal.azure.com>). Click **More services** followed by **Application Insights**.



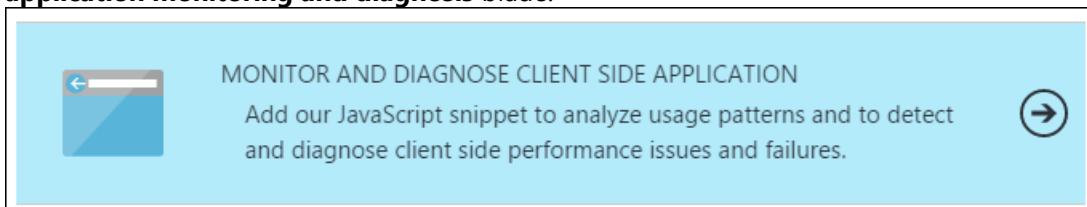
2. Click the Application Insights instance associated with the Contoso E-Commerce Site.



3. In CONFIGURE menu click on **Getting Started**.



4. Next, click the **MONITOR AND DIAGNOSE CLIENT SIDE APPLICATION** arrow. → This will open the **Client application monitoring and diagnosis** blade.



5. Select and copy the full contents of the JavaScript on the **Client application monitoring and diagnosis** blade.

 Client application monitoring and diagnosis ⚡ _ ☰

Client side telemetry

Detect and diagnose performance issues and failures in web pages. Understand how your application is being used.

- See correlated client side, server side and custom telemetry, in the context of a user session, all in one place.
- Set up alerts on the client side metrics collected by default or custom metrics reported using the JavaScript SDK.
- Slice and dice client side metrics alongside server and custom telemetry to trace the causes of performance issues and failures.

Learn more

[Application insights client side monitoring](#) ↗

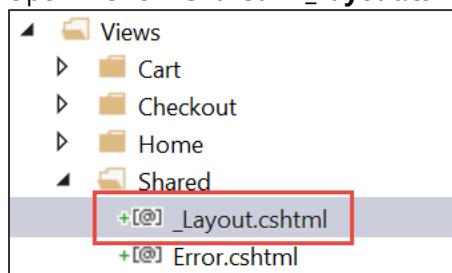
[Privacy statement](#) ↗

Guidance

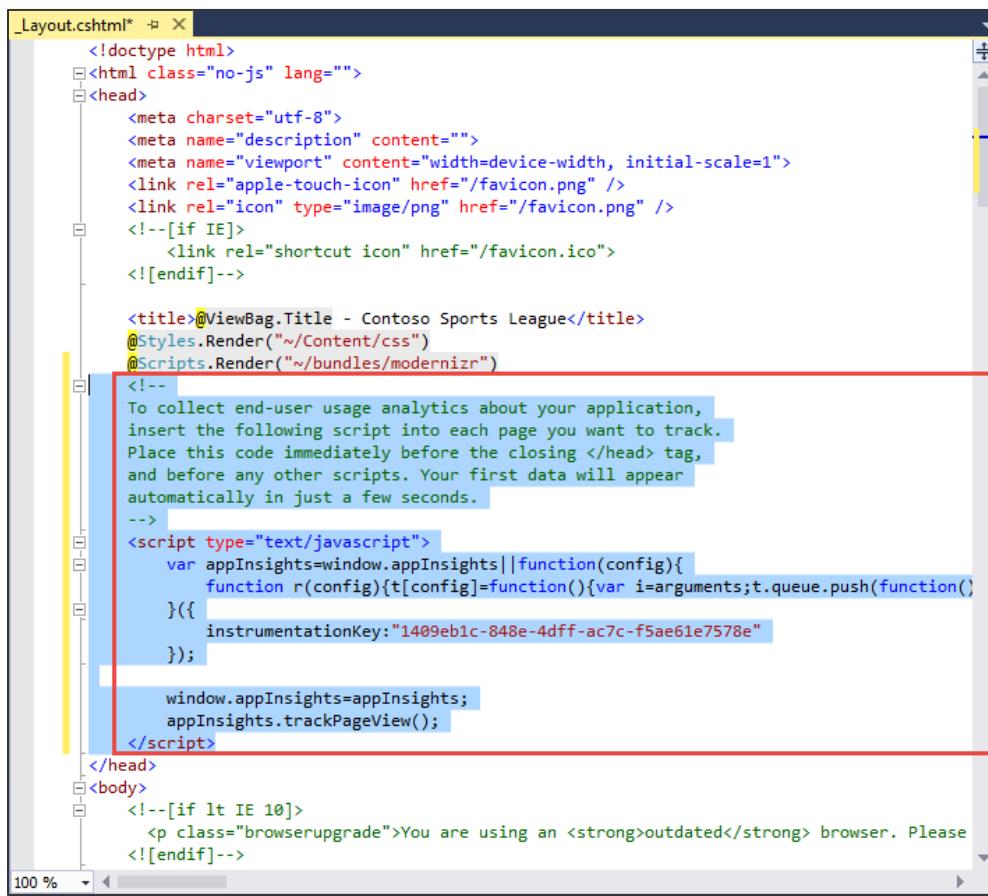
 Easy to get started. Simply paste the following into your master page

```
<!--  
To collect end-user usage analytics about your application,  
insert the following script into each page you want to track.  
Place this code immediately before the closing </head> tag,  
and before any other scripts. Your first data will appear  
automatically in just a few seconds.  
-->  
<script type="text/javascript">  
  var appInsights=window.appInsights||function(config){  
    function i(config){t[config]=function(){var i=arguments;t.queue.  
    "https://az416426.vo.msecnd.net/scripts/a/ai.0.js";u.getElementsByTa  
,i(c+a),i(h+v),i(c+v),i("flush"),config.disableExceptionTracking||(!r  
    }{  
      instrumentationKey:"25278dea-a476-4eca-a644-5c8dad345fa5"  
    });  
  
    window.appInsights=appInsights;  
    appInsights.trackPageView();
```

6. Navigate to the **Contoso.Apps.SportsLeague.Web** project located in the **Web** folder using the **Solution Explorer** in Visual Studio.
 7. Open **Views > Shared > _Layout.cshtml**.



- Paste in the code before the `</head>` tag.



```

_Layout.cshtml* > X
<!doctype html>
<html class="no-js" lang="">
<head>
    <meta charset="utf-8">
    <meta name="description" content="">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="apple-touch-icon" href="/favicon.png" />
    <link rel="icon" type="image/png" href="/favicon.png" />
    <!--[if IE]>
        <link rel="shortcut icon" href="/favicon.ico">
    <![endif]-->

    <title>@ViewBag.Title - Contoso Sports League</title>
    @Styles.Render("~/Content/css")
    @Scripts.Render("~/bundles/modernizr")

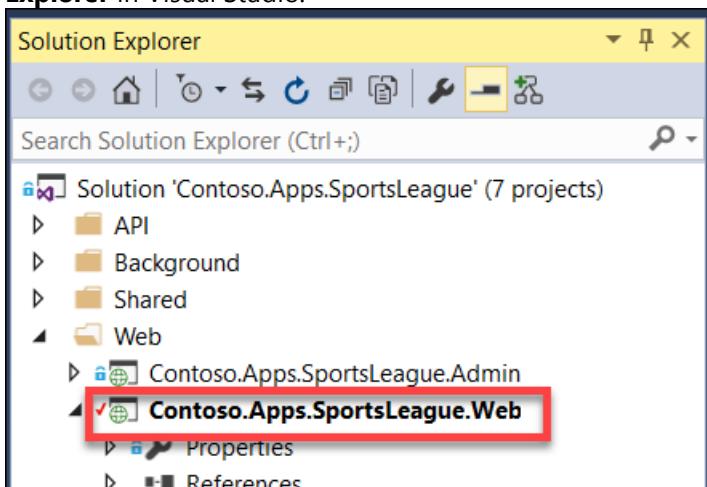
    <!--
        To collect end-user usage analytics about your application,
        insert the following script into each page you want to track.
        Place this code immediately before the closing </head> tag,
        and before any other scripts. Your first data will appear
        automatically in just a few seconds.
    -->
    <script type="text/javascript">
        var appInsights = window.appInsights || function(config){
            function r(config){t[config]=function(){var i=arguments;t.queue.push(function(){
                instrumentationKey: "1409eb1c-848e-4dff-ac7c-f5ae61e7578e"
            });
            window.appInsights = appInsights;
            appInsights.trackPageView();
        }
    </script>
</head>
<body>
    <!--[if lt IE 10]>
        <p class="browserupgrade">You are using an <strong>outdated</strong> browser. Please
    <![endif]-->

```

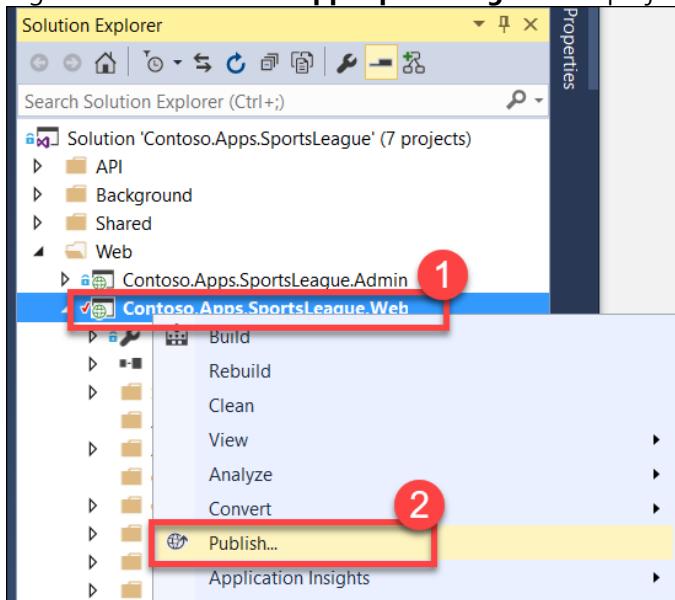
9. Save the **_Layout.cshtml** file.

Subtask 3: Deploy the e-commerce Web App from Visual Studio

1. Navigate to the **Contoso.Apps.SportsLeague.Web** project located in the **Web** folder using the **Solution Explorer** in Visual Studio.



2. Right-click the **Contoso.Apps.SportsLeague.Web** project, and select **Publish**.



3. Click **Publish** again when the Publish dialog appears.

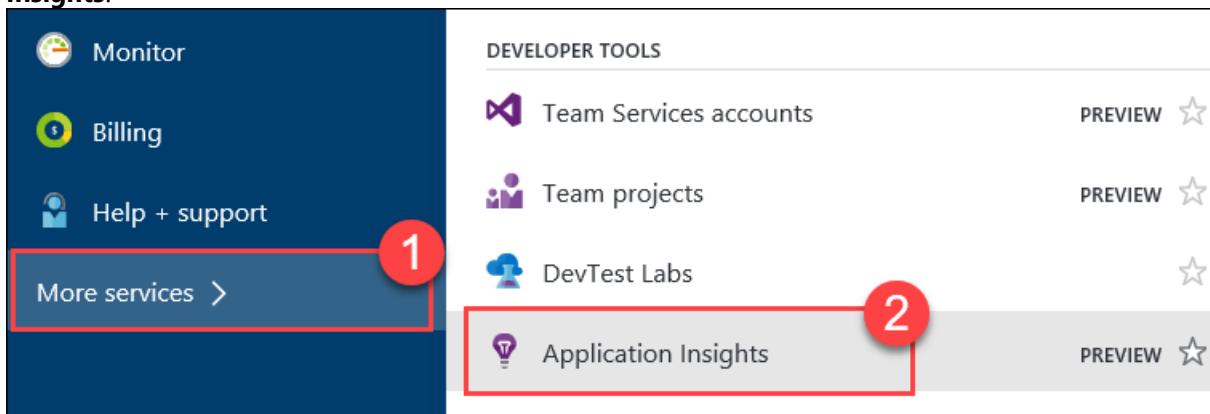
Launch a browser **outside of Visual Studio** for testing if the page is loaded in Visual Studio.

4. Click a few links on the published E-Commerce website, and submit several orders to generate some sample telemetry.

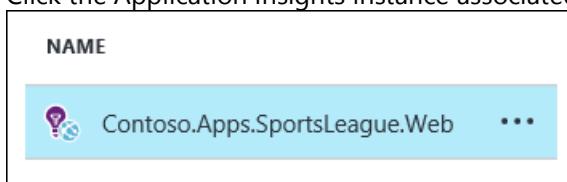
Task 2: Creating the web performance test and load test

Subtask 1: Create the load test

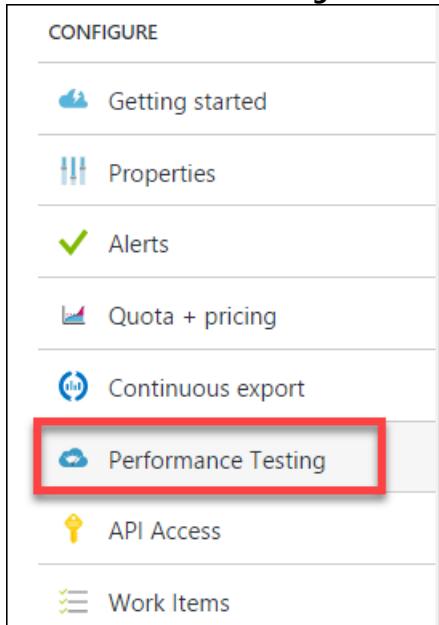
1. Open the Azure Management Portal (<http://portal.azure.com>). Click **More services** followed by **Application Insights**.



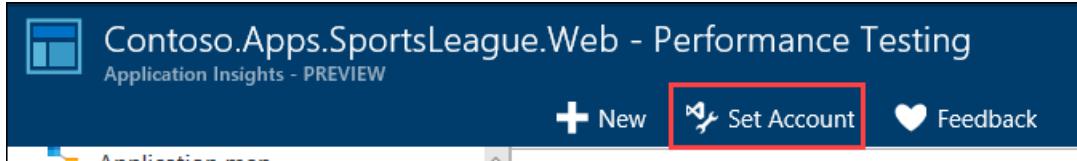
2. Click the Application Insights instance associated with the Contoso E-Commerce Site.



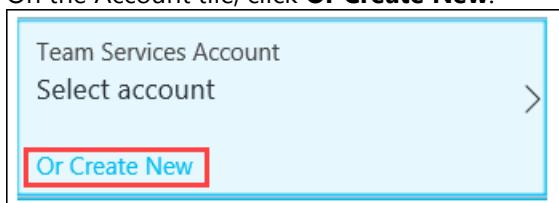
3. Click **Performance Testing**.



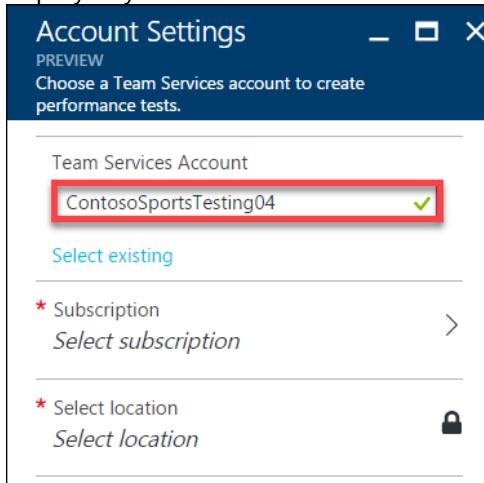
4. Click the Set Account button to associate/create a Visual Studio Team Services account.



