



Microsoft Cloud Workshop

Microservices architecture

Infrastructure edition

Hands-on lab step-by-step

March 2018

Information in this document, including URL and other Internet Web site 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

Microservices architecture – infrastructure edition hands-on lab step-by-step	1
Abstract and learning objectives.....	1
Overview	2
Azure resource map	3
Requirements	4
Before the hands-on lab.....	5
Task 1: Provision Service Fabric Cluster.....	5
Task 2: Provision a lab virtual machine (VM)	10
Task 3: Connect to your lab VM.....	14
Task 4: Install Chrome on LabVM	18
Task 5: Install Service Fabric SDK for Visual Studio.....	20
Task 6: Validate Service Fabric ports.....	23
Exercise 1: Environment setup.....	27
Task 1: Download and open the ContosoEventsPoC starter solution	27
Task 2: API Management.....	30
Task 3: Azure Active Directory B2C	31
Task 4: Web app	33
Task 5: Function app	35
Task 6: Storage account.....	37
Task 7: Cosmos DB	39
Exercise 2: Placing ticket orders.....	44
Task 1: Run the solution	44
Task 2: Test the application	47
Task 3: Service Fabric Explorer.....	51
Task 4: Set up the Ticket Order Sync queue.....	52
Task 5: Set up the functions	58
Task 6: Test order data sync	70
Exercise 3: Publish the Service Fabric Application.....	74
Task 1: Publish the application	74
Task 2: Test an order from the cluster.....	76
Exercise 4: API Management.....	79
Task 1: Import API.....	79
Task 2: Retrieve the user subscription key.....	81
Task 3: Configure the Function App with the API Management key	82
Exercise 5: Configure and publish the web application.....	85
Task 1: Configure the Web App settings.....	85
Task 2: Running the web app and creating an order.....	86
Task 3: Publish the web app.....	90

Exercise 6: Upgrading	93
Task 1: How upgrade works	93
Task 2: Update an actor state	94
Task 3: Perform a smooth upgrade.....	95
Task 4: Submit a new order	99
Exercise 7: Rollback.....	102
Task 1: Add health checks.....	102
Task 2: Perform a troubled upgrade.....	102
Exercise 8: Load testing	106
Task 1: Simulate a 50-order request.....	106
BONUS Exercise 9: Load testing w/ partitions.....	107
Task 1: Setting up for load testing	107
Task 2: Using the current partition count, simulate 100 orders.....	109
Task 3: Using a partition count of 10, simulate 100 orders.....	113
Task 4: Using a partition count of 15, simulate 100 orders.....	113
Task 5: Using a partition count of 20, simulate 100 orders.....	114
Task 6: Determine the performance variations among the different load tests.....	115
BONUS Exercise 10: Secure the web application.....	118
Task 1: Configure the Azure Active Directory B2C	118
Task 2: Configure the web app Settings.....	137
Task 3: Add security features to the web application	138
Task 4: Running the web app and signing in.....	139
After the hands-on lab	141
Task 1: Delete the resource group	141

Microservices architecture – infrastructure edition hands-on lab step-by-step

Abstract and learning objectives

This whiteboard design session is designed to help attendees gain a better understanding of implementing architectures leveraging aspects from microservices and serverless architectures, by helping an online concert ticket vendor survive the first 5 minutes of crushing load. They will handle the client's scaling needs through microservices built on top of Service Fabric, and apply smooth updates or roll back failing updates. Finally, students will design an implementation of load testing to optimize the architecture for handling spikes in traffic.

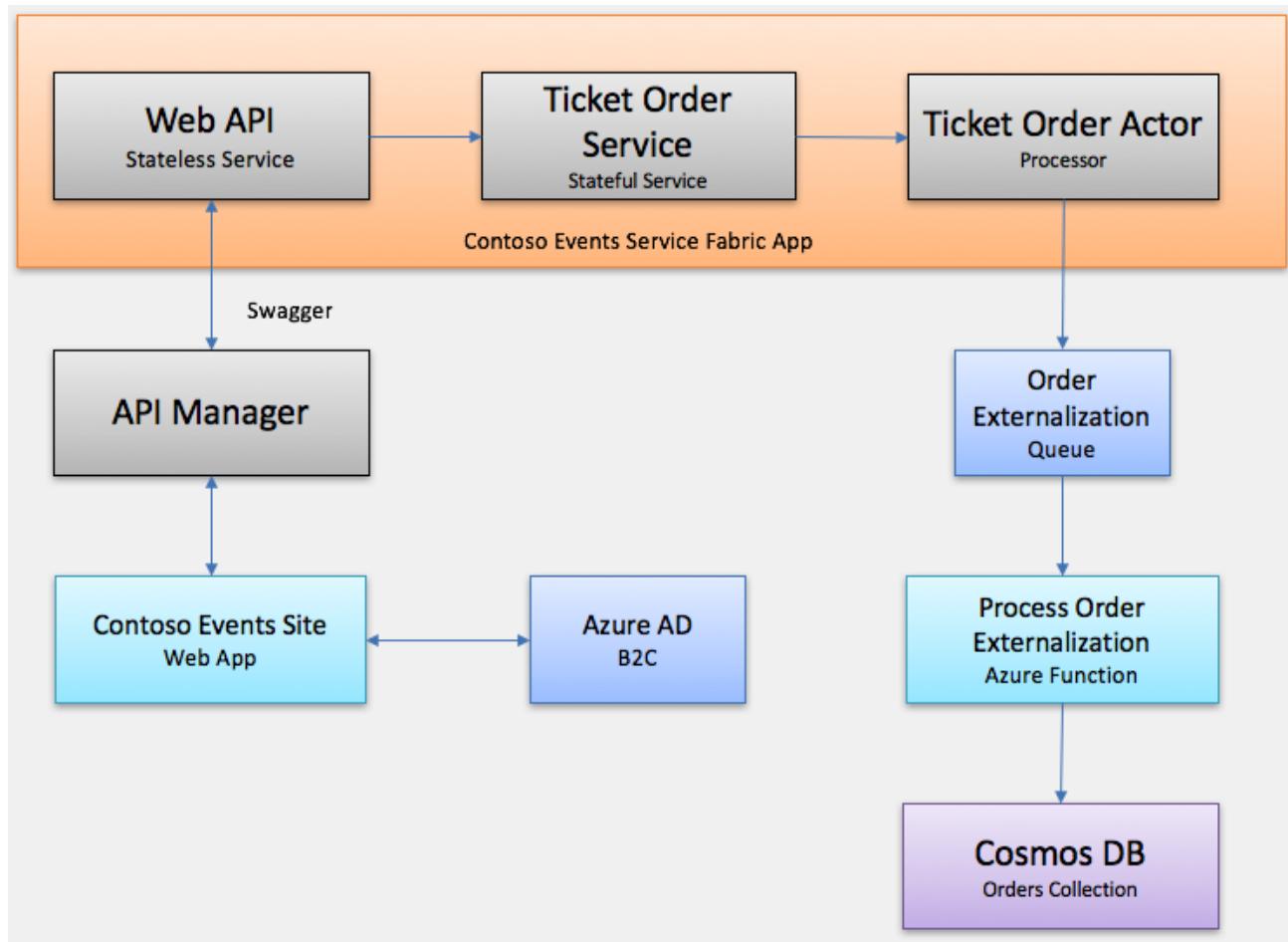
Attendees will learn how to:

- Implement scale and resiliency with Service Fabric
- Enable serverless solutions with Azure Functions
- Control API access with API Management
- Provide query flexibility with Cosmos DB

Overview

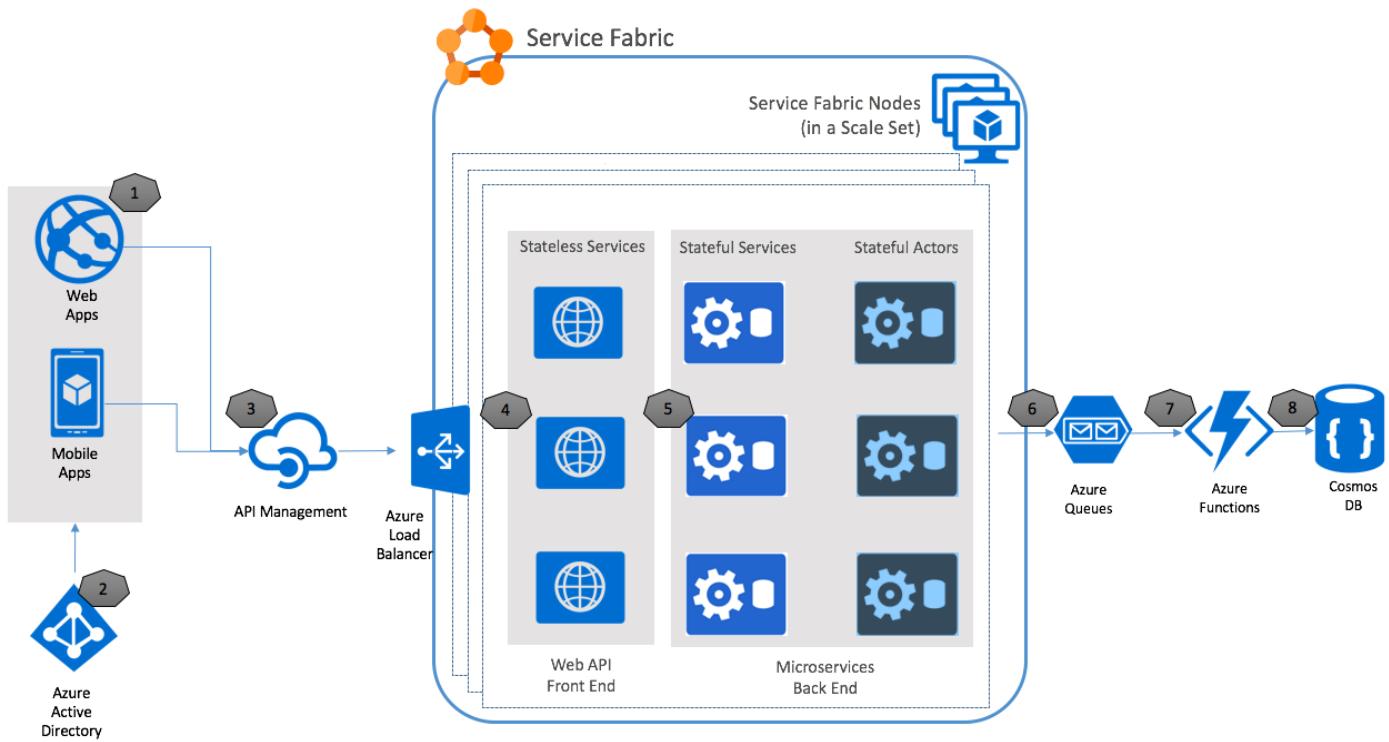
Contoso Events is an online service for concerts, sporting and other event ticket sales. They are redesigning their solution for scale with a microservices strategy and want to implement a POC for the path that receives the most traffic: ticket ordering.

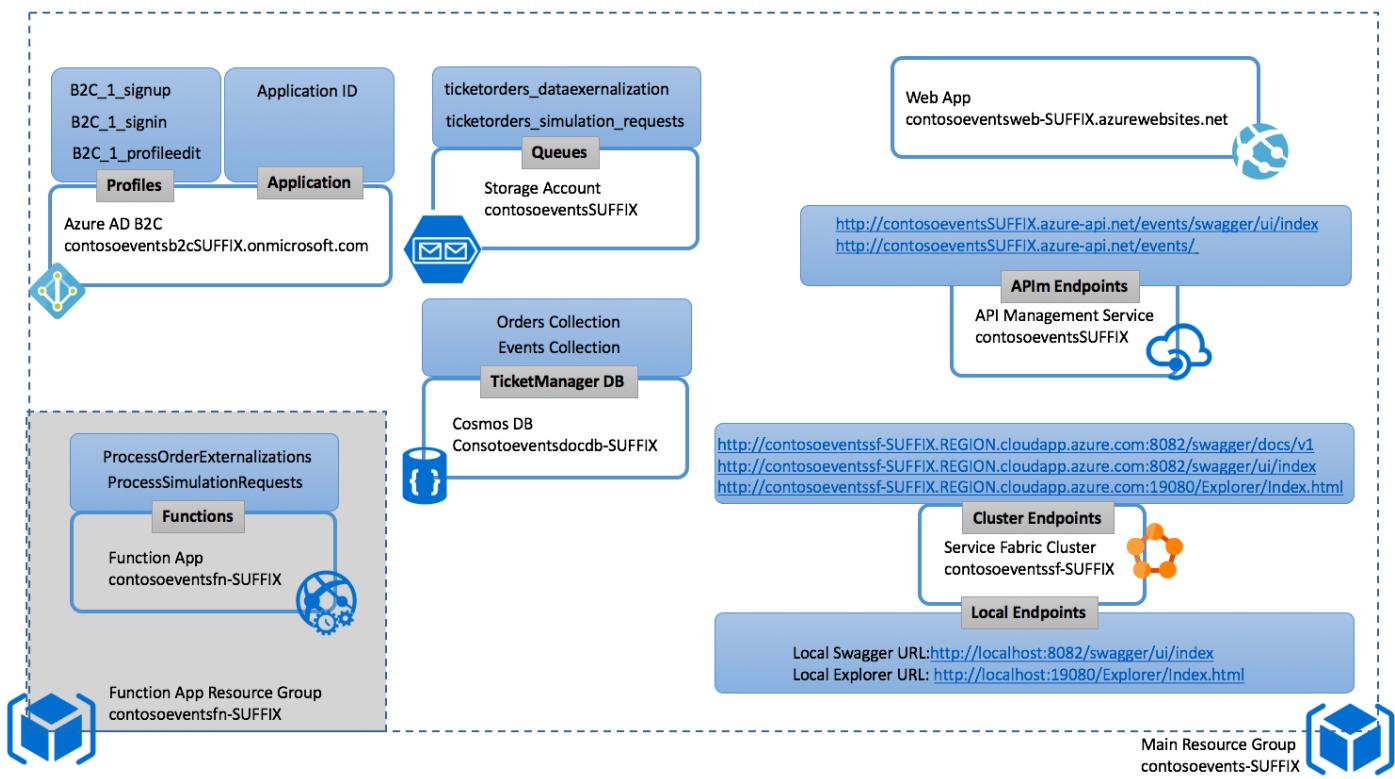
In this hands-on lab, you will construct an end-to-end POC for ticket ordering. You will leverage Service Fabric, API Management, Function Apps, Web Apps, and Cosmos DB.



Solution architecture

The following figures are intended to help you keep track of all the technologies and endpoints you are working with in this hands-on lab. The first figure is the overall architecture, indicating the Azure resources to be employed. The second figure is a more detailed picture of the key items you will want to remember about those resources as you move through the exercises.





Requirements

1. Microsoft Azure subscription must be pay-as-you-go or MSDN.
 - a. Trial subscriptions will not work.
2. A virtual machine configured with (see [Before the hands-on lab](#)):
 - a. Visual Studio 2017 Community edition, or later
 - b. Azure Development workload enabled in Visual Studio 2017 (enabled by default on the VM)
 - c. Service Fabric SDK 2.7 or later for Visual Studio 2017
 - d. Google Chrome browser (Swagger commands do not work in IE)
 - e. PowerShell 3.0 or higher (v5.1 already installed on VM)

Before the hands-on lab

Duration: 45 minutes

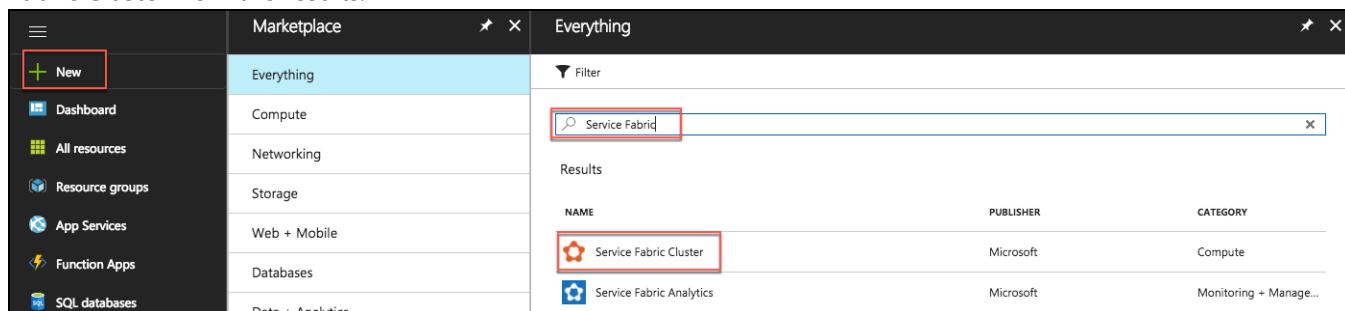
Synopsis: In this exercise, you will set up your environment for use in the rest of the hands-on lab. You should follow all the steps provided in the Before the hands-on lab section to prepare your environment before attending the hands-on lab.

IMPORTANT: Most Azure resources require unique names. Throughout these steps, you will see the word "SUFFIX" as part of resource names. You should replace this with your Microsoft alias, initials, or other value to ensure the resource is uniquely named.

Task 1: Provision Service Fabric Cluster

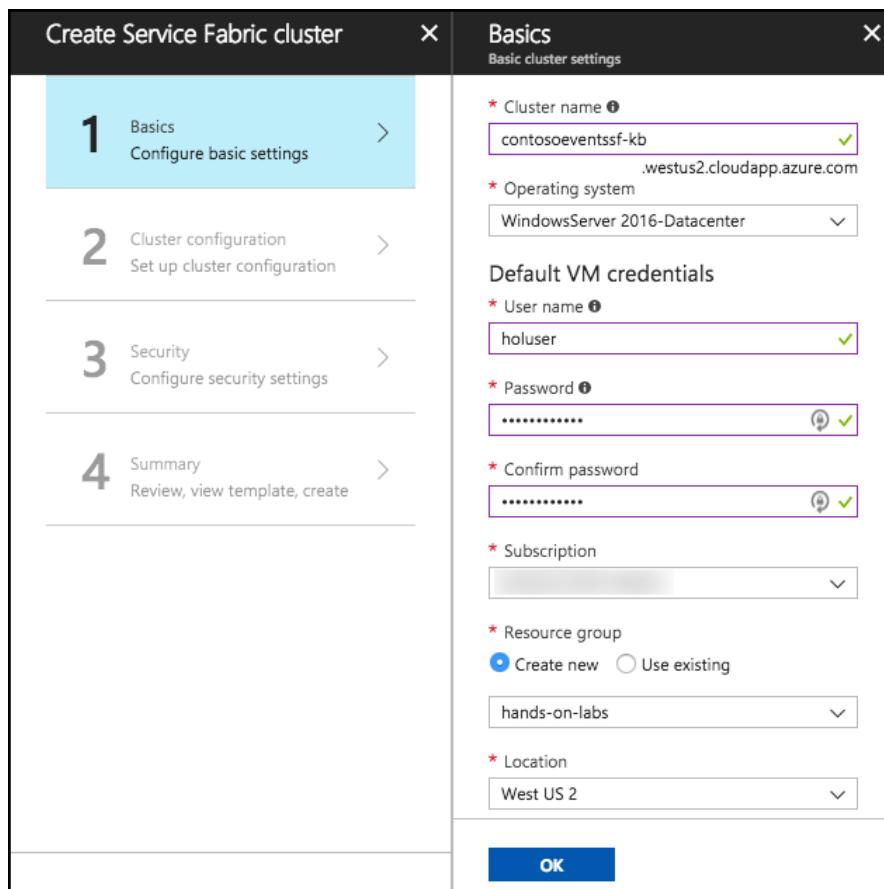
In this task, you will provision the Service Fabric Cluster in Azure.

1. In the Azure portal, select +New, then type "Service Fabric" into the Search the Marketplace box. Select Service Fabric Cluster from the results.



2. On the Service Fabric Cluster blade, select Create.
3. On the Basics blade of the Create Service Fabric cluster screen, enter the following:
 - a. Cluster name: Enter contosoeventssf-SUFFIX, replacing SUFFIX with your alias, initials, or another value to make the name unique (indicated by a green check in the text box).
 - b. Operating system: Leave set to WindowsServer 2016-Datacenter
 - c. User name: Enter holuser
 - d. Password: Enter Password.1!!
 - e. Subscription: Select the subscription you are using for this lab
 - f. Resource Group: Select Create new, and enter hands-on-labs for the resource group name. You can add - SUFFIX, if needed to make resource group name unique. This is the resource group you will use for all resources you create for this hands-on lab.
 - g. Location: Select the region to use.

h. Select OK.



4. On the Cluster configuration blade, set the following:

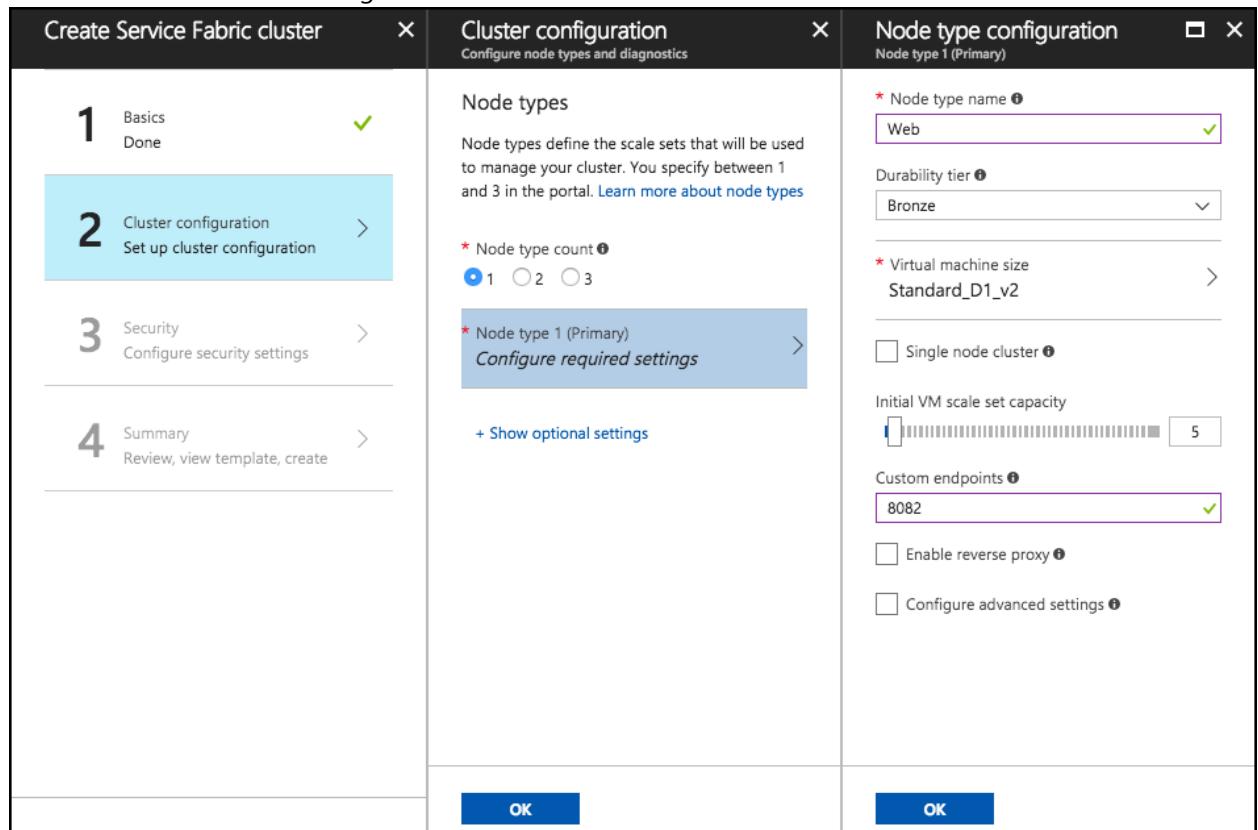
- Node type count: Select 1
- Node type 1 (Primary): Select to configure required settings. On the Node type configuration blade enter:
 - Node type name: Enter Web
 - Durability tier: Leave Bronze selected

- iii. Virtual machine size: Select a VM size of D1_V2 Standard and select Select on the Choose a size blade.