5. On the Account tile, click **Or Create New**.



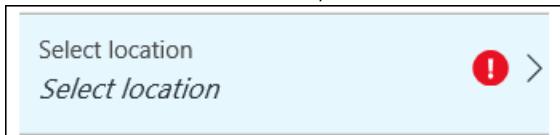
6. Specify a unique name for the account and select a region. Note the region may differ from the region you have deployed your resources.



7. Click **Subscription**, and select **your Subscription**.



8. Click **Select location**. Next, select a Location.



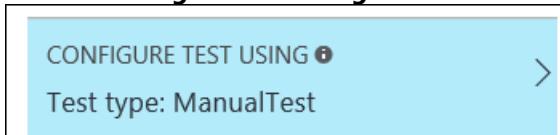
9. Then, click **OK**.

The VSTS account creation will take a minute to complete.

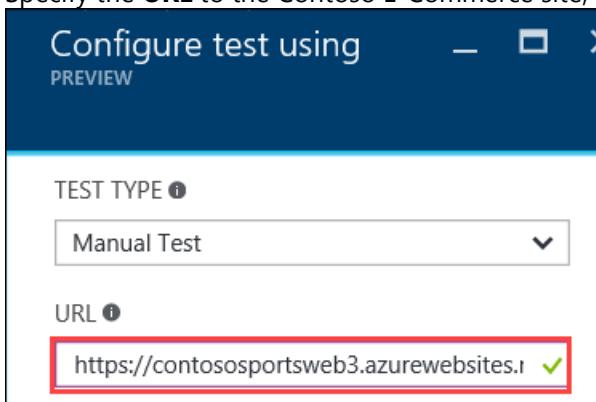
10. Click **New**.



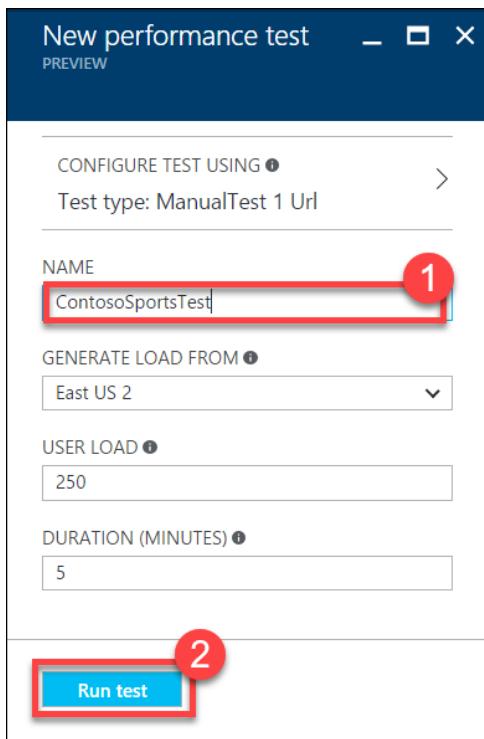
11. Click on **Configure Test Using**.



12. Specify the **URL** to the Contoso E-Commerce site, and click **Done**



13. Name the test **ContosoSportsTest**, and click the **Run test** button.

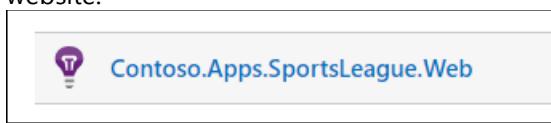


14. Wait until the load test has completed.

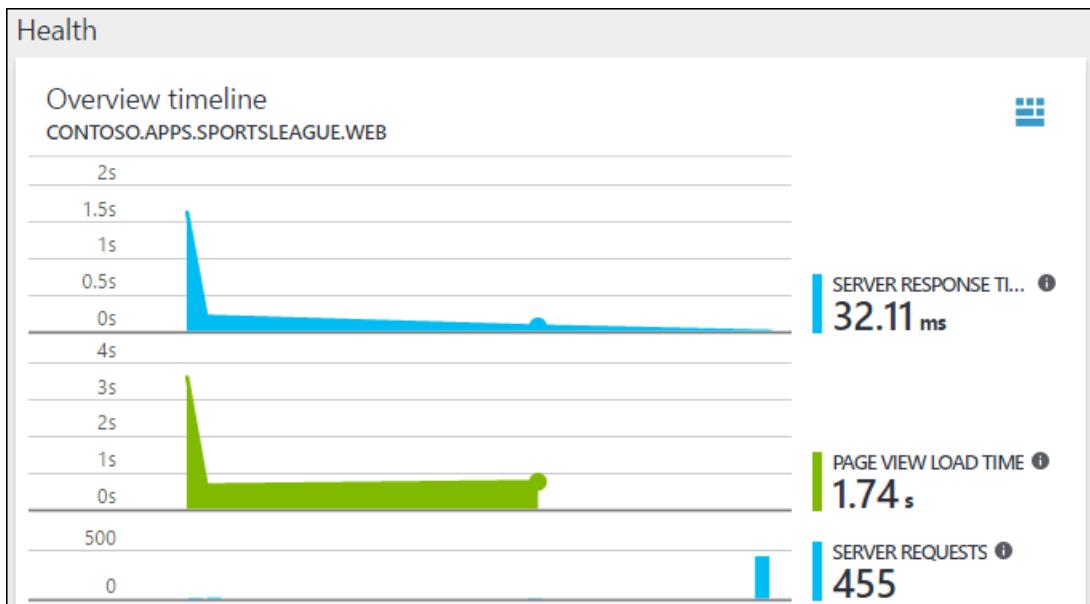
Recent runs				
NAME	STATE	START TIME	AVG RESP TIME (SEC)	TARGET LOAD
ContosoSportsTest	Completed	1/25/2016 4:19 PM	17.16	250

Subtask 2: View the Application Insights logs

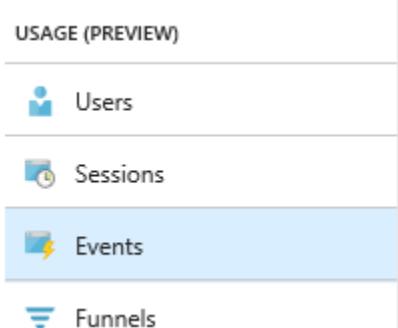
1. Using a new tab or instance of your browser, navigate to the Azure Management portal <http://portal.azure.com>.
2. On the left menu area, click **More services**.
3. On the **More services** blade, select **Application Insights**.
4. On the **Application Insights** blade, select the Application Insights configuration you created for the e-commerce website.



5. View the performance timeline to see the overall number of requests and page load time.



- Under **Usage Preview**, Check out the Events Button.



- After several minutes, you should see several Custom events from your previous order testing. This is reported through the TelemetryClient's TrackEvent method.

Note: If you do not see data here, come back later after the lab is complete.

Custom events		
OrderCompleted	<div style="width: 100%;"> </div>	3
SuccessfulPaymentAuth	<div style="width: 100%;"> </div>	3

- Drilling into the OrderCompleted events provides you with more detail about the specific order.

The screenshot shows a Microsoft Cloud Workshop application window titled "OrderCompleted" under "Contoso.Apps.SportsLeague.Web". The window has a dark blue header bar with a "New Work Item" button and a "View Work Items" link. Below the header, there are three main sections: "Custom Event Properties", "Custom Data", and "Related Items".

Custom Event Properties:

Event time	1/13/2017, 3:00:20 PM	...
Event name	OrderCompleted	...
Device type	PC	...
...		

Custom Data:

OrderTotal	\$52.97	...
PaymentTransactionId	hpzPXxjzn2rS	...

Related Items:

Request in which this custom event was logged	1	1
All available telemetry for this user session	128	1
All available telemetry for this operation	10	1
All available telemetry 5 minutes before and after this event	129	1

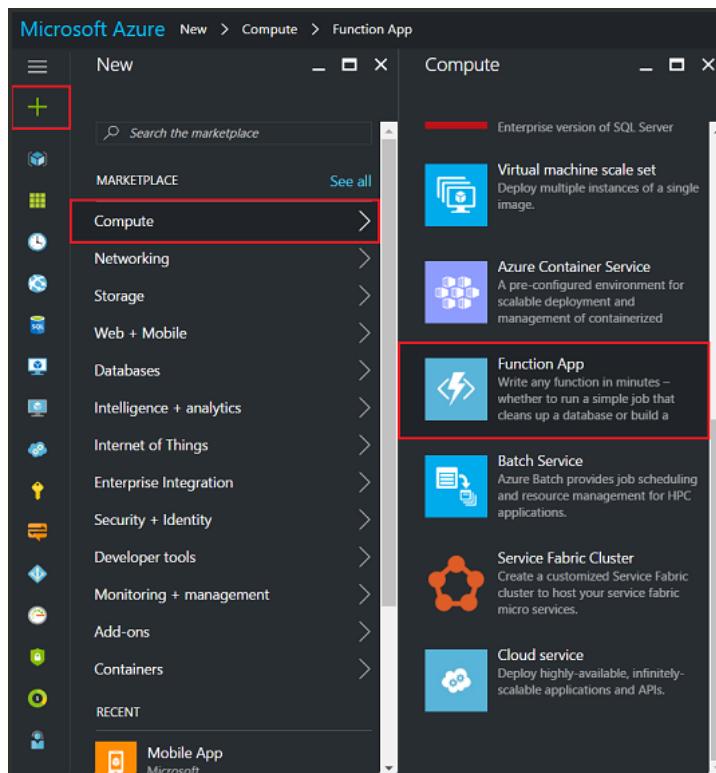
Exercise 5: Automating backend processes with Azure Functions and Logic Apps

Contoso wants to automate the process of generating receipts in PDF format and alerting users when their orders have been processed using Azure Logic App and Functions. To run custom snippets of C# or node.js in logic apps, you can create custom functions through Azure Functions. [Azure Functions](#) offers server-free computing in Microsoft Azure and are useful for performing these tasks:

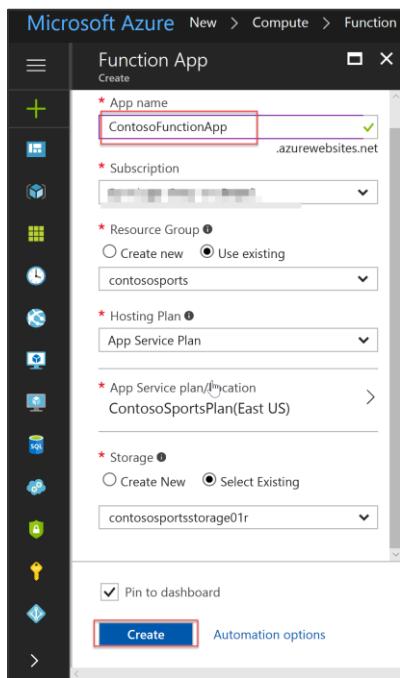
- Advanced formatting or compute of fields in logic apps
- Perform calculations in a workflow.
- Extend the logic app functionality with functions that are supported in C# or node.js

Task 1: Create an Azure Function to Generate PDF Receipts

1. Click the New button found on the upper left-hand corner of the Azure portal and then click **Compute > Function App**, select your Subscription, type a unique App name that identifies your function app, then specify the following settings:
 - **Resource Group:** Use the existing resource group for **contososports**.
 - **Hosting plan,** which can be one of these plans:
 - **Consumption plan:** The default plan type for Azure Functions. When you choose a consumption plan, you must also choose the **Location**.
 - **App Service plan:** An App Service plan requires you to create an **App Service plan/location** or select an existing one. These settings determine the location, features, cost, and compute resources associated with your app.
 - **Storage account:** Each function app requires a storage account. Choose the existing storage account by clicking **Select Existing** and choosing the storage account in the contososports resource group.



2. Click **Create** to provision and deploy the new function app.



3. Open the Function App you just created. Click the **+** beside **Functions**, scroll down, and select **Custom function**.

1. Choose a scenario

2. Choose a language

For F#, PowerShell, Python, and Batch, [create your own custom function](#).

Create this function

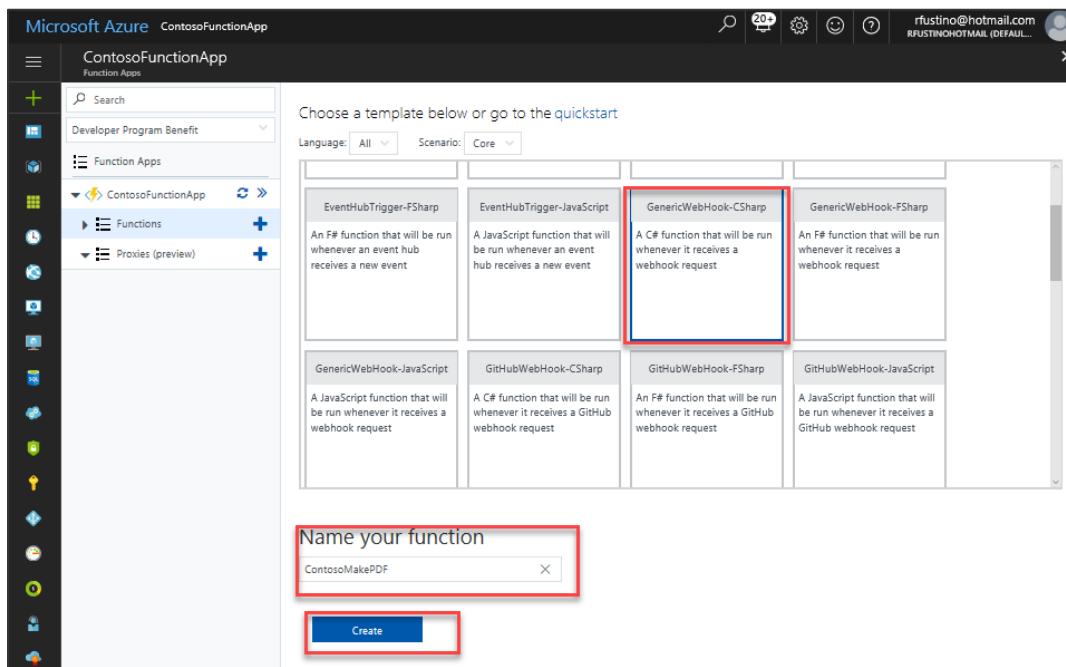
or

Get started on your own

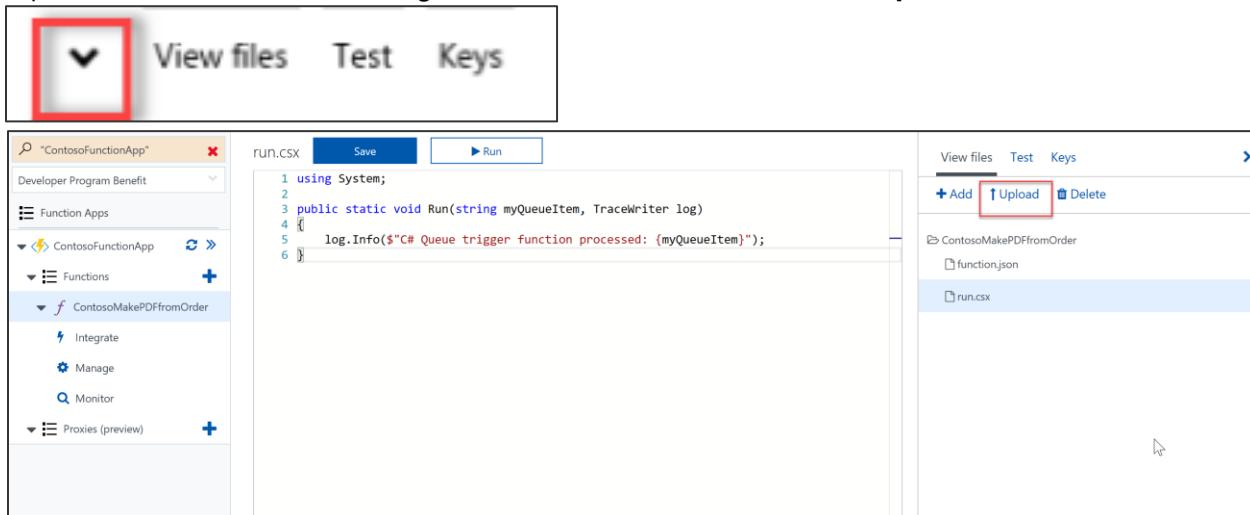
Custom function

Start from source control

4. Select **GenericWebHook-CSharp** with a Name of **ContosoMakePDF**, and press **Create**.

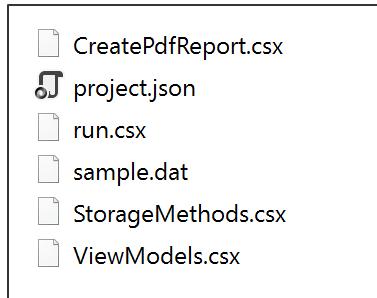


5. Expand the View files area on the right of the code window and then click **Upload**.



6. Upload the following files in the (Contoso Sports League\Contoso.CreatePDFReport) folder beneath:
C:\Hackathon.

- ViewModels.csx
- CreatePdfReport.csx
- run.csx
- sample.dat
- StorageMethods.csx
- Project.json



7. Click on run.csx, to refresh the code editor.

```

1 #load "C:\Windows\Microsoft.NET\assembly\GAC_MSIL\System.IO\version\4.0_4.0.0.0__31bf3856ad364e35\System.IO.dll"
2 #load "C:\Windows\Microsoft.NET\assembly\GAC_MSIL\System\version\4.0_4.0.0.0__31bf3856ad364e35\System.dll"
3 #load "C:\Windows\Microsoft.NET\assembly\GAC_MSIL\System.Core\version\4.0_4.0.0.0__31bf3856ad364e35\System.Core.dll"
4 #load "C:\Windows\Microsoft.NET\assembly\GAC_MSIL\System.Numerics\version\4.0_4.0.0.0__31bf3856ad364e35\System.Numerics.dll"
5 using Newtonsoft.Json;
6 using System;
7 using System.Net;
8
9 public static void Run(string myQueueItem, TraceWriter log)
10 {
11     OrderViewModel Order = null;
12
13     string jsonContent = myQueueItem;
14     dynamic payload = JsonConvert.DeserializeObject(jsonContent);
15     if (payload != null && payload.Order != null)
16     {
17         var base64EncodedData = payload.Order.Value;
18         var base64EncodedBytes = System.Convert.FromBase64String(base64EncodedData);
19         Order = JsonConvert.DeserializeObject<OrderViewModel>(System.Text.Encoding.UTF8.GetString(base64EncodedBytes));
20     }
21
22     if (Order == null || Order.OrderId <= 0)
23     {
24         //error handling code here
25     }
26
27     log.Info($"Webhook was triggered! Order: {Order.OrderId}");
28
29     ProcessOrder(Order, log);
30
31 }
32
33 static OrderViewModel ProcessOrder(OrderViewModel Order, TraceWriter log)
34 {
35     string fileName = string.Format("ContosoSportsLeague-Store-Receipt-{0}.pdf", Order.OrderId);
36     log.Info($"Using Filename {fileName}");
37
38     var receipt = CreatePdfReport(Order, fileName);
39 }
```

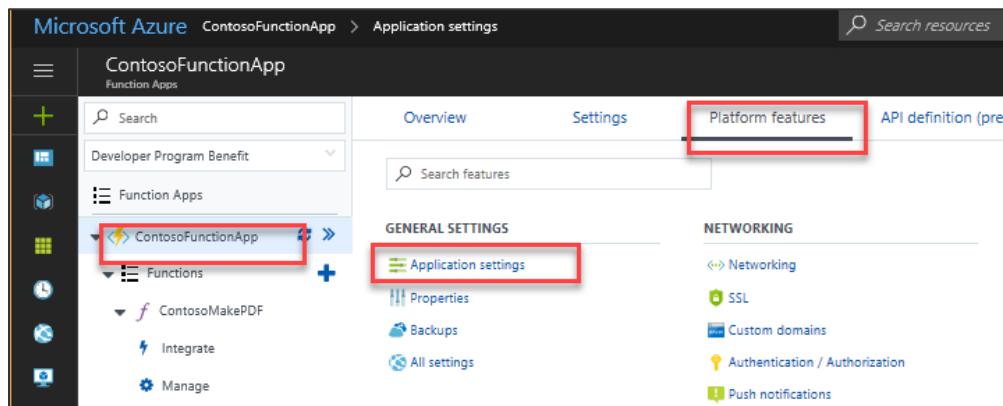
8. Open the Log windows on the bottom.

The screenshot shows the Azure Functions developer portal. At the top, there's a code editor window titled "run.csx" containing C# code for a function named "ContosoMakePDF". Below the code editor is a "Logs" panel. At the top of the logs panel, there are several buttons: "Pause", "Clear", "Copy log", and "Expand" (which is highlighted with a red box). The logs panel itself is currently empty, showing only the header row.

```
1 #Load "StorageMethods.csx"
2 #Load "ViewModels.csx"
3 #Load "CreatePdfReport.csx"
4 #r "Newtonsoft.Json"
5
6
7 using Newtonsoft.Json;
8 using System;
9 using System.Net;
10
11 public static async Task<object> Run(HttpRequestMessage req, TraceWriter log)
12 {
13     OrderViewModel Order = null;
14
15     try
16     {
17         log.Info("Webhook was triggered! Order");
18         string jsonContent = await req.Content.ReadAsStringAsync();
19         dynamic payload = JsonConvert.DeserializeObject(jsonContent);
20         log.Info("Payload received");
21         if (payload != null && payload.Order != null)
22         {
23             log.Info("payload not null! Order");
24             var base64EncodedData = payload.Order.Value;
25             var base64EncodedBytes = System.Convert.FromBase64String(base64EncodedData);
```

Note: You should see several messages about downloading dependent assemblies such as the Azure SDK and iText Sharp that were defined in the project.json file.

9. Select the name of your function app, and then click on **Platform Features** followed by **Application settings**.



10. Add a new entry called **contososportsstorage**, and paste the value of the connection string noted in an earlier exercise. Click **Save** after adding the value.

The screenshot shows the 'App settings' section in the Azure portal. A new entry 'contososportsstorage' has been added, highlighted with a red box. The 'Value' field contains a long connection string. Other settings listed include 'AzureWebJobsDashboard', 'AzureWebJobsStorage', 'FUNCTIONS_EXTENSION_VERSION', 'WEBSITE_NODE_DEFAULT_VERSION', and others.

Note: You can find the value by opening the storage account, and clicking the Access Keys tile.

11. Open the **sample.dat** file, and select as well as copy (Ctrl+C) the test data.

The screenshot shows the 'sample.dat' file content in the Azure portal. The 'Test' tab is selected, displaying a JSON object with an 'Order' key containing a long string of characters. This string is highlighted with a red box.

12. Select the **Run.csx** file, click on the **Test** tab, and replace the contents by pasting (CTRL-V) in the Test tab Request Body.

The screenshot shows the 'Run.csx' file content in the Azure portal. The 'Test' tab is selected, showing a 'Request body' with the same JSON object as the 'sample.dat' file. The code in the main editor is related to processing queue items.

13. Select the **View Files** tab, select **Run.csx**, and click run.

14. You should see messages in the Logs window stating the Webhook was triggered, and the PDF was generated / saving it the storage account. Also, you should see that actual message text in the Output Window.

The screenshot shows the Azure Functions developer portal interface. On the left, the code editor displays `run.csx` with C# code for processing orders. In the center, the 'Test' tab is selected, showing a successful POST request with a JSON payload and a response body containing a JSON object representing an order. Below the test results, the 'Logs' section shows detailed logs of the function's execution, including the creation of a PDF and its upload to blob storage. The 'Output' section at the bottom right shows the status as 200 OK.

```

43     catch (Exception ex)
44     {
45         log.Info($"Exception {ex}");
46         return req.CreateResponse(HttpStatusCode.InternalServerError, new
47         {
48             error = string.Format("Error Processing Order: {0}", ex.Message)
49         });
50     }
51 }
52 }
53
54 static OrderViewModel ProcessOrder(OrderViewModel Order, TraceWriter log)
55 {
56     string fileName = string.Format("ContosoSportsLeague-Store-Receipt-{0}.pdf", Order.Id);
57
58     var receipt = CreatePdfReport(Order, fileName, log);
59     log.Info("PDF generated. Saving to blob storage...");
60     Order.ReceiptUrl = UploadPdfToBlob(receipt, fileName, log);
61
62     return Order;
63 }
64 }

```

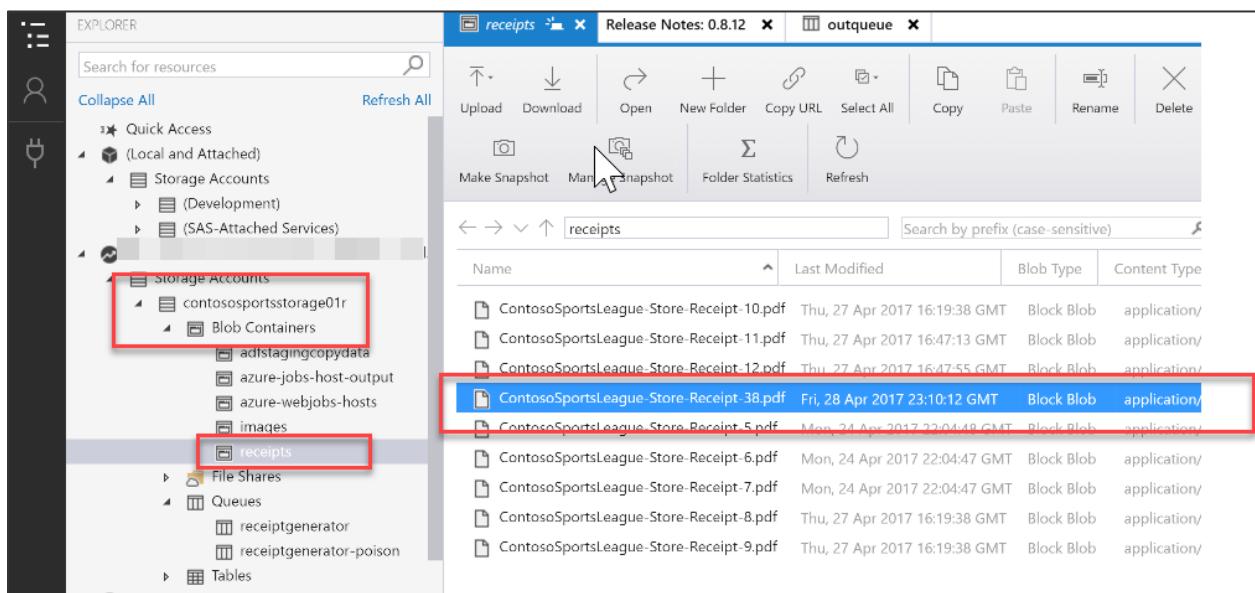
Output: Status: 200 OK

```

{
    "OrderId": 38,
    "OrderDate": "2017-03-10T19:48:02.62",
    "FirstName": "Bob",
    "LastName": "Loblaw",
    "Address": "1313 Mockingbird Lane",
    "City": "Virginia Beach",
    "State": "VA",
    "PostalCode": "23456",
    "Country": "United States",
    "Phone": "+55512345678",
    "Email": "bobloblaw@contosportsleague.com",
    "ReceiptUrl": "https://contosportsstorage01.blob.core.windows.net/receipts/ContosoSportsLeague-Store-Receipt-38.pdf?sv=2016-05-31&r=b&sig=oLaYYfcfK1sEB%2FwSTxp%2BV2EUTPxbit17kdzCKpTKx3o&se=2024-04-30T18%3A20%3A29Z&spr=r",
    "SMSSoptin": false,
    "SMSstatus": null,
    "PaymentTransactionId": null,
    "HasBeenShipped": false,
    "Total": 879.45,
    "OrderDetails": null
}

```

15. To see the PDF indeed landed in the receipts container in blob storage, download the Microsoft Storage Explorer at <http://storageexplorer.com>. Use Microsoft Storage explorer to verify the PDF landed on the Blob Container for receipts. You may need to refresh and/or select another folder, and arrive back to the receipts folder to see the PDF.



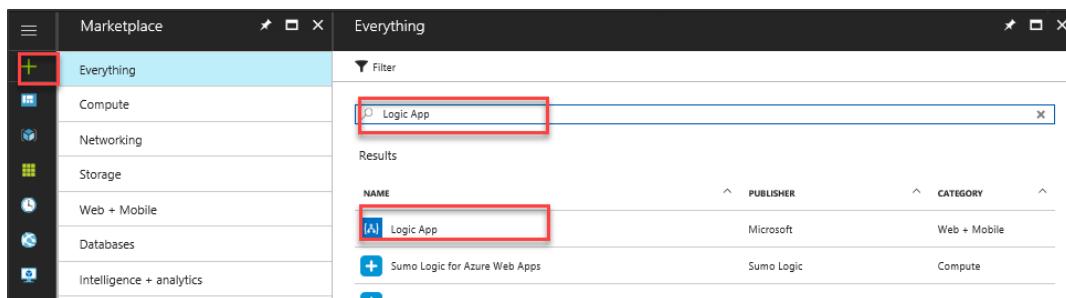
Task 2: Create an Azure Logic App to Process Orders

Without writing any code, you can automate business processes more easily and quickly when you create and run workflows with Azure Logic Apps. Logic Apps provide a way to simplify and implement scalable integrations and workflows in the cloud. It provides a visual designer to model and automate your process as a series of steps known as a workflow. There are [many connectors](#) across the cloud and on-premises to quickly integrate across services and protocols.