D1_V2 Standard ★	D2_V2 Standard ★	D3_V2 Standard ★
1 vCPU	2 vCPUs	4 vCPUs
3.5 GB	7 GB	14 GB
4 Data disks	8 Data disks	16 Data disks
2x500 Max IOPS	4x500 Max IOPS	8x500 Max IOPS
50 GB Local SSD	100 GB Local SSD	200 GB Local SSD
Load balancing	Load balancing	Load balancing
91.51 USD/MONTH (ESTIMATED)	183.02 USD/MONTH (ESTIMATED)	365.30 USD/MONTH (ESTIMATED)

- iv. Single node cluster: Leave unchecked
v. Initial VM scale set capacity: Leave set to 5
vi. Custom endpoints: Enter 8082. This will allow the Web API to be accessible through the cluster.
vii. Enable reverse proxy: Leave unchecked
viii. Configure advanced settings: Leave unchecked
ix. Select OK on the Node type configuration blade.

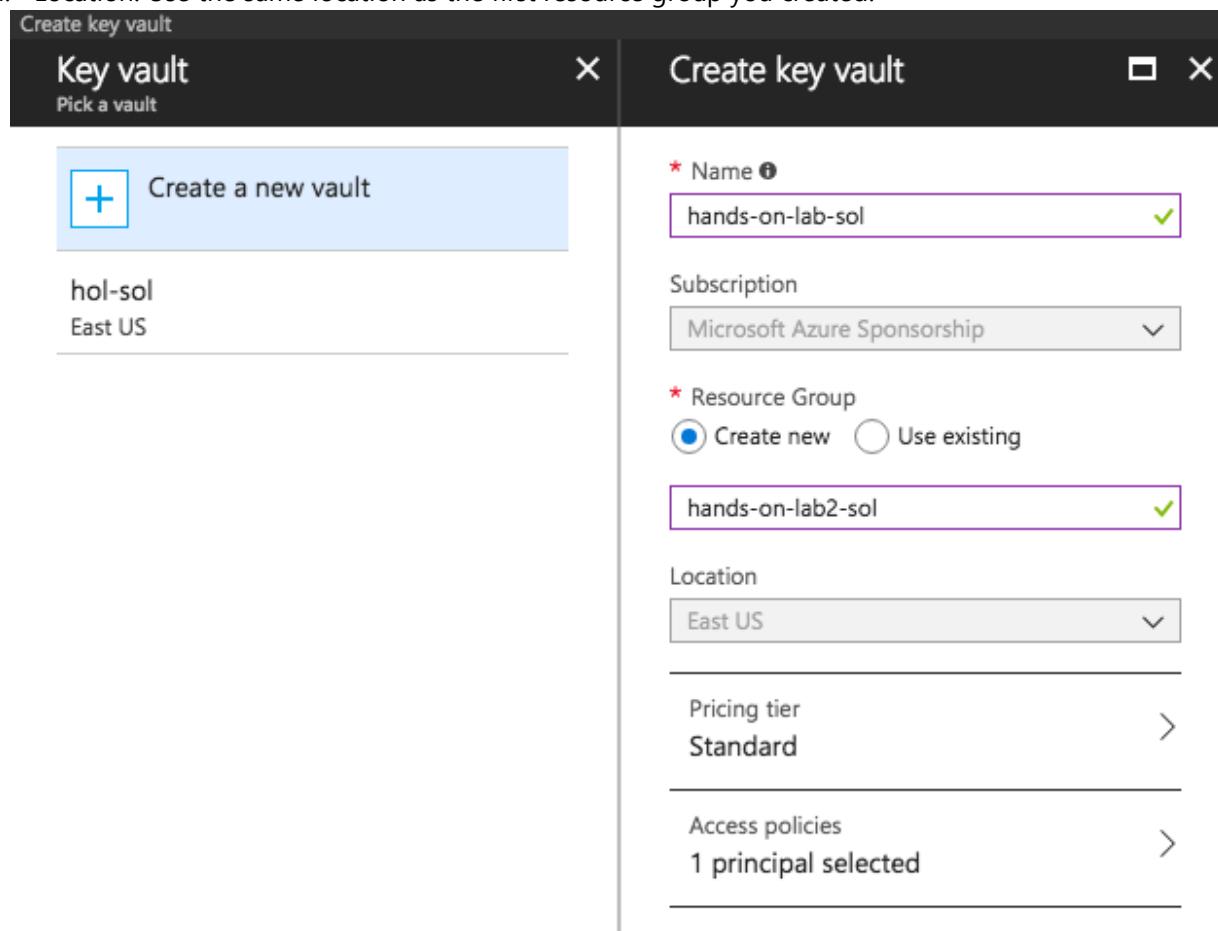
- c. Select OK on the Cluster configuration blade.



5. On the Security blade, you can provide security settings for your cluster. This configuration is completed up front, cannot be changed later. Set the following:
- Configuration Type: select "Basic"
 - Key vault: Select to configure required settings. On the Key vault configuration blade click "Create a new vault".

- c. On the “Create key vault” configuration blade enter:

- i. Name: hands-on-lab-SUFFIX
- ii. Resource Group: hands-on-lab2-SUFFIX
- iii. Location: Use the same location as the first resource group you created.



- d. Click “Create” on the Create key vault configuration blade. Wait for the key vault deployment to complete.
- e. When the key vault deployment completes you will return to the Security configuration blade. You will see a warning that the key vault is not enabled for deployment. Follow these steps to resolve the warning:
- i. Click “Edit access policies for hands-on-lab-SUFFIX”
 - ii. In the Access policies configuration blade, click the link “Click to show advanced access policies”
 - iii. Check the “Enable access to Azure Virtual Machines for deployment” checkbox.

- iv. Click "Save". When the key vault update completes, close the Access policies blade.

The screenshot shows two overlapping windows. The left window is titled 'Security' and shows a 'Configuration Type' section with 'Basic' selected. It also displays a note about certificate creation for basic mode and a key vault named 'hands-on-lab-sol'. A warning message states that the selected key vault is not enabled for deployment and provides instructions to enable it via access policies. The right window is titled 'Access policies' and contains a list of checkboxes for enabling access to Azure Virtual Machines, Resource Manager, and Disk Encryption, with the first checkbox checked. There are buttons for 'Save', 'Discard', and 'Refresh' at the top.

- f. Enter "hands-on-lab-sol" as the certificate name. Then click Ok on the Security configuration blade.
6. On the Summary blade, review the summary, and select Create to begin provisioning the new cluster.

The screenshot shows the 'Create Service Fabric cluster' blade with the 'Summary' tab selected. The left sidebar lists four steps: 1. Basics (Done), 2. Cluster configuration (Done), 3. Security (Done), and 4. Summary (Review, view template, create). The main area displays validation status ('Validation passed'), basic settings (Subscription: hands-on-labs, Resource group: hands-on-labs, Location: West US 2), and cluster settings (Cluster name: contosoeventssf-kb, User name: holuser, Node type count: 1, Security mode: Unsecure, Create application log storage: On). Below these are node type details (Node type 1 (Primary): Web (5xStandard_D1_v2), Virtual machine size: Standard_D1_v2, Custom endpoints: 8082). At the bottom are 'Create' and 'Download template and parameters' buttons.

7. It can take up to 30 minutes or more to provision your Service Fabric Cluster. You can move on to the next task while you wait.
8. **Note:** If you experience errors related to lack of available cores, you may have to delete some other compute resources, or request additional cores be added to your subscription, and then try this again.

Task 2: Provision a lab virtual machine (VM)

In this task, you will provision a virtual machine (VM) in Azure. The VM image used will have Visual Studio Community 2017 installed.

1. Launch a web browser and navigate to the [Azure portal](#).
2. Select +New, then type "Visual Studio" into the search bar. Select Visual Studio Community 2017 (latest release) on Windows Server 2016 (x64) from the results.

The screenshot shows the Azure Marketplace interface. On the left, there's a sidebar with a '+ New' button highlighted with a red box. The main area is titled 'Marketplace' with a 'Everything' filter. A search bar contains the text 'visual studio'. Below it, the results table has columns for NAME, PUBLISHER, and CATEGORY. Two items are listed:

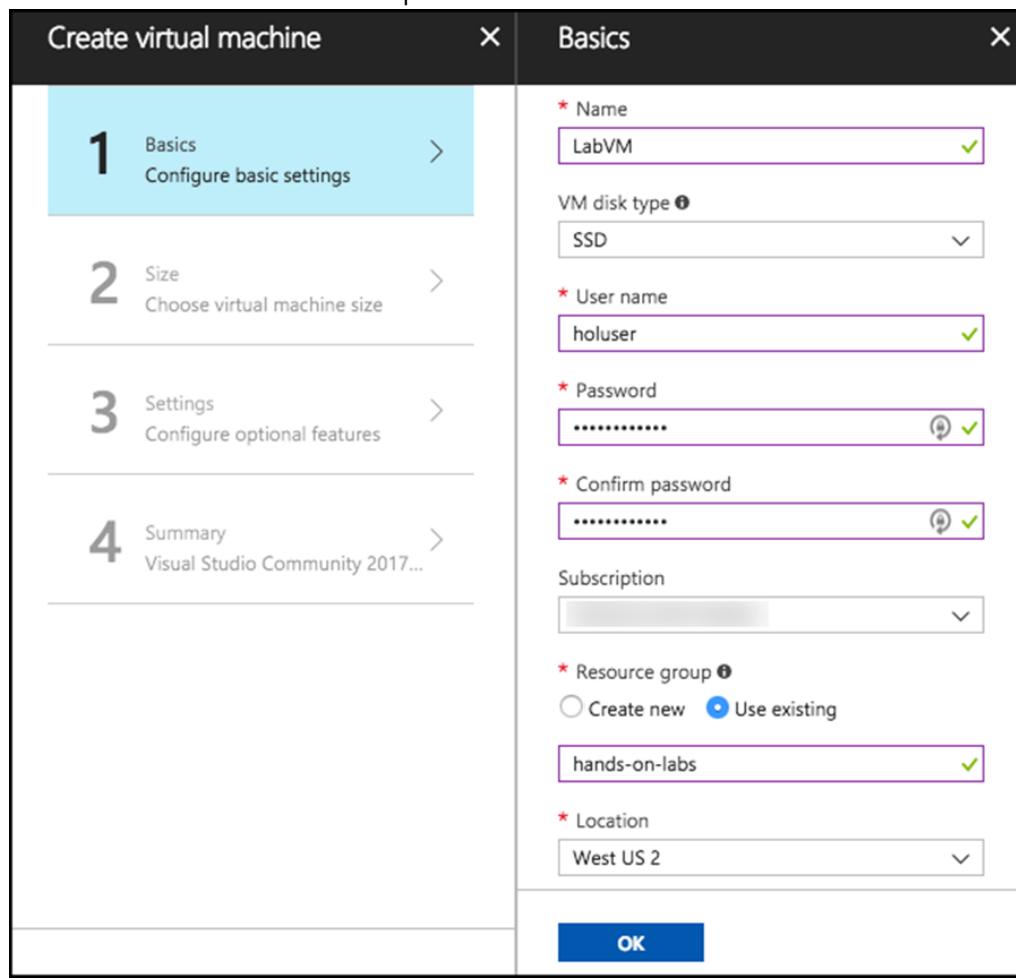
NAME	PUBLISHER	CATEGORY
Visual Studio Community 2017 (latest release) on Windows Server 2016 (x64)	Microsoft	Compute
Visual Studio Community 2017 on Windows Server 2016 (x64)	Microsoft	Compute

3. On the blade that comes up, ensure the deployment model is set to Resource Manager and select Create.

A screenshot of a dropdown menu titled 'Select a deployment model'. It contains a single option: 'Resource Manager', which is highlighted with a red box. Below the dropdown is a blue 'Create' button.

4. Set the following configuration on the Basics tab.
 - a. Name: Enter LabVM
 - b. VM disk type: Select SSD
 - c. User name: Enter holuser
 - d. Password: Enter Password.1!!
 - e. Subscription: Select the subscription group you are using for this lab
 - f. Resource group: Select Use existing, and select the hands-on-labs resource group created previously.
 - g. Location: Select the region you are using for resources in this lab.

h. Select OK to move to the next step.



5. On the Choose a size blade, ensure the Supported disk type is set to SSD, and select View all. This machine won't be doing much heavy lifting, so selecting D2S_V3 Standard is a good baseline option.

Supported disk type	Minimum vCPUs	Minimum memory (GiB)
SSD	1	0
★ Recommended View all		
D2S_V3 Standard	D4S_V3 Standard	D8S_V3 Standard
2 vCPUs	4 vCPUs	8 vCPUs
8 GB	16 GB	32 GB
4 Data disks	8 Data disks	16 Data disks
4000 Max IOPS	8000 Max IOPS	16000 Max IOPS
16 GB Local SSD	32 GB Local SSD	64 GB Local SSD
Load balancing	Load balancing	Load balancing
142.85 USD/MONTH (ESTIMATED)	285.70 USD/MONTH (ESTIMATED)	571.39 USD/MONTH (ESTIMATED)
Select		

6. Select Select to move on to the Settings blade.
7. Accept all the default values on the Settings blade and select OK.

8. Select Create on the Create blade to provision the virtual machine.

The screenshot shows the 'Create' blade for provisioning a virtual machine. At the top, a blue bar indicates 'Validation passed'. Below it, the 'Offer details' section shows a 'Standard D2s v3' VM by Microsoft, priced at 0.1920 USD/hr, with a link to 'Pricing for other VM sizes'. There's also a link to 'Terms of use | privacy policy'. A note about Azure resources states that users can use monetary commitment funds or subscription credits for purchases. The 'Summary' section includes 'Basics' settings: Subscription (grayed out), Resource group '(new) hands-on-labs', and Location 'West US 2'. The 'Terms of use' section contains a legal agreement text. At the bottom, there are 'Create' and 'Download template and parameters' buttons.

9. It may take 10+ minutes for the virtual machine to complete provisioning.

Task 3: Connect to your lab VM

In this step, you will open an RDP connection to your Lab VM and disable Internet Explorer Enhanced Security Configuration.

1. Connect to the Lab VM. (If you are already connected to your Lab VM, skip to Step 9.)

- From the left side menu in the Azure portal, select Resource groups, then enter your resource group name into the filter box, and select it from the list.

The screenshot shows the Azure Resource Groups blade. On the left, there's a sidebar with options like New, Dashboard, All resources, Resource groups (which is selected and highlighted with a red box), and App Services. The main area is titled 'Resource groups' and shows a search bar with 'hands-on' (also highlighted with a red box). Below the search bar, it says 'Subscriptions: [redacted] - Do' and '1 items'. A table lists one item: 'NAME' (hands-on-labs) with a small icon next to it.

- Next, select your lab virtual machine, LabVM, from the list.

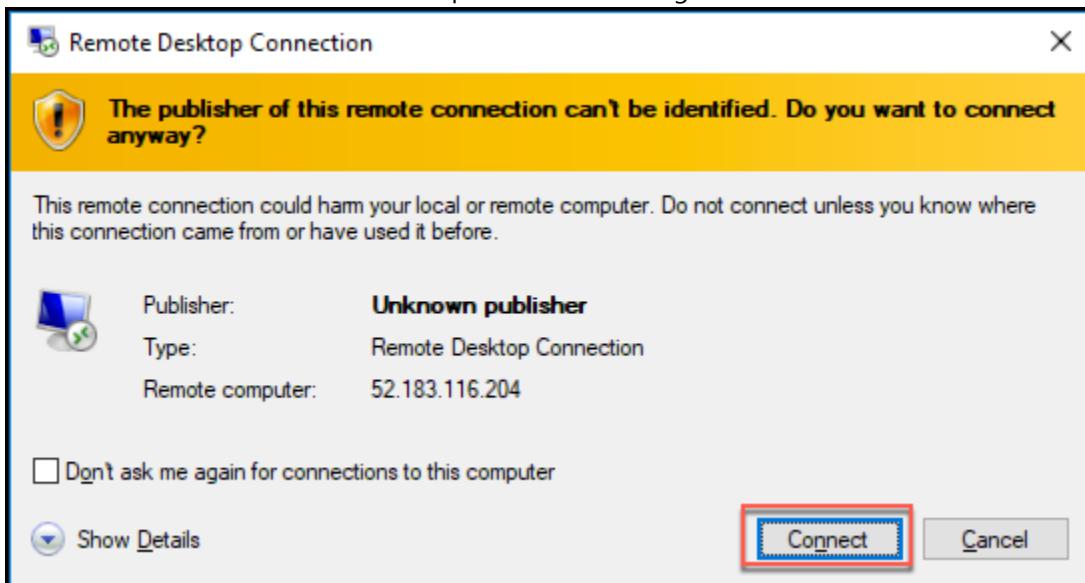
	NAME ↑↓	TYPE ↑↓	LOCATION
<input type="checkbox"/>	hands-onlabsdiag459	Storage account	West US 2
<input type="checkbox"/>	hands-on-labs-vnet	Virtual network	West US 2
<input type="checkbox"/>	LabVM	Virtual machine	West US 2
<input type="checkbox"/>	LabVM_OsDisk_1_c95974cae1fd410b85653d6505c22d9a	Disk	West US 2
<input type="checkbox"/>	labvm697	Network interface	West US 2

- On your Lab VM blade, select Connect from the top menu.

The screenshot shows the Lab VM blade. At the top, there's a menu bar with several icons: Connect (highlighted with a red box), Start, Restart, Stop, Capture, Move, Delete, and Refresh. Below the menu, there's information about the resource group ('hands-on-labs'), status ('Running'), location ('West US 2'), computer name ('LabVM'), operating system ('Windows'), and size ('Standard D2s v3 (2 vcpus, 8 GB memory)').

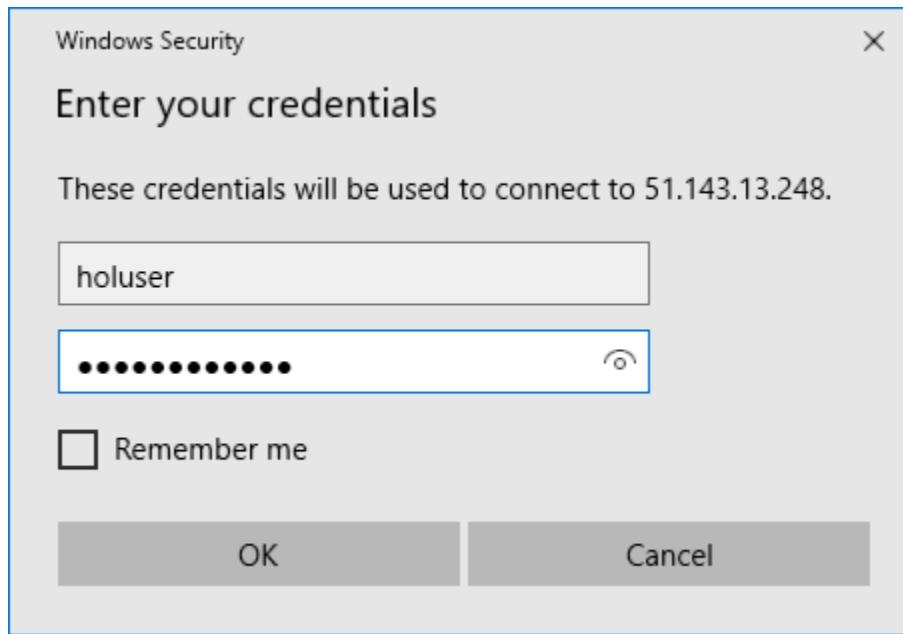
- Download and open the RDP file.