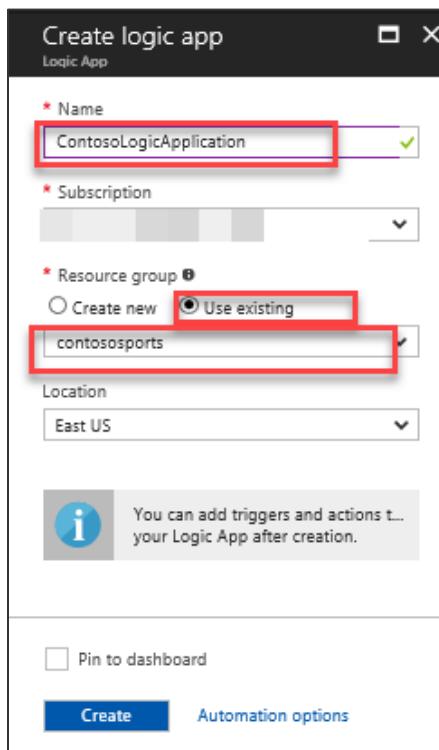
The advantages of using Logic Apps include the following:

- Saving time by designing complex processes using easy to understand design tools
- Implementing patterns and workflows seamlessly, that would otherwise be difficult to implement in code
- Getting started quickly from templates
- Customizing your logic app with your own custom APIs, code, and actions
- Connect and synchronize disparate systems across on-premises and the cloud
- Build off of BizTalk server, API Management, Azure Functions, and Azure Service Bus with first-class integration support

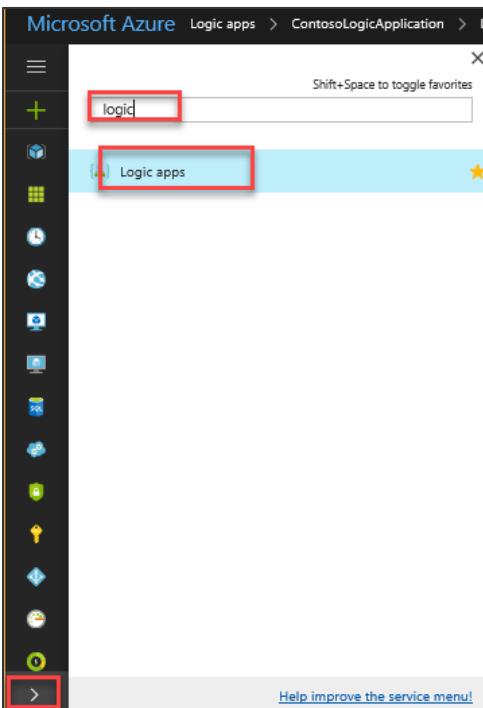
1. Next, let us create a Logic App that will trigger when an item is added to the receiptgenerator queue. In the Azure Management Portal, click the + button, search for **Logic App**, click the returned Logic App result, and click **Create**.



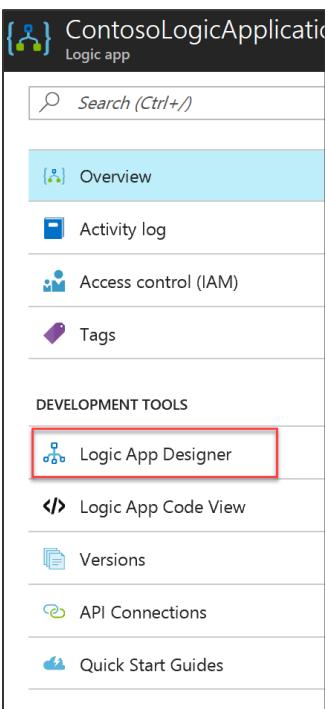
2. Fill out the name as **ContosoLogicApplication** along with your subscription, and use the existing resource group **contososports**. Choose the **same region** as your Web App and storage account. Click **Create**.



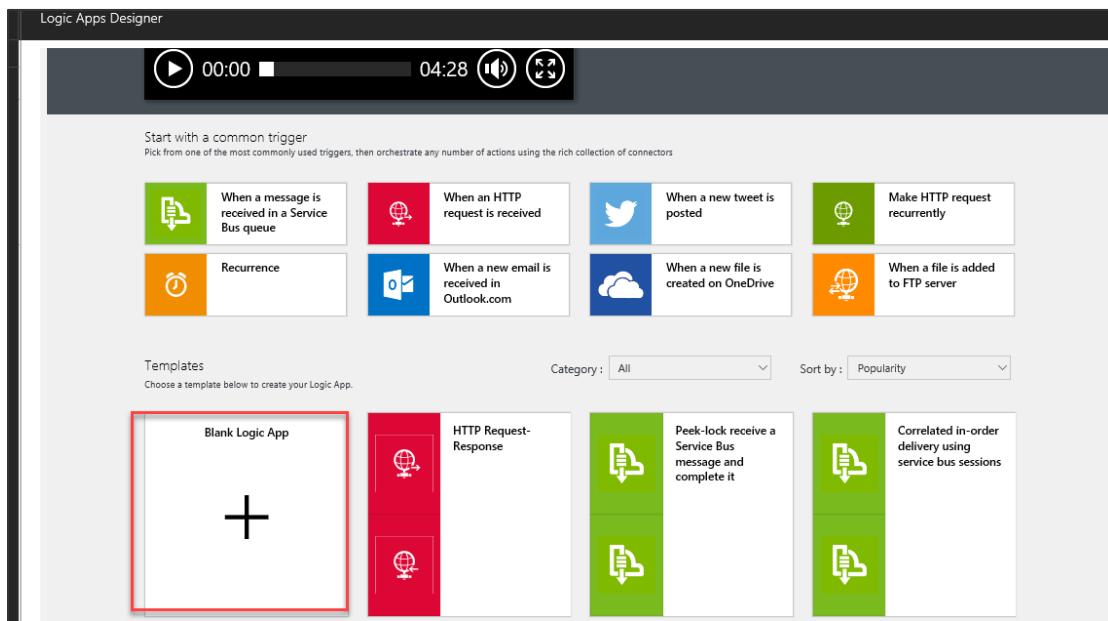
3. Open up the logic app after it is deployed by clicking more services and search on logic.



4. Click on the **Logic App Designer** link.



5. In the Logic Apps Designer, select **Blank Logic App**.



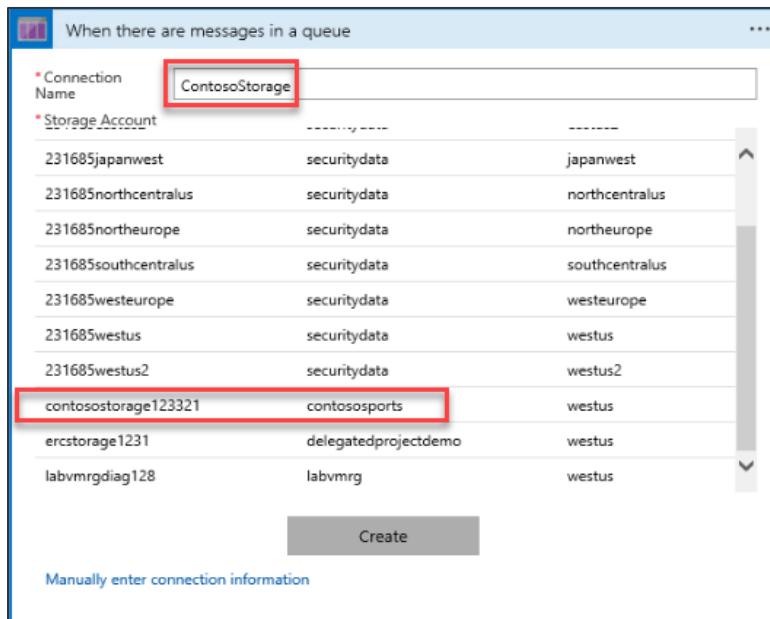
6. Select **Azure Queues**.

The screenshot shows the Azure portal's search bar at the top. Below it, the 'SERVICES' section displays various icons for services like Request / Response, Schedule, Service Bus, Twitter, Office 365 Outlook, SharePoint, FTP, Dynamics 365, SFTP, Salesforce, RSS, OneDrive, Dropbox, and Azure Queues. The 'Azure Queues' icon is highlighted with a red box. Below the services is the 'TRIGGERS (147)' section, which lists several triggers including '10to8 Appointment Scheduling - When a booking is made', 'appFigures - When there is a new review', 'Asana - When a project is created', 'Azure API Management - Choose an Azure API Management trigger', 'Azure App Services - Choose an Azure App Services trigger', 'Azure Blob Storage - When one or more blobs are added or modified (metadata only)', 'Azure Queues - When there are messages in a queue', and 'Basecamp 2 - When a document is created'. Each trigger item includes a preview link and an information icon.

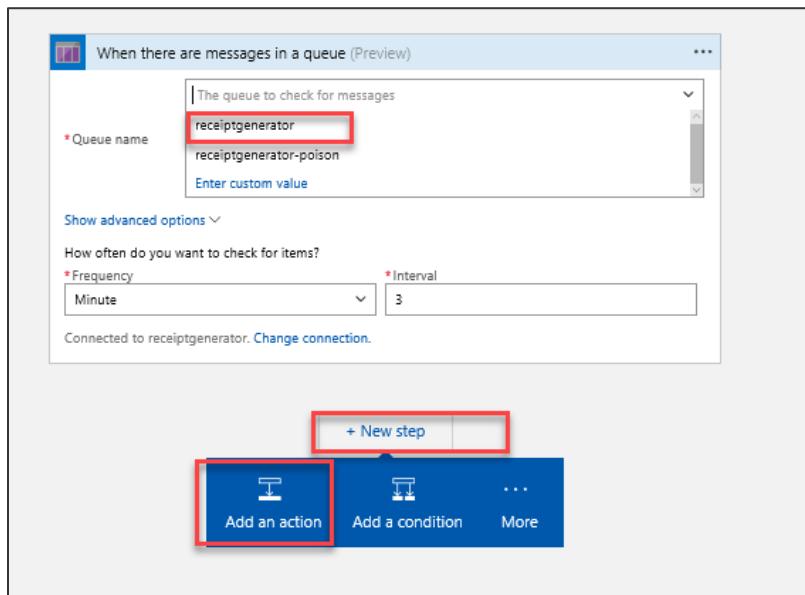
7. Select **Azure Queues – When there are messages in a queue**.

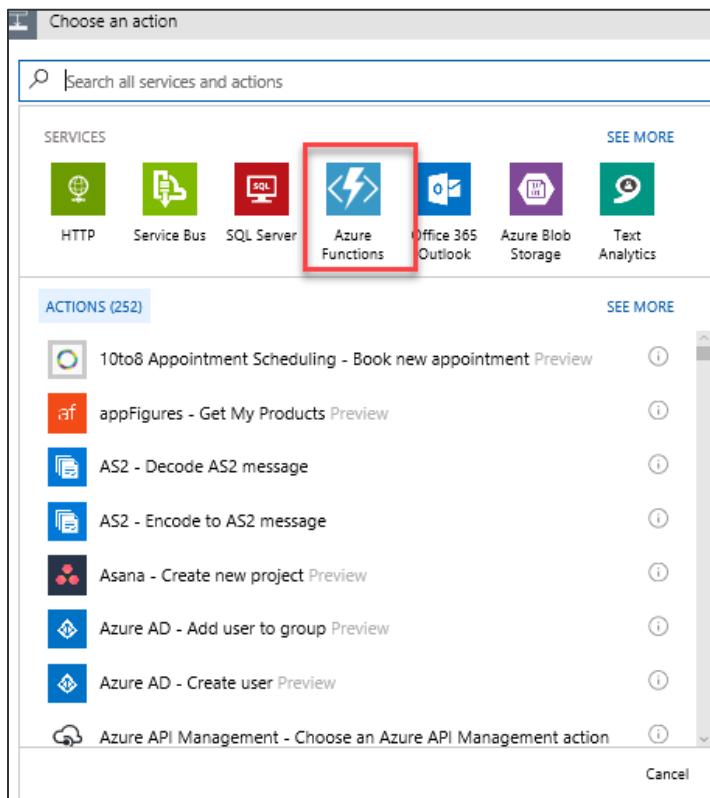
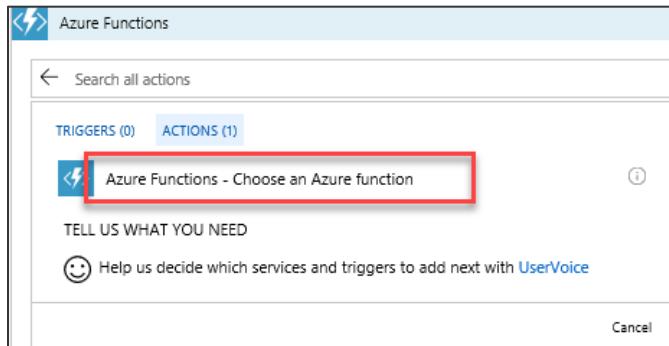
The screenshot shows the 'TRIGGERS (2)' section of the Azure portal. It lists two triggers: 'Azure Queues - When there are messages in a queue' (highlighted with a red box) and 'Azure Queues - When a specified number of messages are in a given queue'. Below the triggers is a 'TELL US WHAT YOU NEED' section with a 'UserVoice' link. There is also a smiley face icon and a 'Help us decide which services and triggers to add next with UserVoice' message.

8. Specify **ContosoStorage** as the connection name, select the Contoso storage account from the list, and click **Create**.

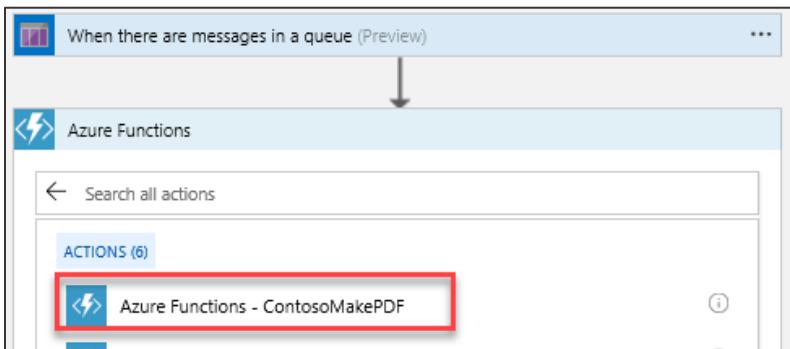


9. Select the **receiptgenerator** queue from the drop-down, click **New Step**, and **Add an Action**.



10. Select **Azure Functions.****11. Click **Azure Functions – Choose an Azure function**.**

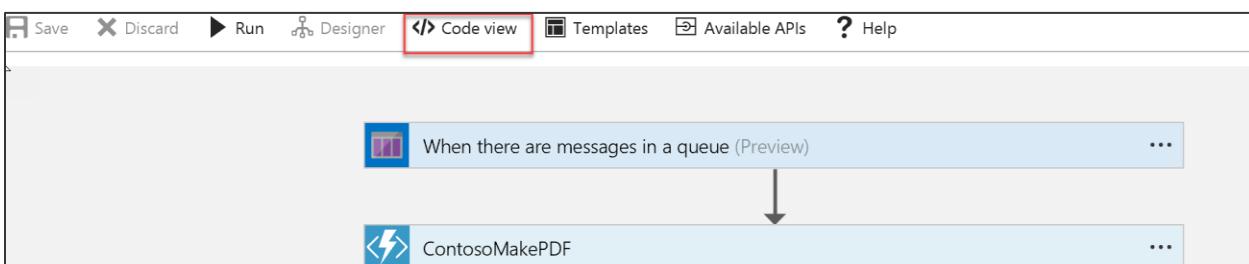
12. Select the Azure function created earlier followed by selecting the **ContosoMakePDF** function.