6. Select Connect on the Remote Desktop Connection dialog.

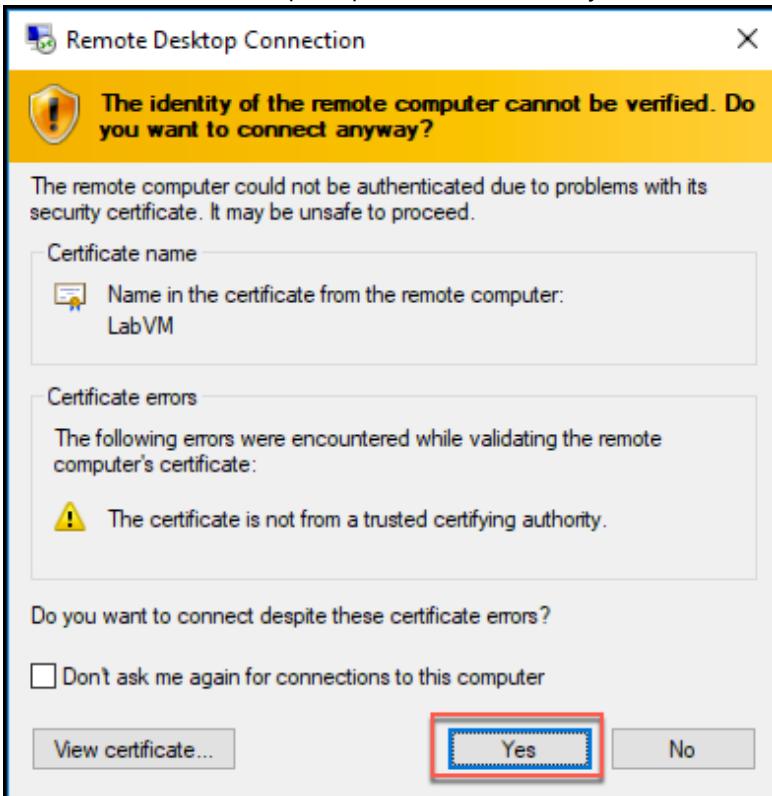


7. Enter the following credentials (or the non-default credentials if you changed them):

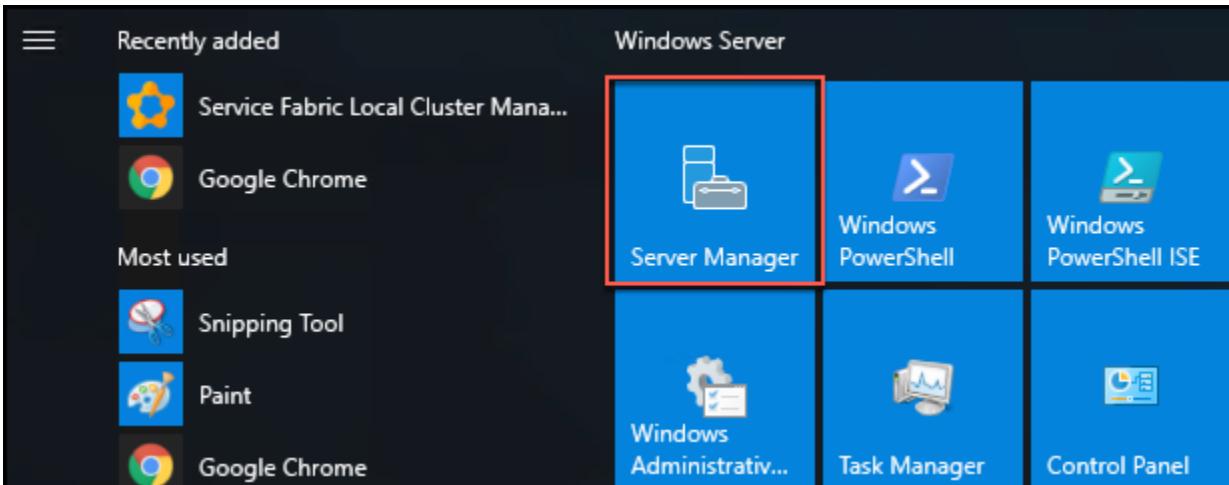
- a. User name: holuser
- b. Password: Password.1!!



8. Select Yes to connect, if prompted that the identity of the remote computer cannot be verified.



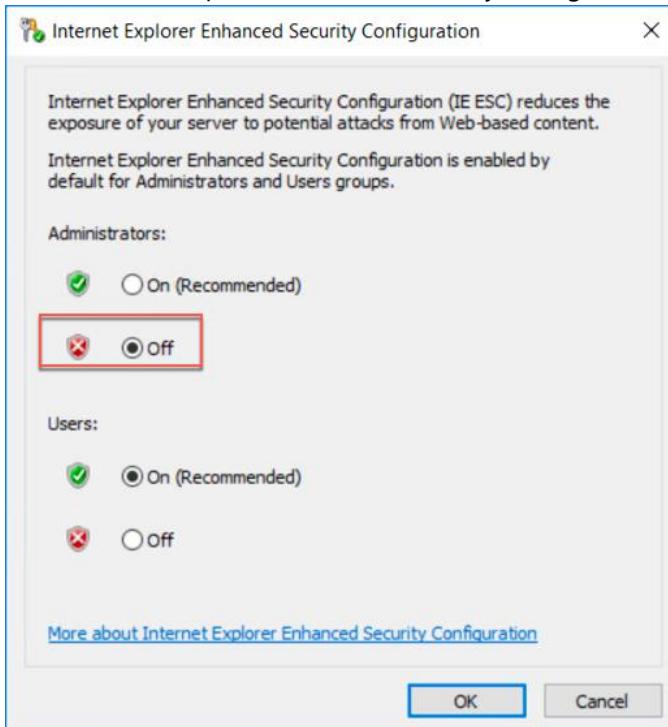
9. Once logged in, launch the Server Manager. This should start automatically, but you can access it via the Start menu if it does not start.



10. Select Local Server, the select On next to IE Enhanced Security Configuration.



11. In the Internet Explorer Enhanced Security Configuration dialog, select Off under Administrators, then select OK.



12. Close the Server Manager.

Task 4: Install Chrome on LabVM

In this task, you will install the Google Chrome browser on your Lab VM.

1. On your Lab VM, open a web browser, and navigate to <https://www.google.com/chrome/browser/desktop/index.html>, and select Download Chrome.

DOWNLOAD CHROME

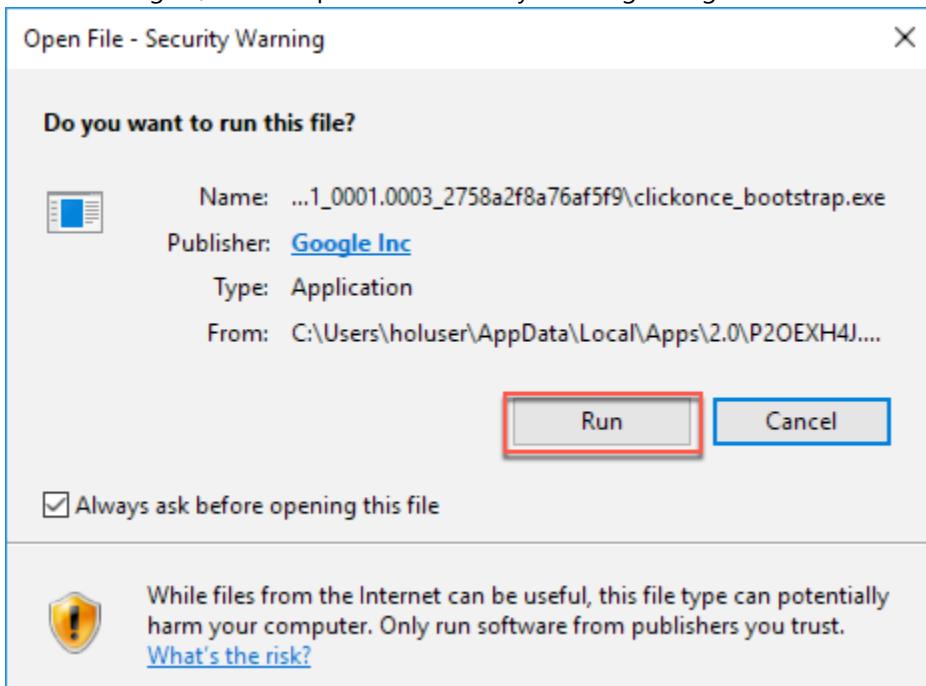
2. Select Accept and Install on the terms of service screen.

The screenshot shows the 'Download Chrome for Windows' page. At the top, it says 'For Windows 10/8.1/8/7 64-bit'. Below that is the 'Google Chrome Terms of Service' section. It contains a paragraph about the terms of service and a heading '1. Your relationship with Google'. Under this heading, there is a dropdown menu showing the first item: '1.1 Your use of Google's products, software, services and web sites (referred to collectively as the "Services" in this document and excluding any services provided to you by Google under a separate written agreement) is'. To the right of this is a 'Printer-friendly version' link. Below the dropdown is a checked checkbox for 'Help make Google Chrome better by automatically sending usage statistics and crash reports to Google.' followed by a 'Learn more' link. At the bottom is a large blue button labeled 'ACCEPT AND INSTALL'.

3. Select Run on the Application Run – Security Warning dialog.

The screenshot shows the 'Application Run - Security Warning' dialog. It asks 'Do you want to run this application?'. Below this are sections for 'Name:' (Google Installer), 'From (Hover over the string below to see the full domain):' (dl.google.com), and 'Publisher:' (Google Inc). At the bottom are two buttons: 'Run' (which is highlighted with a red box) and 'Don't Run'. A warning message at the bottom left says: 'While applications from the Internet can be useful, they can potentially harm your computer. If you do not trust the source, do not run this software. [More Information...](#)'.

4. Select Run again, on the Open File – Security Warning dialog.



5. Once the Chrome installation completes, a Chrome browser window should open. For ease, you can use the instructions in that window to make Chrome your default browser.

Task 5: Install Service Fabric SDK for Visual Studio

In this task, you will install the latest Service Fabric SDK for Visual Studio 2017 on your Lab VM.

1. On your Lab VM, open a browser, and navigate to <https://docs.microsoft.com/azure/service-fabric/service-fabric-get-started>.
2. Scroll down on the page to the Install the SDK and tools section and select Install the Microsoft Azure Service Fabric SDK under the To use Visual Studio 2017 heading.