13. Type this in the Request Body {"**Order**": [pick MessageText from list on right] }. Make sure the syntax is json format. Sometimes the ":" will go to the right side of MessageText by mistake. Keep it on the left. It should look like this:

14. Click **Save** to save the Logic App.

15. There is one modification we need to make in the code. Click on the **CodeView** button.



15. Find the line of code in the body for the Order item that reads the MessageText value from the queue, and add the base64 function around it to ensure it is encoded before passing it off to the Azure function. It should look like the following:

```
"Order": "@{base64(triggerBody()?['MessageText'])})"
```

```
definition : {
    "$schema": "https://schema.management.azure.com/providers/Microsoft.Logic/schemas/2016-06-01/workflowdefinition.json#",
    "actions": {
        "ContosoMakePDF": {
            "inputs": {
                "body": {
                    "Order": "@{base64(triggerBody()?['MessageText'])})"
                }
            }
        }
    }
}
```

16. Run the logic app. It should process the orders you have submitted previously to test PDF generation. Using Azure Storage Explorer or Visual Studio Cloud Explorer you can navigate to the storage account and open the receipts container to see the created PDFs.

Name	Last Modified	Blob Type	Content Type
ContosoSportsLeague-Store-Receipt-10.pdf	Thu, 27 Apr 2017 16:19:38 GMT	Block Blob	application/octe
ContosoSportsLeague-Store-Receipt-11.pdf	Thu, 27 Apr 2017 16:47:13 GMT	Block Blob	application/octe
ContosoSportsLeague-Store-Receipt-12.pdf	Thu, 27 Apr 2017 16:47:55 GMT	Block Blob	application/octe
ContosoSportsLeague-Store-Receipt-38.pdf	Sun, 30 Apr 2017 18:20:28 GMT	Block Blob	application/octe
ContosoSportsLeague-Store-Receipt-5.pdf	Mon, 24 Apr 2017 22:04:48 GMT	Block Blob	application/octe
ContosoSportsLeague-Store-Receipt-6.pdf	Mon, 24 Apr 2017 22:04:47 GMT	Block Blob	application/octe
ContosoSportsLeague-Store-Receipt-7.pdf	Mon, 24 Apr 2017 22:04:47 GMT	Block Blob	application/octe
ContosoSportsLeague-Store-Receipt-71.pdf	Sun, 30 Apr 2017 18:51:03 GMT	Block Blob	application/octe
ContosoSportsLeague-Store-Receipt-72.pdf	Sun, 30 Apr 2017 18:51:03 GMT	Block Blob	application/octe
ContosoSportsLeague-Store-Receipt-8.pdf	Thu, 27 Apr 2017 16:10:39 GMT	Block Blob	application/octe

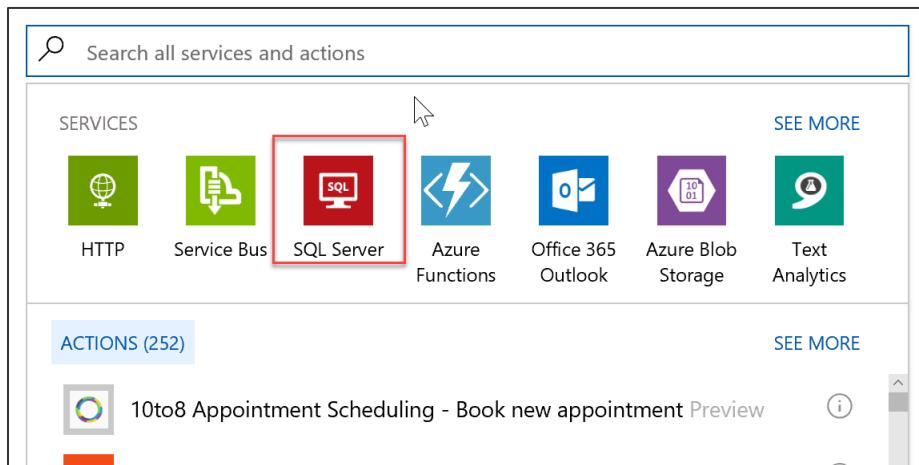
17. Double click it to see the Purchase receipt.

Product	Quantity	Unit Price	Total
Product A	1	\$1,250.99	\$1,250.99

Total \$1,250.99

17. Now, add two more steps to the flow for updating the database and removing the message from the queue after it has been processed. Switch back to the designer, click **+ New Step** and select **Add an Action**.

18. Select **SQL Server**.



19. Select **SQL Server - Update row**.

The screenshot shows the "SQL Server" actions list. At the top, there is a search bar labeled "Search all actions". Below it, there are tabs for "TRIGGERS (0)" and "ACTIONS (8)", with "ACTIONS (8)" being the active tab. The actions listed are: SQL Server - Delete row, SQL Server - Execute stored procedure, SQL Server - Get row, SQL Server - Get rows, SQL Server - Insert row, SQL Server - Update row, SQL Server - Get tables, and SQL Server - GetTables with OData paging and filtering. The "SQL Server - Update row" action is highlighted with a red box.

20. Name the connection ContosoSportsDB, and select the primary ContosoSportsDB database for your solution. Under the user name and password used to create it, click **Create**.

The screenshot shows the 'Update row' dialog for a SQL database connection. At the top, it says 'ContosoSportsDB'. Below that, there's a table with columns 'Name', 'Resource Group', and 'Location'. It lists two entries: 'ContosSportsDB' (Resource Group: contososports, Location: West US) and 'master' (Resource Group: contososports, Location: West US). There are fields for 'Username' and 'Password' (redacted), and a checkbox for 'Connect via on-premise data gateway'. A large red button at the bottom right says 'Create'.

21. From the drop-down select the name of the table, **Orders**.

The screenshot shows the 'Update row' dialog with the 'Table name' field highlighted. A dropdown menu lists several table names: CartItems, Categories, OrderDetails, Orders (which is selected and highlighted with a red box), Products, _MigrationHistory, and Enter custom value. Below the dropdown, there's a 'Row id' field and a note saying 'Connected to ContosoDB. Change connection.'

22. Press **save** and ignore the error. Navigate to the code view.

23. Replace these lines:

```
"Address": null,
"City": null,
"Country": null,
"Email": null,
"FirstName": null,
"HasBeenShipped": null,
"LastName": null,
"OrderDate": null,
"Phone": null,
"PostalCode": null,
"SMSOptIn": null,
"State": null,
"Total": null
```

With these:

```
"OrderDate": "@{body('ContosoMakePDF')['OrderDate']}",
"FirstName": "@{body('ContosoMakePDF')['FirstName']}",
"LastName": "@{body('ContosoMakePDF')['LastName']}",
"Address": "@{body('ContosoMakePDF')['Address']}",
"City": "@{body('ContosoMakePDF')['City']}",
"State": "@{body('ContosoMakePDF')['State']}",
"PostalCode": "@{body('ContosoMakePDF')['PostalCode']}",
"Country": "@{body('ContosoMakePDF')['Country']}",
"Phone": "@{body('ContosoMakePDF')['Phone']}",
"SMSOptIn": "@{body('ContosoMakePDF')['SMSOptIn']}",
"SMSStatus": "@{body('ContosoMakePDF')['SMSStatus']}",
"Email": "@{body('ContosoMakePDF')['Email']}",
"ReceiptUrl": "@{body('ContosoMakePDF')['ReceiptUrl']}",
"Total": "@{body('ContosoMakePDF')['Total']}",
"PaymentTransactionId": "@{body('ContosoMakePDF')['PaymentTransactionId']}",
"HasBeenShipped": "@{body('ContosoMakePDF')['HasBeenShipped']}
```

24. And modify the path variable to include the index key or OrderId to be as follows:

```
"path":
"/datasets/default/tables/@{encodeURIComponent(encodeURIComponent('[dbo].[Orders]'))}/items/@{encodeURIComponent(encodeURIComponent(body('ContosoMakePDF')['OrderId']))}"
```

The code should now look as follows for the update_row method:

```
"Update_row": {
  "inputs": {
    "body": {
      "OrderDate": "@{body('ContosoMakePDF')['OrderDate']}",
      "Address": "@{body('ContosoMakePDF')['Address']}",
      "City": "@{body('ContosoMakePDF')['City']}",
      "Country": "@{body('ContosoMakePDF')['Country']}",
      "Email": "@{body('ContosoMakePDF')['Email']}",
      "FirstName": "@{body('ContosoMakePDF')['FirstName']}",
      "LastName": "@{body('ContosoMakePDF')['LastName']}",
      "HasBeenShipped": "@{body('ContosoMakePDF')['HasBeenShipped']}",
      "PaymentTransactionId": "@{body('ContosoMakePDF')['PaymentTransactionId']}",
      "Phone": "@{body('ContosoMakePDF')['Phone']}",
      "PostalCode": "@{body('ContosoMakePDF')['PostalCode']}",
      "ReceiptUrl": "@{body('ContosoMakePDF')['ReceiptUrl']}",
      "SMSOptIn": "@{body('ContosoMakePDF')['SMSOptIn']}",
      "SMSStatus": "@{body('ContosoMakePDF')['SMSStatus']}",
      "State": "@{body('ContosoMakePDF')['State']}",
      "Total": "@{body('ContosoMakePDF')['Total']}"
    },
    "host": {
      "api": {
        "runtimeUrl": "https://logic-apis-eastus.azure-apim.net/apim/sql"
      },
      "connection": {
        "name": "@parameters('$connections')['sql']['connectionId']"
      }
    },
    "method": "patch",
    "path": "/datasets/default/tables/@{encodeURIComponent(encodeURIComponent('[dbo].[Orders]'))}/items/@{encodeURIComponent(encodeURIComponent(body('ContosoMakePDF')['OrderId']))}"
  },
  "runAfter": {
    "ContosoMakePDF": [
      "Succeeded"
    ]
  }
}
```

25. **Save** and return to the designer.

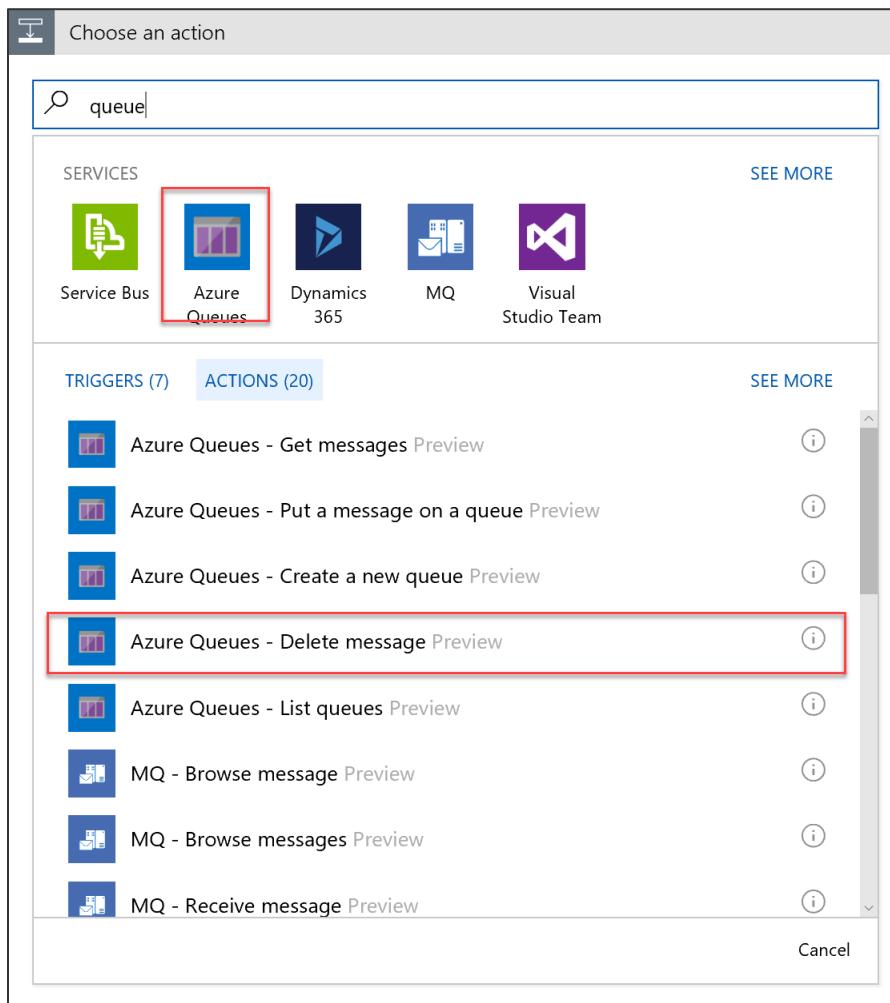
26. Your updated designer view should look like this:

Update row

* Table name	Orders
* Row id	OrderId x
* OrderDate	OrderDate x
* FirstName	FirstName x
* LastName	LastName x
* Address	Address x
* City	City x
* State	State x
* PostalCode	PostalCode x
* Country	Country x
* Phone	Phone x
* SMSOptIn	SMSOptIn x
* Email	Email x
* Total	Total x
* HasBeenShipped	HasBeenShipped x
SMSStatus	SMSStatus x
ReceiptUrl	ReceiptUrl x
PaymentTransactionId	PaymentTransactionId x

Connected to ContosoDB. [Change connection.](#)

27. Finally, let us add one more step to remove the message from the queue. Press **+New Step** and **Add an Action**. Type in Queue in the search box, and select Azure Queues – Delete message.



28. Select the **receiptgenerator** queue from the list.

29. Select **Message Id > Pop Receipt** from the list, and click **Save**.

The screenshot shows the Logic App Designer interface. A step named "Delete message (Preview)" is selected. It has two inputs: "Message ID" and "Pop receipt", both of which are highlighted with red boxes. To the right, a sidebar titled "Add dynamic content from the apps and services used in this flow" lists several message properties. The "Pop receipt" property is also highlighted with a red box.

30. Click Run on the Logic App Desinger, and then run the Contoso sports Web App and check out an Item.

31. Run the admin website app, and select the last Details link in the list.

Details	4/30/2017 8:38:55 PM	Bob	Loblaw	\$1,250.99
Details	4/30/2017 8:51:32 PM	Bob	Loblaw	\$45.50
Details	4/30/2017 9:01:54 PM	Bob	Loblaw	\$12.98
Details	4/30/2017 10:06:32 PM	Bob	Loblaw	\$45.50

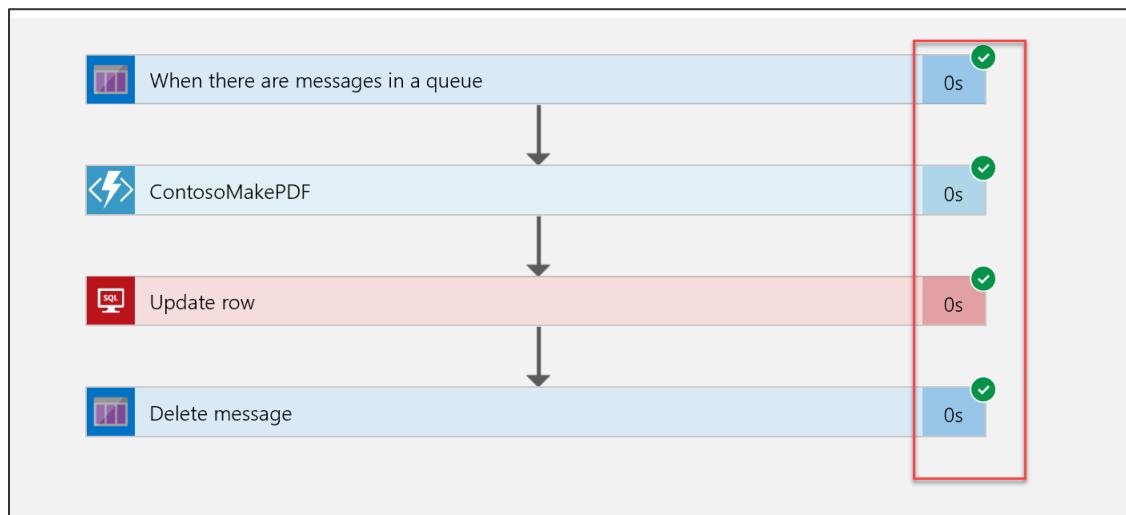
32. You should now see a Download receipt link because the database has been updated.

The screenshot shows the "Order Details" page. It displays contact information for a customer and a table of order history. The last row in the history table, which corresponds to the most recent purchase, has a "Download receipt" button highlighted with a red box.

33. Click on the Download receipt link to see the receipt.

Contoso Sports League - Purchase Receipt				
Product	Quantity	Unit Price	Total	
Baseball Mitt	1	\$45.50	\$45.50	
Page Summary				Summary
Total				\$45.50

34. Return to the Logic app and you should see all green check marks for each step. If not, click the yellow status icon to find out details.



Task 3: Use Twilio to send SMS Order Notifications

Subtask 1: Configure your Twilio trial account

1. If you do not have a Twilio account, sign up for one for free at the following URL:
<https://www.twilio.com/try-twilio>.

Sign up for free

First Name Last Name

Company Name (optional)

Email

Choose a password Password, again

WHICH PRODUCT DO YOU PLAN TO USE FIRST?

SMS

WHAT ARE YOU BUILDING?

Other

CHOOSE YOUR LANGUAGE

C#

Get Started By clicking the button, you agree to our [legal policies](#).

Already have an account? [Login](#)

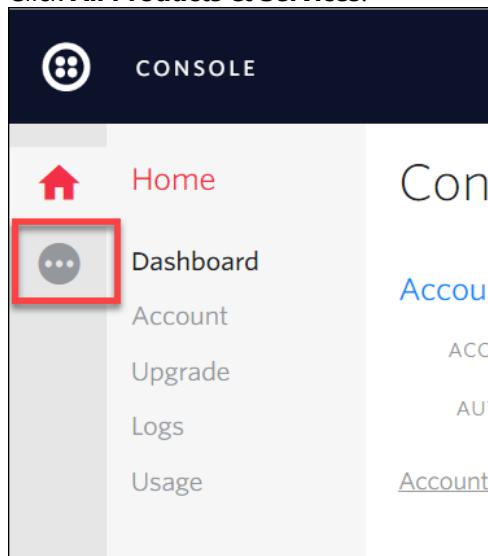
- When you sign up for a free Twilio trial, you will be asked to verify your personal phone number. This is an important security step that is mandatory for trying Twilio.

We need to verify you're a human.

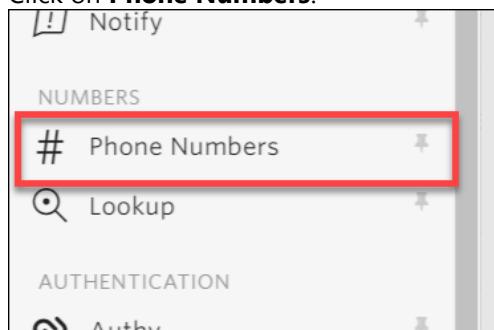
  +1 Phone Number Verify via SMS

We will send a verification code via **SMS** to number above
Or, we [call you instead](#).

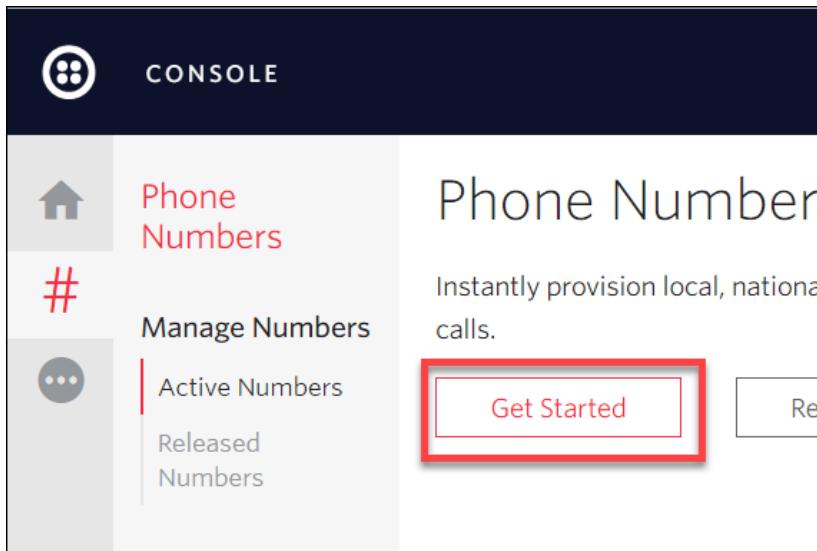
- Click **All Products & Services**.



- Click on **Phone Numbers**.



5. Click **Get Started**.



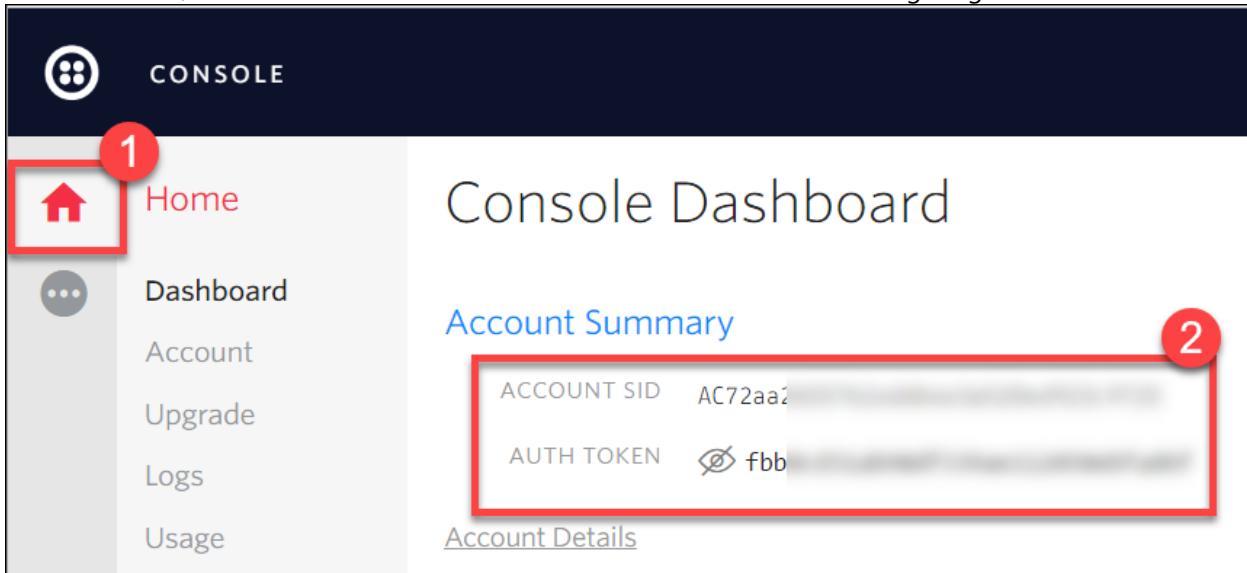
6. Click the **Get your first Twilio phone number** button.

This screenshot shows a 'Get Started with Phone Numbers' page. It features a prominent red button labeled 'Get your first Twilio phone number'. Below the button, there's a note: 'Looking for a short-code? [Apply for a short-code here.](#)'

7. Record the **Phone Number**, click the **Choose this Number** button on the **Your first Twilio Phone Number** prompt, and click **Done**.

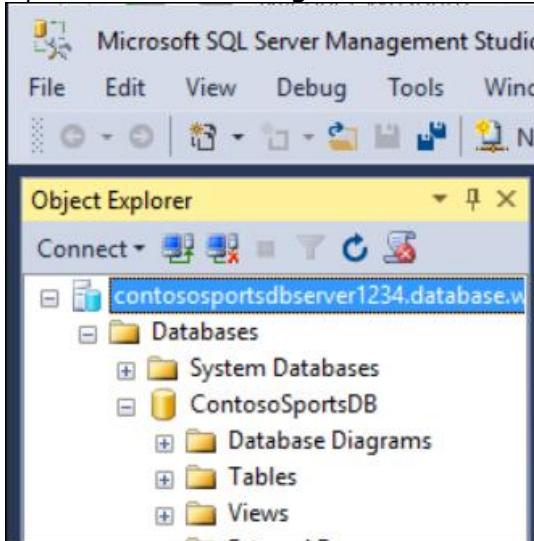
This screenshot shows a modal window titled 'Your first Twilio Phone Number'. It displays a phone number '(262) 675-XXXX' and a link to search for a different number if desired. Below the number, it lists capabilities: Voice, SMS, and MMS. At the bottom are 'Cancel' and 'Choose this Number' buttons.

8. Click on **Home**, record the **Account SID** and **Auth Token** for use when configuring the Twilio Connector.

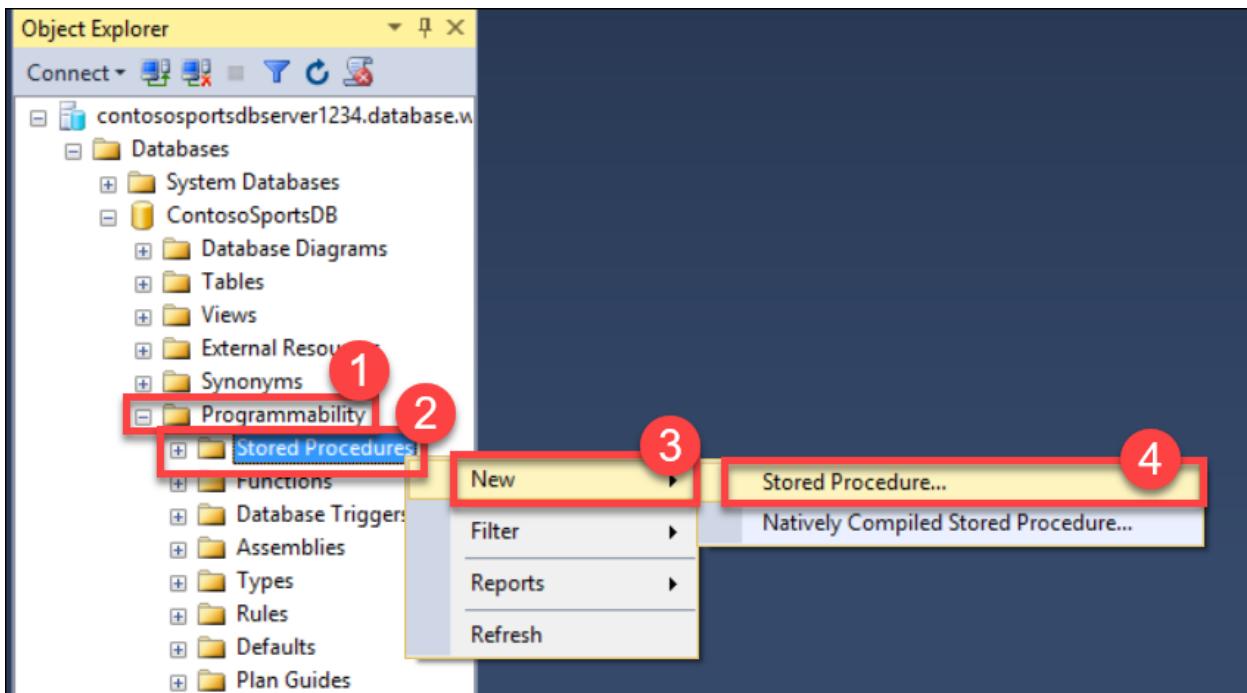


Subtask 2: Create a new logic app

1. Open **SQL Server Management Studio** and connect to the SQL Database for the **ContosoSportsDB** database.



2. Under the **ContosoSportsDB** database, expand **Programmability**, right-click on **Stored Procedures**, click **New**, followed by **Stored Procedure...**



- Replace the Stored Procedure Template code with the following:

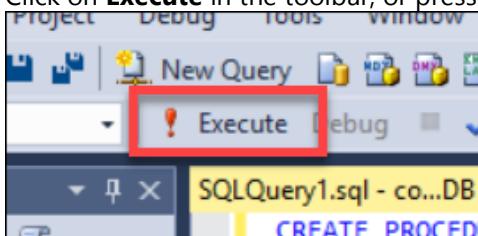
```

CREATE PROCEDURE [dbo].[GetUnprocessedOrders]
AS
declare @returnCode int
SELECT @returnCode = COUNT(*) FROM [dbo].[Orders] WHERE PaymentTransactionId
is not null AND PaymentTransactionId <> '' AND Phone is not null AND Phone
<> '' AND SMSOptIn = '1' AND SMSStatus is null
return @returnCode

GO

```

- Click on **Execute** in the toolbar, or press the F5 key.