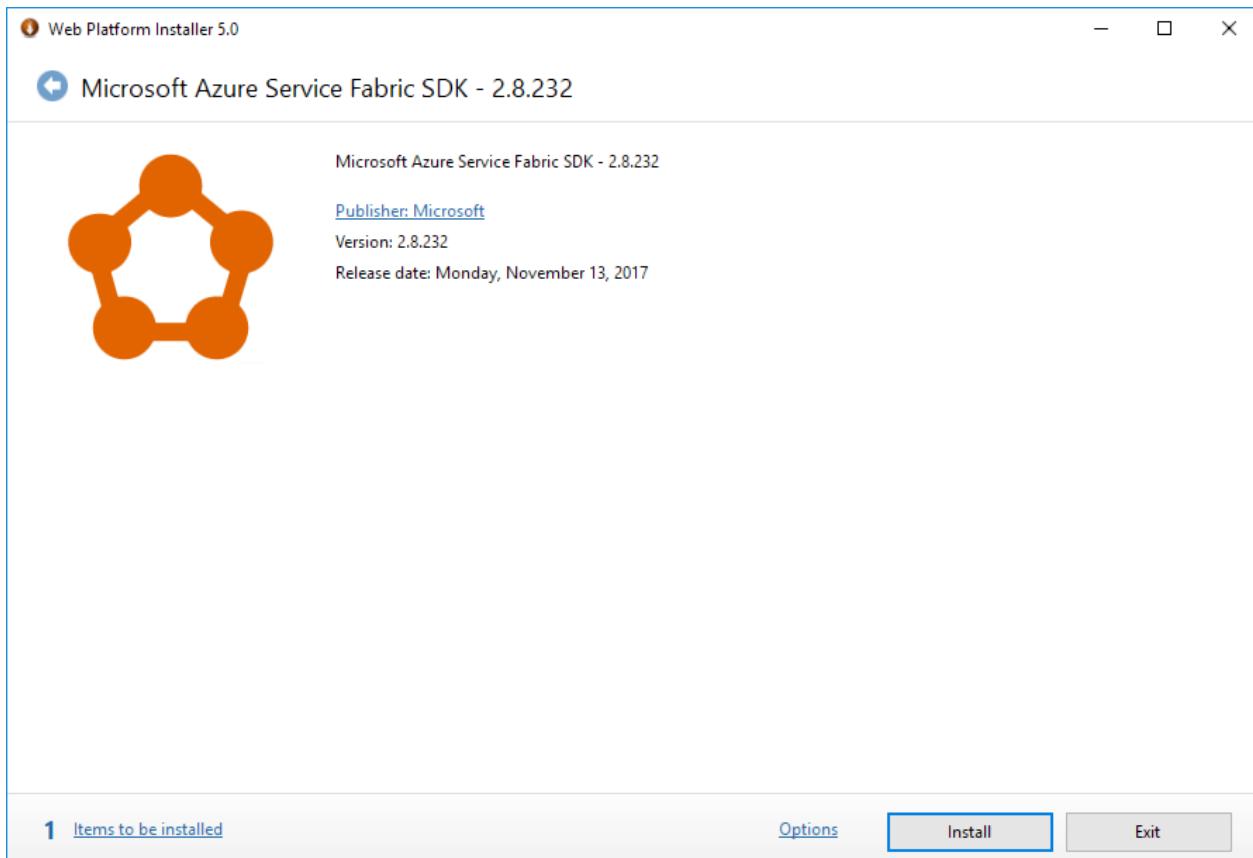
Install the SDK and tools

To use Visual Studio 2017

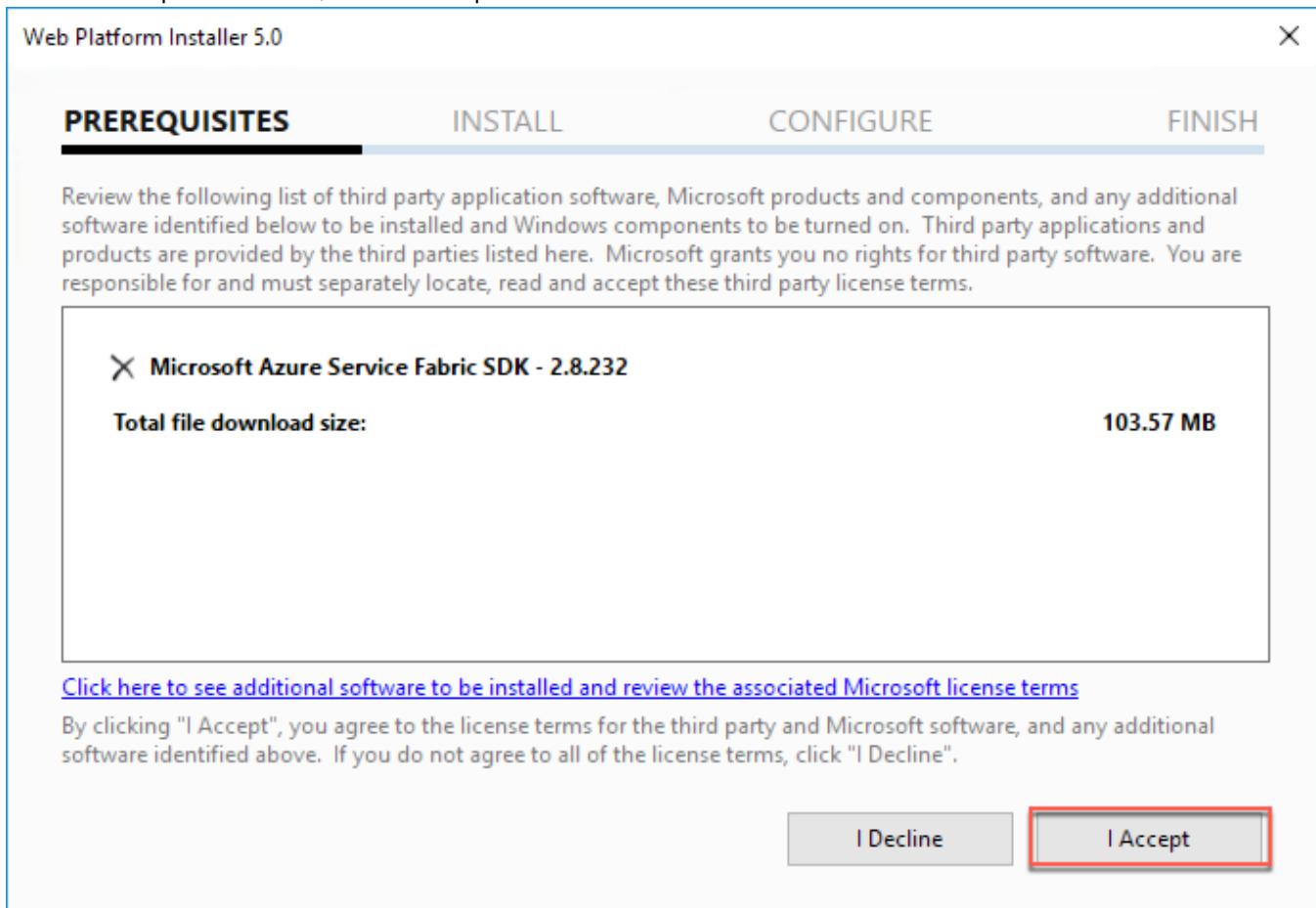
Service Fabric Tools are part of the Azure Development workload in Visual Studio 2017. Enable this workload as part of your Visual Studio installation. In addition, you need to install the Microsoft Azure Service Fabric SDK, using Web Platform Installer.

- [Install the Microsoft Azure Service Fabric SDK](#)

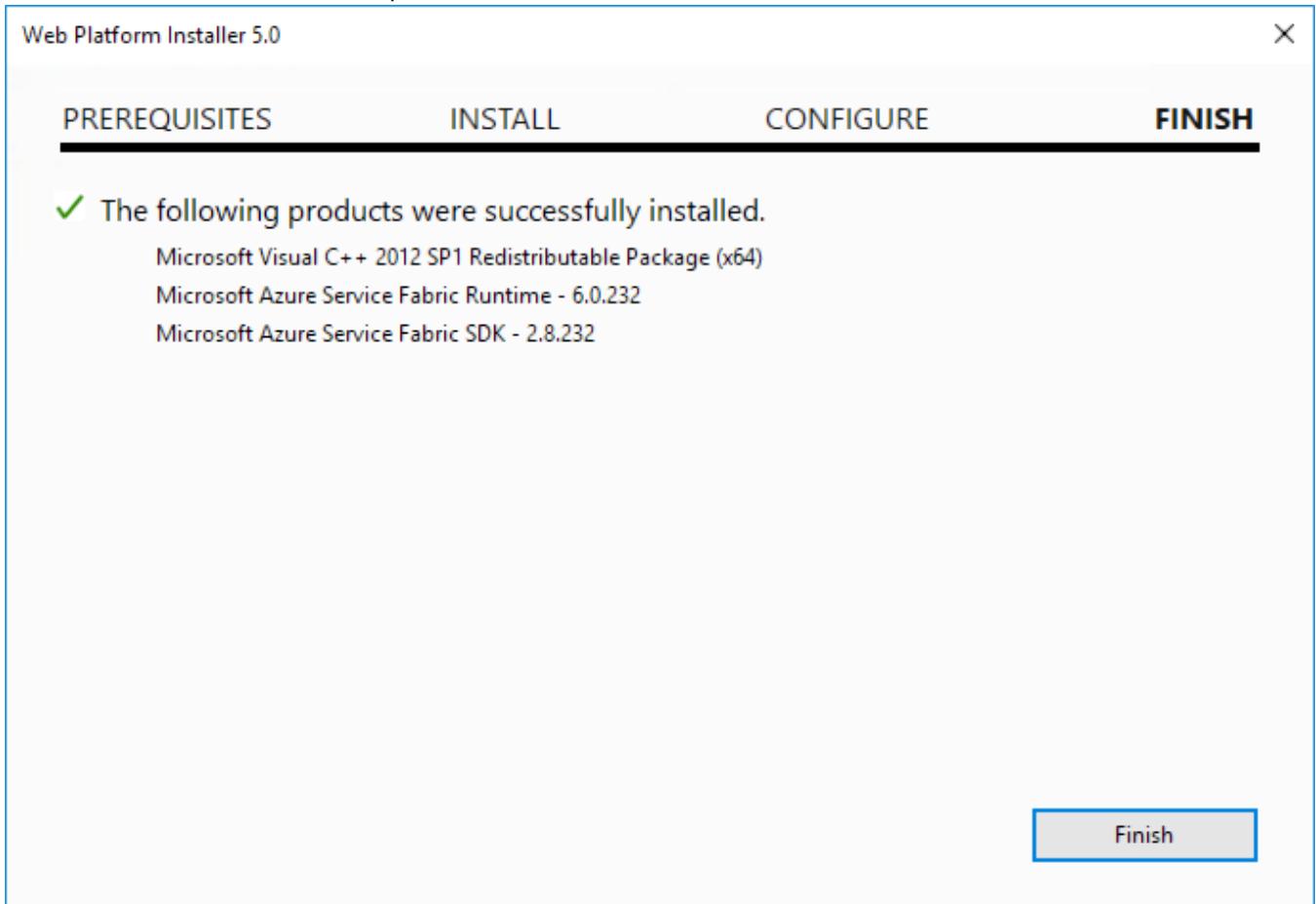
3. Run the downloaded executable and select Install in the Web Platform Installer screen.



4. On the Prerequisites screen, select I Accept.



5. Select Finish when the install completes.



6. Select Exit on the Web Platform installer to close it.
7. Restart the VM to complete the installation and start the local Service Fabric cluster service.

Task 6: Validate Service Fabric ports

Occasionally, when you create a new Service Fabric Cluster using the portal, the ports that you requested are not created. This will become evident when you try to deploy and run the Web App, because the required ports will not be accessible through the cluster.

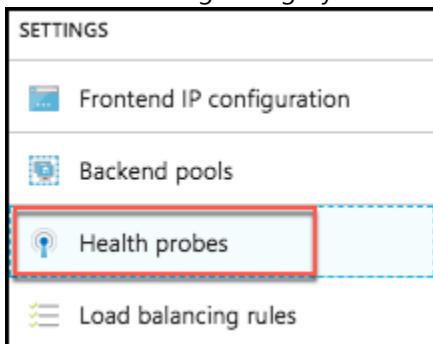
In this task, you will validate that the ports are open and if not, fix the issue.

1. In the Azure portal, navigate to the Resource Group you created previously, and where you created the cluster. If your Service Fabric cluster is still deploying, do not proceed to the next step until the deployment is completed.

2. Select the load balancer from the list of resources in the resource group.

	NAME ↑↓	TYPE ↑↓	LOCATION
<input type="checkbox"/>	contosoeventssf-kb	Service Fabric cluster	West US 2
<input type="checkbox"/>	handsonlabsdiag571	Storage account	West US 2
<input type="checkbox"/>	hands-on-labs-vnet	Virtual network	West US 2
<input type="checkbox"/>	LabVM	Virtual machine	West US 2
<input type="checkbox"/>	LabVM_OsDisk_1_59445148b9bf45e2b7dfca434a5d3cea	Disk	West US 2
<input type="checkbox"/>	labvm310	Network interface	West US 2
<input type="checkbox"/>	LabVM-ip	Public IP address	West US 2
<input type="checkbox"/>	LabVM-nsg	Network security group	West US 2
<input type="checkbox"/>	LB-contosoeventssf-kb-Web	Load balancer	West US 2
<input type="checkbox"/>	LBIP-contosoeventssf-kb-0	Public IP address	West US 2

3. Under the Settings category in the left-hand menu, select Health Probes.



4. Verify if a probe exists for port 8082, and that it is "Used By" a load balancing rule. If both of these are true, you can skip the remainder of this task. Otherwise, proceed to the next step to create the probe and load-balancing rule.