- Delete the SQL script for the Stored Procedure from the code editor, and replace it with the following:

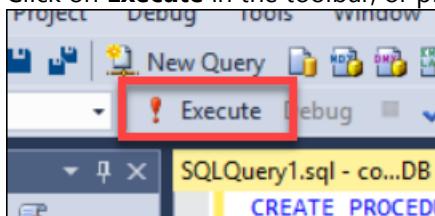
```

CREATE PROCEDURE [dbo].[ProcessOrders]
AS
SELECT * FROM [dbo].[Orders] WHERE PaymentTransactionId is not null AND
PaymentTransactionId <> '' AND Phone is not null AND Phone <> '' AND
SMSOptIn = '1' AND SMSStatus is null;

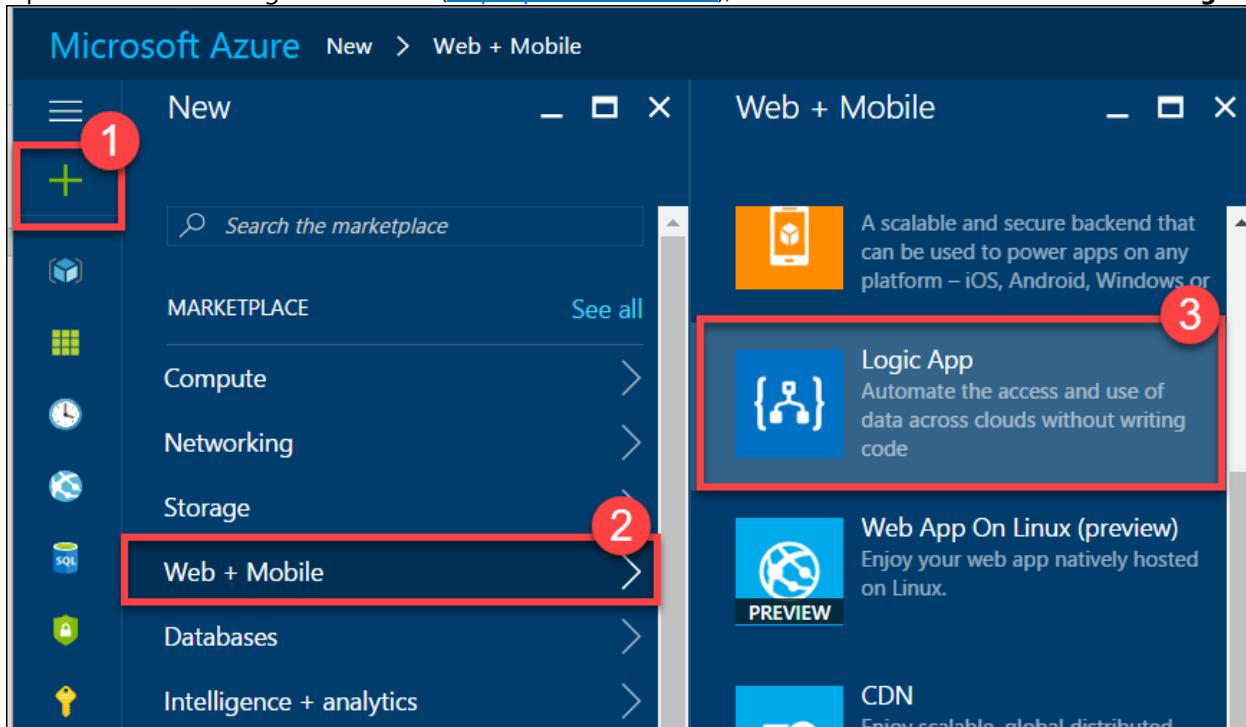
```

```
UPDATE [dbo].[Orders] SET SMSStatus = 'sent' WHERE PaymentTransactionId is not null AND PaymentTransactionId <> '' AND Phone is not null AND Phone <> '' AND SMSOptIn = '1' AND SMSStatus is null;
```

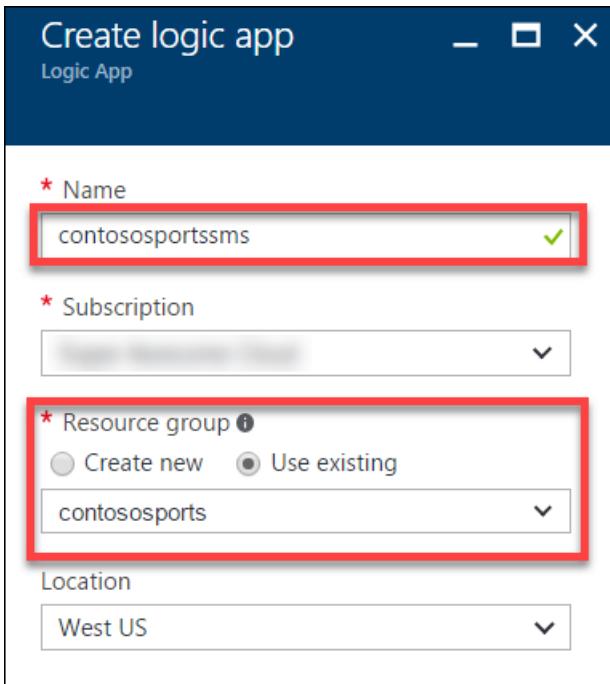
6. Click on **Execute** in the toolbar, or press the F5 key.



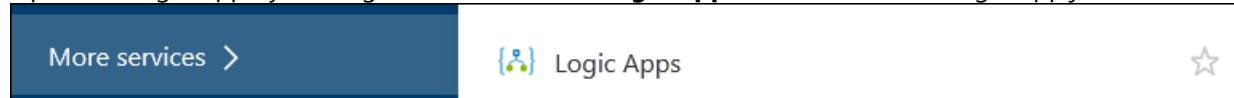
7. Open the Azure Management Portal (<http://portal.azure.com>), and click **+New > Web + Mobile > Logic App**.



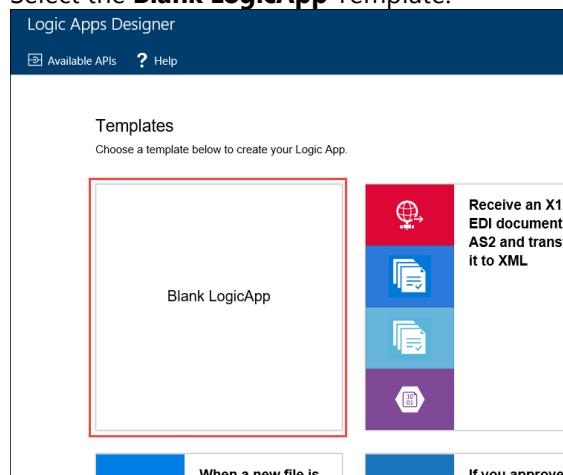
8. On the **Create logic app** blade, assign a value for **Name**, and set the Resource Group to **contososports**.



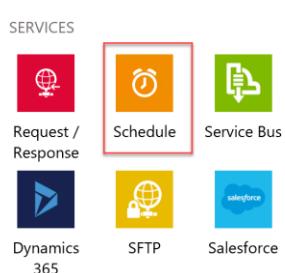
9. Open the Logic App by clicking **More services -> Logic Apps**, and click on the Logic App just created.



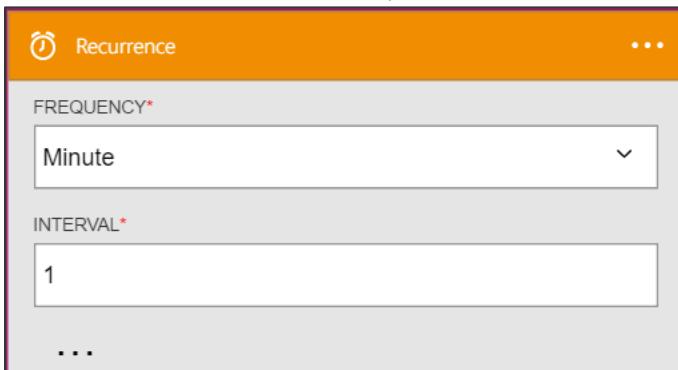
10. Select the **Blank LogicApp** Template.



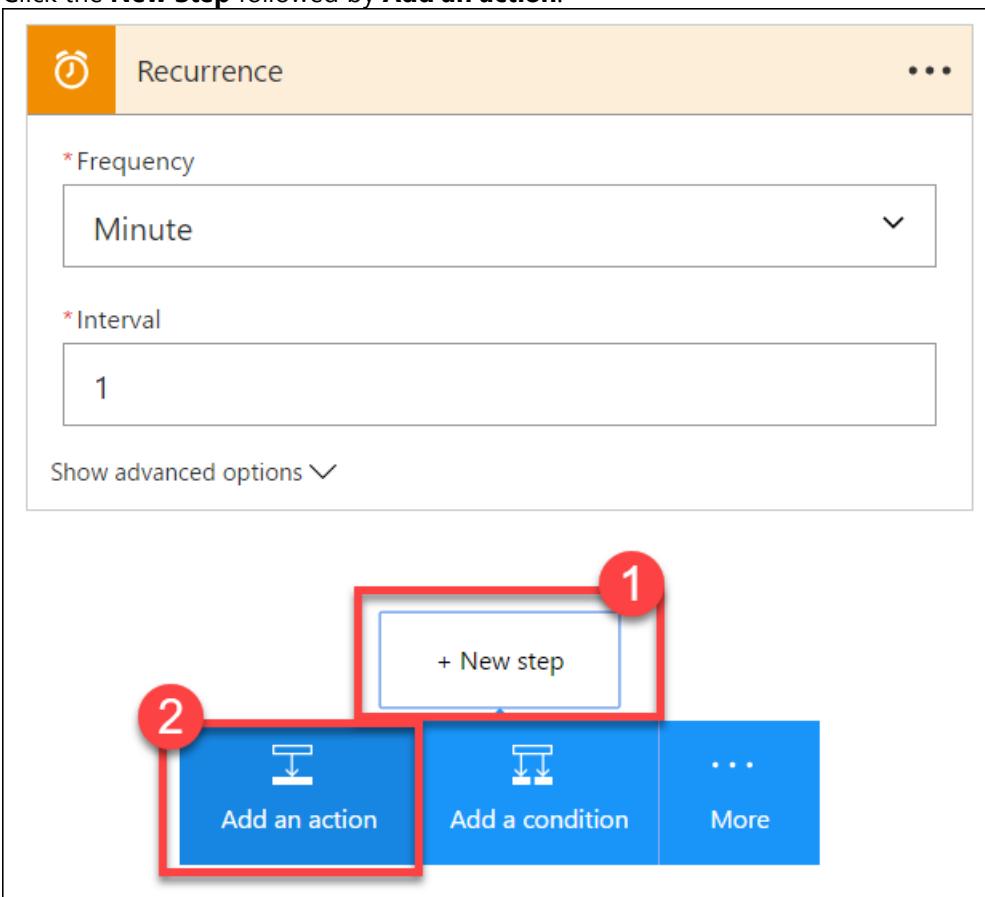
11. On the **Logic Apps Designer**, click **Schedule**.



12. Set the **FREQUENCY** to **MINUTE**, and **INTERVAL** to 1.



13. Click the **New Step** followed by **Add an action**.



14. Type **SQL Server** into the filter box, and click the **SQL Server – Execute stored procedure** action.

The screenshot shows a search interface with a filter bar at the top containing the text "sql server". Below the filter, there's a section titled "SERVICES" with a single result: "SQL Server" represented by a red icon with a white "SQL" logo. In the "ACTIONS" section, there are two items: "Delete row" and "SQL Server - Execute stored procedure". The "SQL Server - Execute stored procedure" item is highlighted with a red border and has a red number "1" icon next to it. There are also "SEE MORE" and "TRIGGERS (0)" buttons.

15. The first time you add a SQL action, you will be prompted for the connection information. Name the connection **ContosoDB**, input the server and database details used earlier, and click **Create**.

The screenshot shows the configuration dialog for the "SQL Server - Execute stored procedure" action. It includes fields for "GATEWAYS" (checkbox for "Connect via on-premise data gateway"), "CONNECTION NAME" (set to "ContosoDB" with a red box and red number 1), "SQL SERVER NAME" (set to "contososportsdbserver1234.database.windows.net" with a red box and red number 2), "SQL DATABASE NAME" (set to "ContosoSportsDB"), "USERNAME" (set to "demouser"), and "PASSWORD" (redacted). A large red box surrounds the entire form area. At the bottom is a prominent purple "Create" button.

16. Select the **[dbo].[GetUnprocessedOrders]** stored procedure from the drop-down on the Procedure Name field.

Execute stored procedure

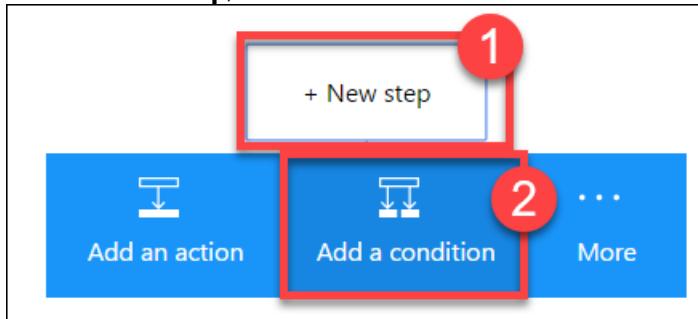
* Procedure name

[dbo].[GetUnprocessedOrders]

* parameters

Connected to ContosoDB. [Change connection.](#)

17. Click on **New Step**, and click the **Add a condition** link.



18. Specify **ReturnCode** for the OBJECT NAME, set the RELATIONSHIP to **is greater than**, and set the VALUE to **0**.

Condition

Object Name

Relationship

Value

ReturnCode

is greater than

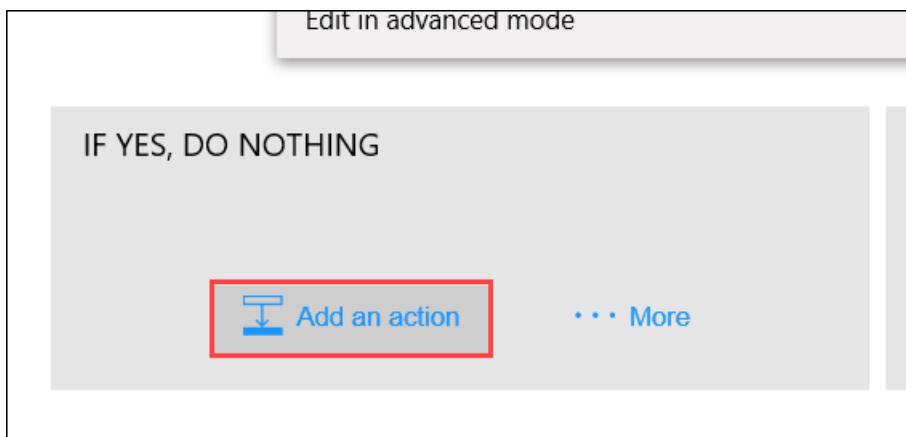
0

Add dynamic content [\[+\]](#)

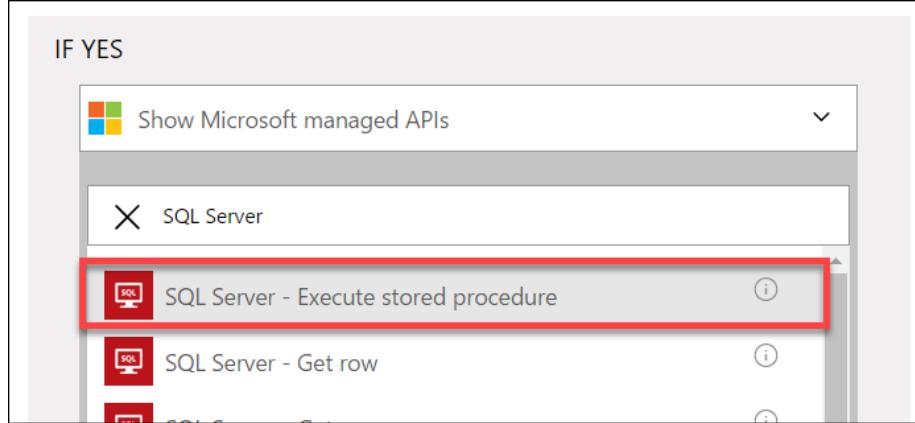
Edit in advanced mode

Collapse condition

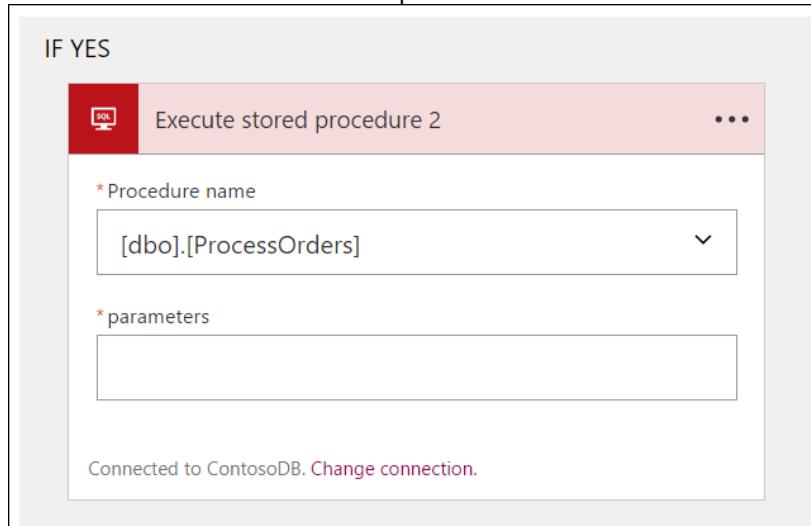
19. Click the **Add an action** link on the **If yes** condition.