NAME	PROTOCOL	PORT	USED BY
AppPortProbe1	TCP	8082	AppPortLBRule1
FabricGatewayProbe	TCP	19000	LBRule
FabricHttpGatewayProbe	TCP	19080	LBHttpRule

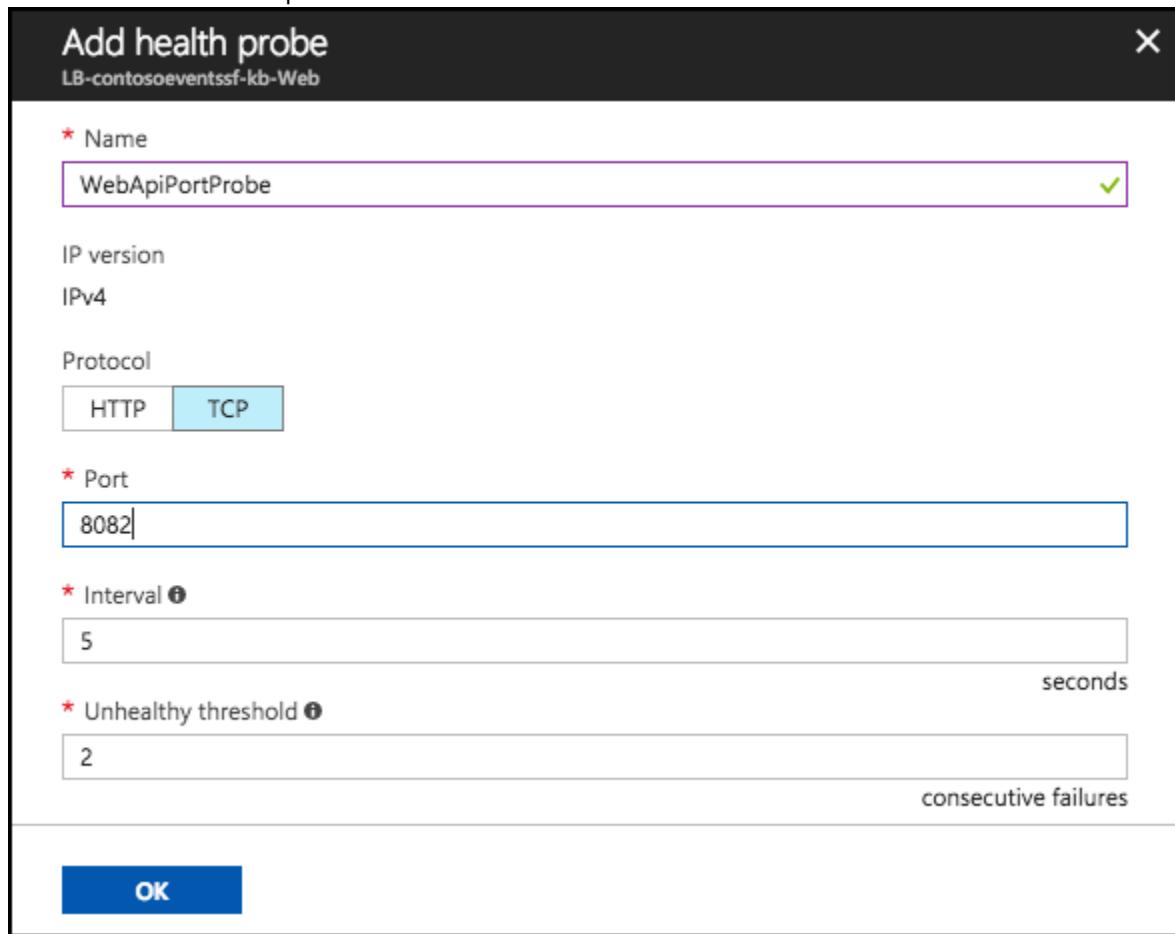
5. Select +Add on the Health Probes blade.



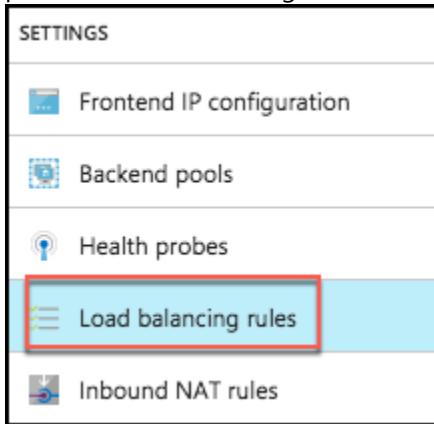
6. On the Add health probe blade, enter the following:

- Name: Enter WebApiPortProbe
- Protocol: Select TCP
- Port: Enter 8082
- Interval: Leave the default value

- e. Unhealthy threshold: Leave the default value
- f. Select OK to create the probe.



7. Once the Health probe is added (this can take a few minutes to update), you will create a rule associated with this probe. Under the Settings block in the left-hand menu, select Load balancing rules.



8. Select +Add on the Load balancing rules blade.



9. On the Add Load balancing rules blade, enter the following:

- a. Name: Enter LBWebApiPortRule
- b. IP Version: Leave IPv4 selected
- c. Frontend IP address: Leave the default value selected

- d. Protocol: Leave as TCP
- e. Port: Set to 8082
- f. Backend port: Set to 8082
- g. Backend pool: Leave the default value selected
- h. Health Probe: Select the WebApiPortProbe you created previously
- i. Leave the default values for the remaining fields, and Select OK.

Add load balancing rule

LB-contosoeventssf-kb-Web

* Name
LBWebApiPortRule

* IP Version
 IPv4 IPv6

* Frontend IP address ⓘ
52.229.18.35 (LoadBalancerIPConfig)

Protocol
 TCP UDP

* Port
8082

* Backend port ⓘ
8082

Backend pool ⓘ
LoadBalancerBEAddressPool

Health probe ⓘ
WebApiPortProbe (TCP:8082)

Session persistence ⓘ
None

Idle timeout (minutes) ⓘ
4

Floating IP (direct server return) ⓘ

OK

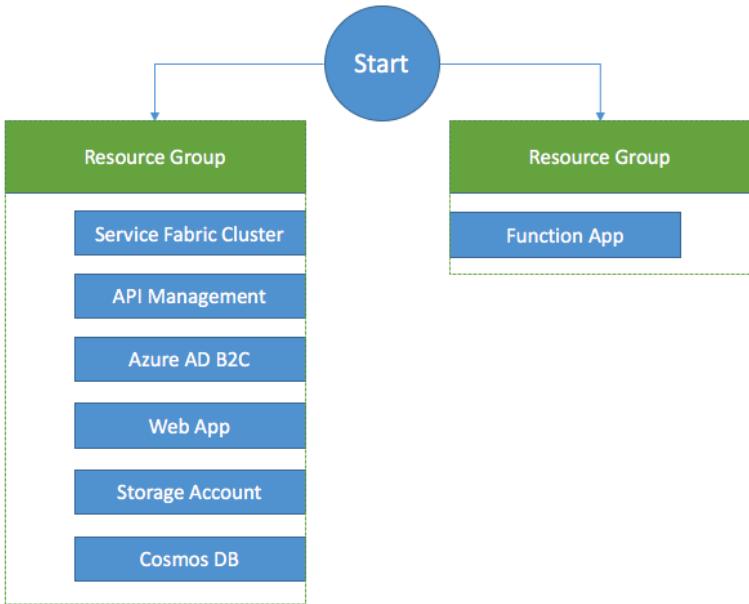
10. If you get an error notification such as "Failure to create probe", ignore this, but just go check that the probe indeed exists. It should. You now have a cluster ready to deploy to and expose 8082 as the Web API endpoint / port.

Exercise 1: Environment setup

Duration: 30 minutes

Contoso Events has provided a starter solution for you. They have asked you to use this as the starting point for creating the Ticket Order POC solution with Service Fabric.

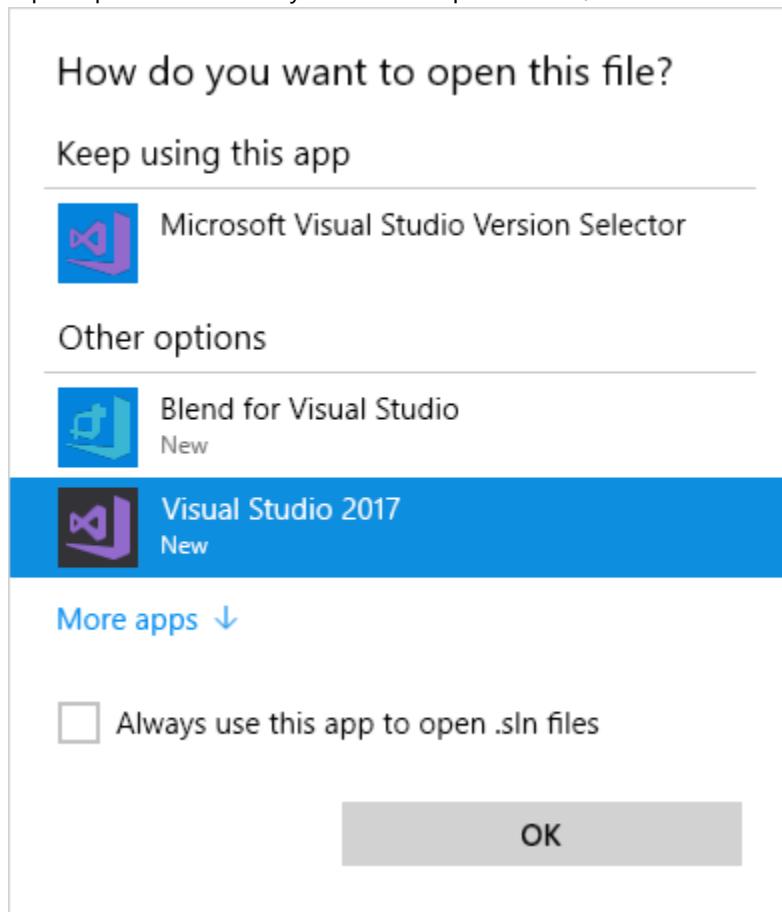
Because this is a “born in Azure” solution, it depends on many Azure resources. You will be guided through creating those resources before you work with the solution in earnest. The following figure illustrates the resource groups and resources you will create in this exercise.



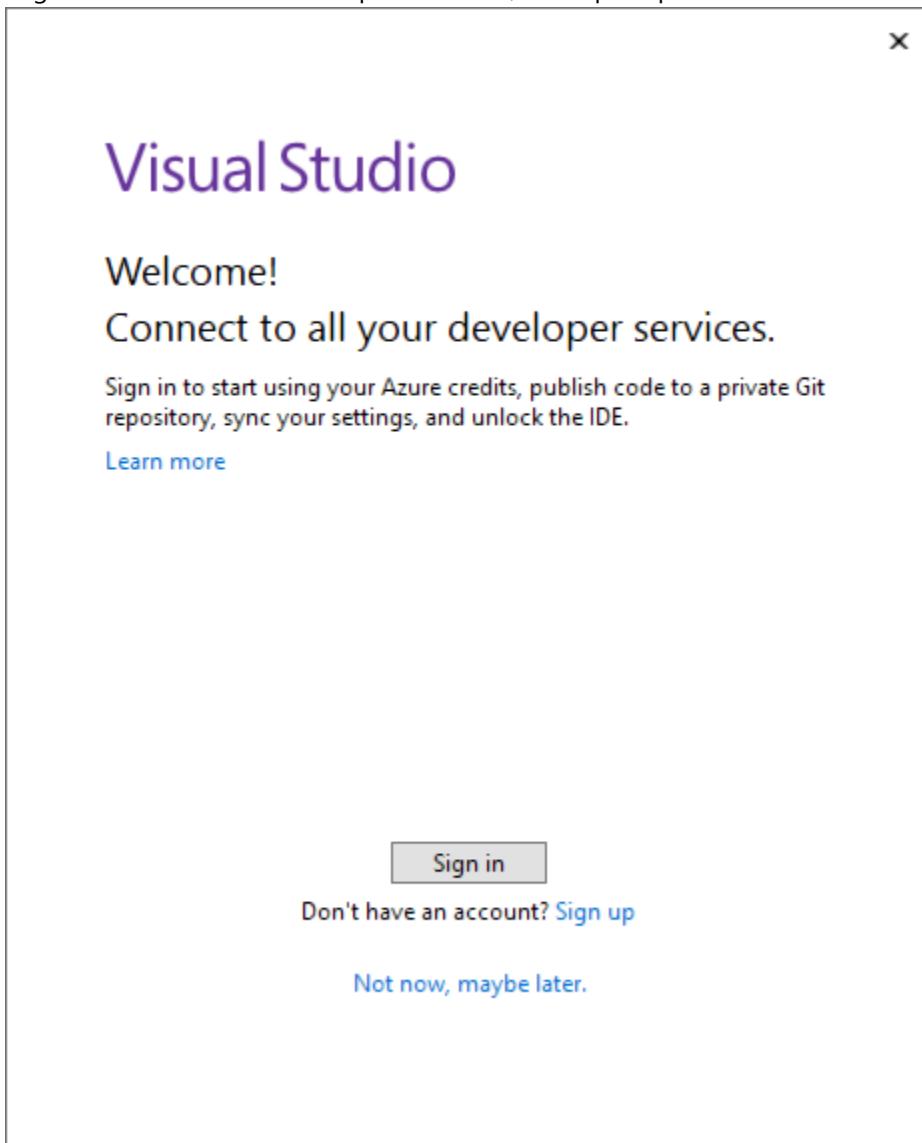
Task 1: Download and open the ContosoEventsPoC starter solution