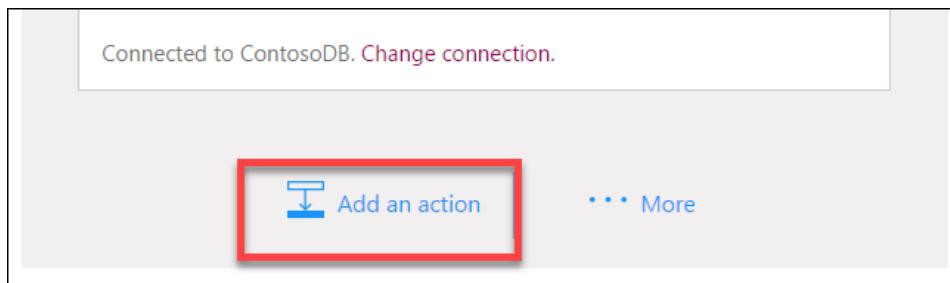
20. Type **SQL Server** into the filter box, and click the **SQL Server – Execute stored procedure** action.



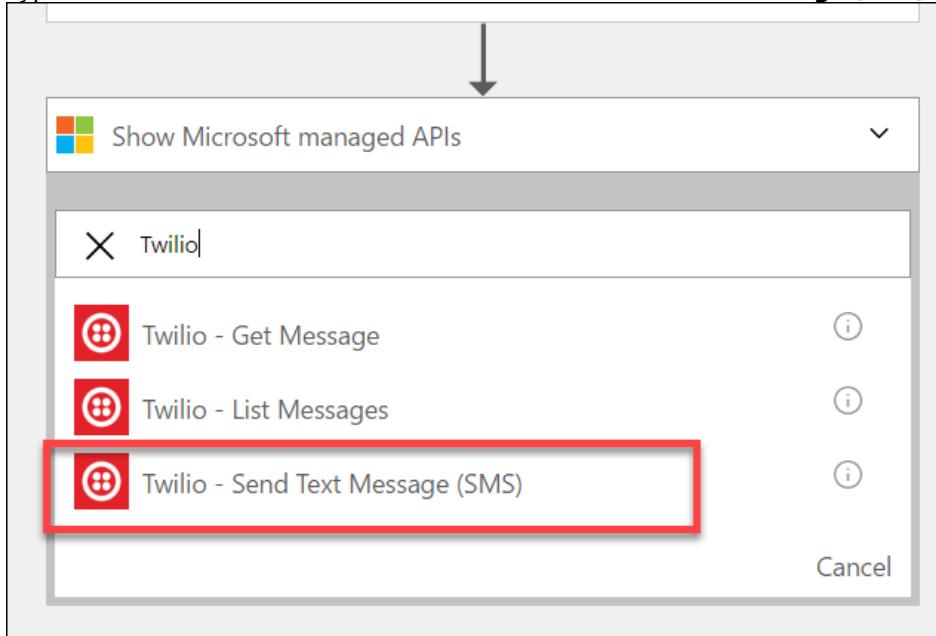
21. Select the **ProcessOrders** stored procedure in the Procedure name dropdown.



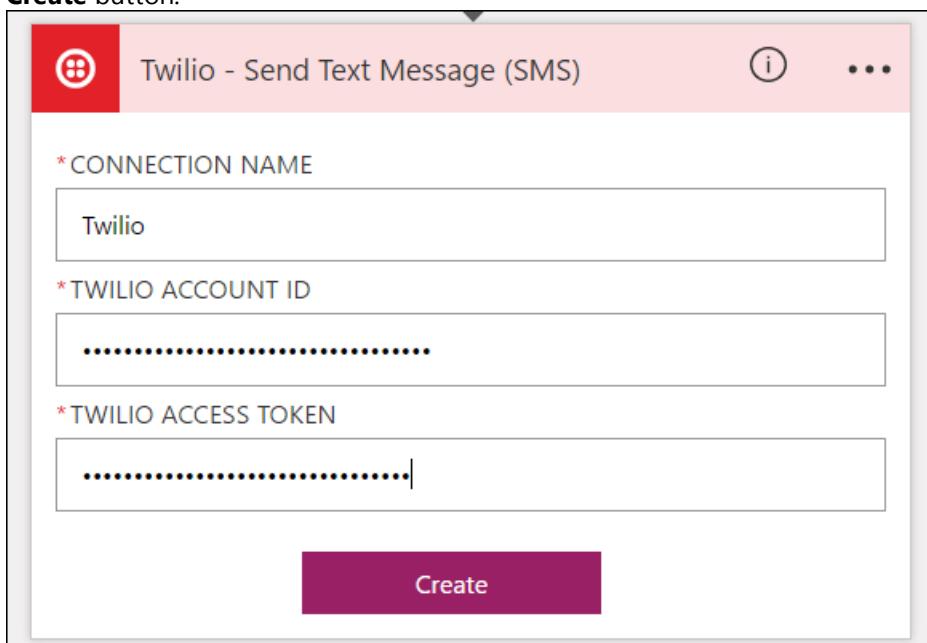
22. Click the **Add an action** link.



23. Type **Twilio** in the filter box, and click the **Twilio – Send Text Message (SMS)** connector.



24. Set the Connection Name to Twilio, specify your Twilio **Account SID** and **Authentication Token**, then click the **Create** button.



25. Using the drop-down, select your Twilio number for the **FROM PHONE NUMBER** field. Specify a place holder phone number in the **TO PHONE NUMBER**, and a **TEXT** message.

The screenshot shows the Logic Apps Designer interface with a 'Send Text Message (SMS)' action selected. The 'From Phone Number' field is highlighted with a red box and contains '(262)'. The 'To Phone Number' field is also highlighted with a red box and contains '214-'. The 'Text' field is highlighted with a red box and contains 'Hello, your order has shipped!'. Below the action, there is a link to 'Add dynamic content' and a button to 'Show advanced options'.

26. On the Logic App toolbar click the **Code View** button.

The screenshot shows the Logic Apps Designer toolbar. The 'Code view' button is highlighted with a red box. Other buttons include 'Save', 'Discard', 'Designer', 'Templates', 'Available APIs', and 'Help'.

27. Find the **Send_Text_Message_(SMS)** action, and modify the body property of the Twilio action:

```
"body": {
    "body": "Hello, your order has shipped!",
    "from": "+1262675_____",
    "to": "214-555-5555"
},
```

Add the following code between Hello and the comma.

```
@{item()['FirstName']}
"body": {
    "body": "Hello @{item()['FirstName']}, your order has shipped!",
    "from": "+1262675_____",
    "to": "214-555-5555"
},
```

28. Modify the **to** property to pull the phone number from the item.

```
@{item()['Phone']}
```

```
"body": {  
    "body": "Hello @{item()['FirstName']}, your order has shipped!",  
    "from": "+1262675",  
    "to": "@{item()['Phone']}"  
},
```

29. Immediately before the **Send_Text_Message_(SMS)**, create a new line, and add the following code:

```
"forEach_email": {  
    "type": "Foreach",  
    "foreach":  
        "@body('Execute_stored_procedure_2')['ResultSets']['Table1']",  
    "actions": {
```

30. Remove the **runAfter** block from the **Send_Text_Message_(SMS)** action.

```
"runAfter": {  
    "Execute_stored_procedure_2": [  
        "Succeeded"  
    ]  
}
```

31. Locate the closing bracket of the **Send_Text_Message_(SMS)** action, create a new line after it, and add the following code:

```
},  
"runAfter": {  
    "Execute_stored_procedure_2": [  
        "Succeeded"  
    ]  
}
```

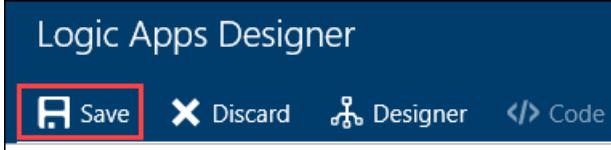
32. After the code for the **Send_Text_Message_(SMS)** has been modified to be contained within the **forEach_email** action, it should look like the following:

```

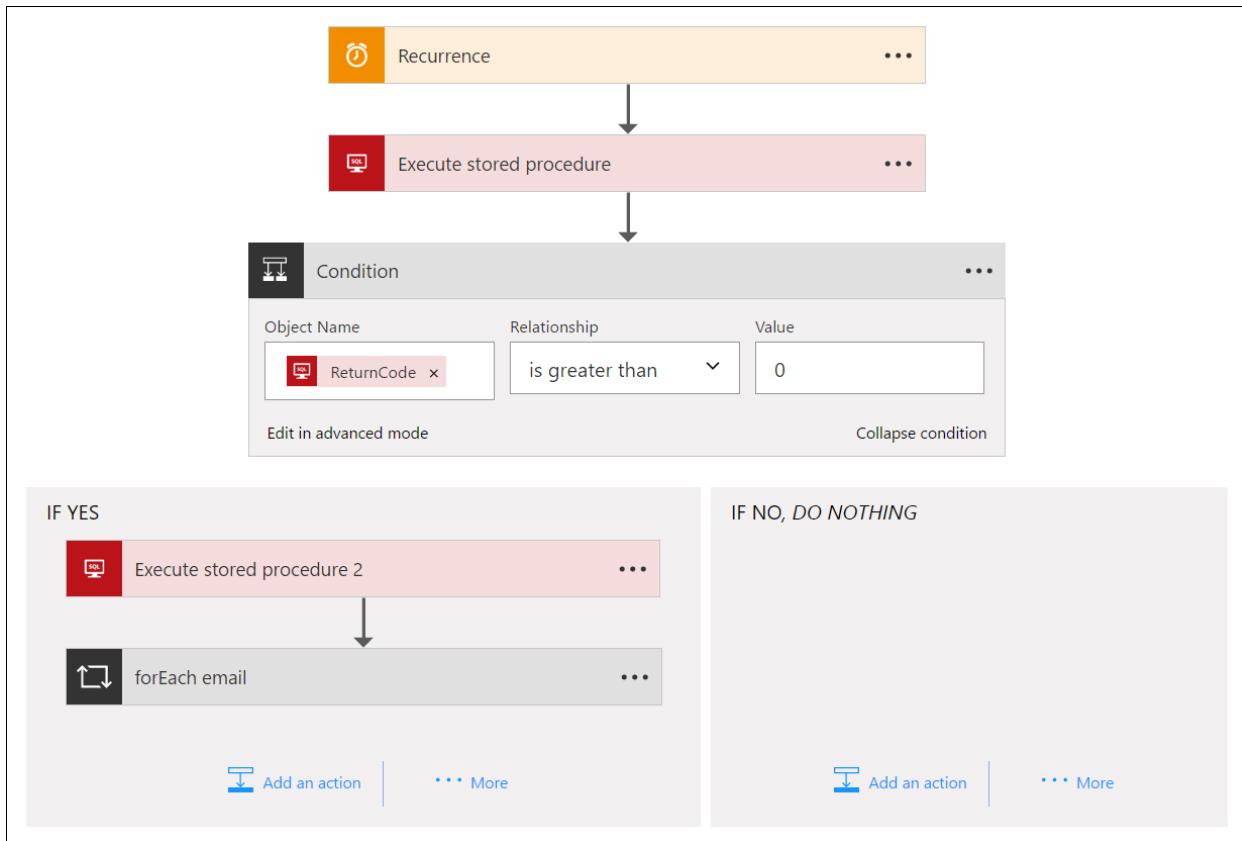
20           "runAfter": {},
21           "type": "ApiConnection"
22     },
23   "forEach_email": {
24     "type": "Foreach",
25     "foreach": "@body('Execute_stored_procedure_2')['ResultSets']['Table1']",
26     "actions": {
27
28       "Send_Text_Message_(SMS)": {
29         "inputs": {
30           "body": {
31             "body": "Hello @{item()['FirstName']}, your order has shipped!",
32             "from": "+1262675_____",
33             "to": "@{item()['Phone']}"
34           },
35           "host": {
36             "api": {
37               "runtimeUrl": "https://logic-apis-westus.azure-apim.net/apim/twilio"
38             },
39             "connection": {
40               "name": "@parameters('$connections')['twilio']['connectionId']"
41             }
42           },
43           "method": "post",
44           "path": "/Messages.json"
45         },
46         "runAfter": {},
47         "type": "ApiConnection"
48       }
49
50
51     },
52     "runAfter": {
53       "Execute_stored_procedure_2": [
54         "Succeeded"
55       ]
56     }
57   }
58 },
59   "expression": "@greater(body('Execute_stored_procedure')?['ReturnCode'], 0)",
60   "runAfter": {}
61

```

33. Click **Save** on the toolbar to enable the logic app.



34. Your workflow should look like below, and you should receive a text for each order you have placed.



After the hands-on lab

Duration: 10 minutes

Task 1: Delete resources

1. Since the HOL is now complete, go ahead and delete all of the Resource Groups that were created for this HOL. You will no longer need those resources and it will be beneficial to clean up your Azure Subscription.

You should follow all steps provided *after* attending the hands-on lab.