1. On your Lab VM, download the starter project from <http://bit.ly/2gQHYSQ>. (Note: the URL is case sensitive, so you may need to copy and paste it into your browser’s address bar.)
2. Unzip the contents to the folder C:\handsonlab.
3. Locate the solution file (C:\handsonlab\src\ContosoEventsPOC.sln), and double-click it to open it with Visual Studio 2017.

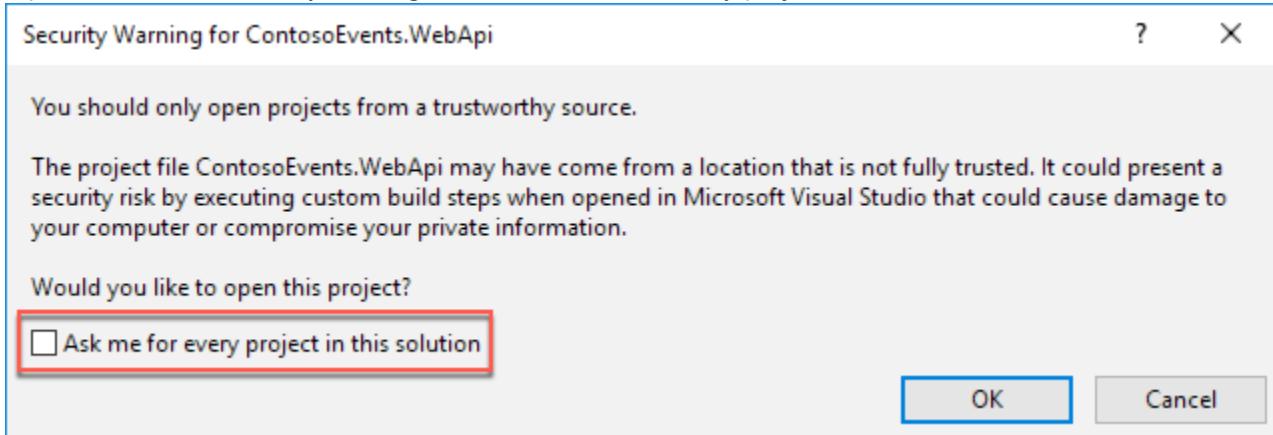
4. If prompted about how you want to open the file, select Visual Studio 2017, and select OK.



5. Log into Visual Studio or set up an account, when prompted.

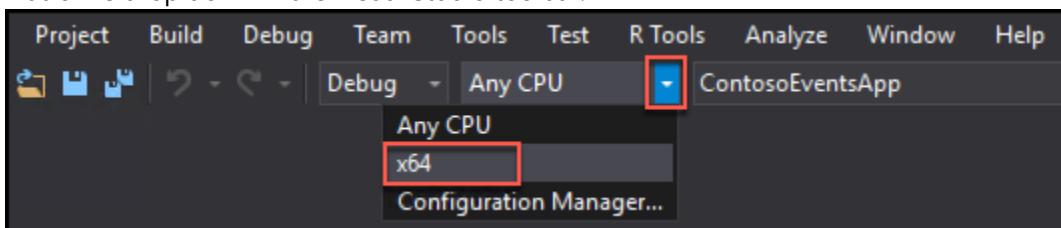


6. If presented with a security warning, uncheck Ask me for every project in this solution, and select OK.

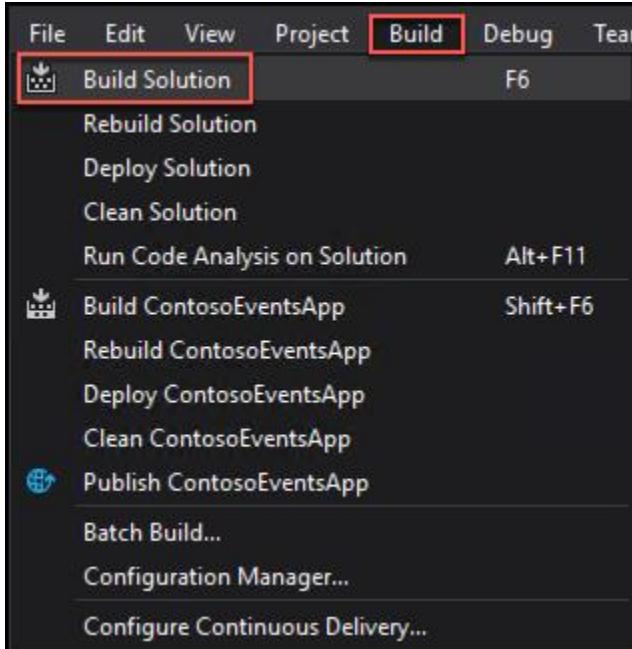


7. If you are missing any prerequisites (listed under [Requirements](#) above), you may be prompted to install these at this point.

- Before you attempt to compile the solution, set the configuration to x64 by selecting it from the Solution Platforms drop down in the Visual Studio toolbar.



- Build the solution, by selecting Build from the Visual Studio menu, then selecting Build Solution.

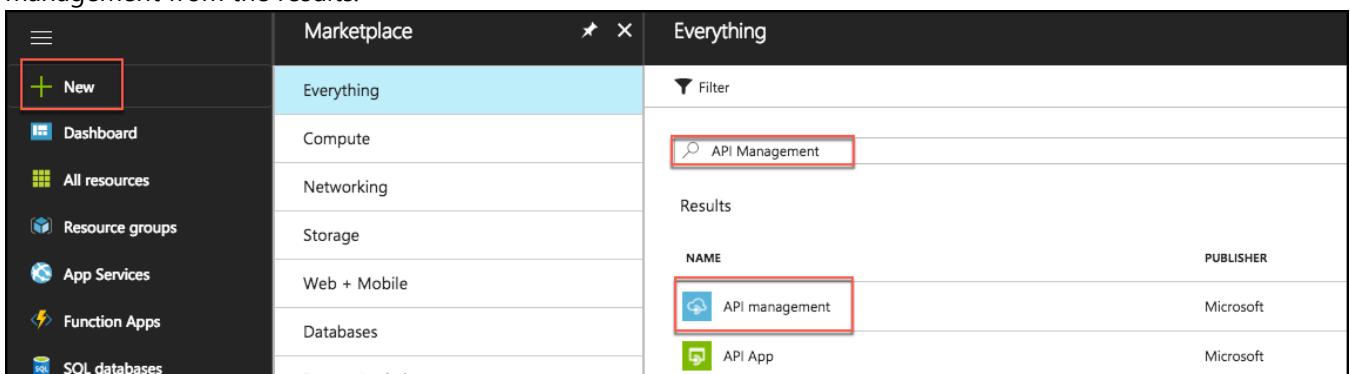


- You will have some compile-time errors at this point. These are expected, and will be fixed as you proceed with the hands-on lab.

Task 2: API Management

In this task, you will provision an API Management Service in the Azure portal.

- In the Azure portal, select +New, enter "API Management" into the Search the Marketplace box, then select API management from the results.



- In the API Management blade, select Create.
- In the API Management service blade, enter the following:
 - Name: Enter a unique name, such as contosoevents-SUFFIX

- b. Subscription: Choose your subscription
- c. Resource group: Select Use existing, and select the hands-on-labs resource group you created previously.
- d. Location: Select the same region used for the hands-on-labs resource group
- e. Organization name: Enter Contoso Events
- f. Administrator email: Enter your email address
- g. Pricing tier: Select Developer (No SLA)
- h. Select Create

The screenshot shows the 'API Management service' creation dialog box. It includes fields for Name (contosoevents-kb), Subscription (selected), Resource group (hands-on-labs), Location (West US 2), Organization name (Contoso Events), Administrator email (empty), Pricing tier (Developer (No SLA)), and a Pin to dashboard checkbox. At the bottom are 'Create' and 'Automation options' buttons.

API Management service

* Name
contosoevents-kb .azure-api.net

* Subscription

* Resource group
Create new Use existing

hands-on-labs

* Location
West US 2

* Organization name ⓘ
Contoso Events

* Administrator email ⓘ

Pricing tier (View full pricing details)
Developer (No SLA)

Pin to dashboard

Create Automation options

4. After the API Management service is provisioned, the service will be listed in the Resource Group. This may take up to 10-15 minutes, so move to Task 3 and return later to verify.

Task 3: Azure Active Directory B2C

In this section, you will provision an Azure Active Directory B2C tenant. You will use this if you do Bonus Exercise 10, so it is best to set it up in advance to avoid having to wait for provisioning.

1. In the Azure portal, select +New, and enter "Azure Active Directory" into the Search the Marketplace box, then select Azure Active Directory B2C from the results.

The screenshot shows the Azure Marketplace search results. On the left is a sidebar with a '+ New' button highlighted by a red box. The main area has a search bar with 'Azure Active Directory' typed in, also highlighted by a red box. Below the search bar, the results are listed under the heading 'Results'. There are two items: 'Azure Active Directory B2C' and 'Azure Active Directory'. Both items have their names and publishers ('Microsoft') displayed. The 'Azure Active Directory B2C' item is also highlighted by a red box.

2. Select Create in the Azure Active Directory B2C blade.
3. In the Create new B2C Tenant... blade, select Create a new Azure AD B2C Tenant.

The screenshot shows the 'Create new B2C Tenant or Link to existing Tenant' blade. It features two main options: 'Create a new Azure AD B2C Tenant' (highlighted by a red box) and 'Link an existing Azure AD B2C Tenant to my Azure subscription' (highlighted by a yellow box).

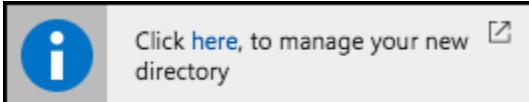
4. In the Azure AD B2C Create Tenant blade, enter:
 - a. Organization name: Enter a unique name, such as contosoeventsb2cSUFFIX
 - b. Initial domain name: Enter the same name as was entered for Organization name
 - c. Country or region: Select United States
 - d. Select Create

The screenshot shows the 'Azure AD B2C Create Tenant' blade. It contains the following fields:

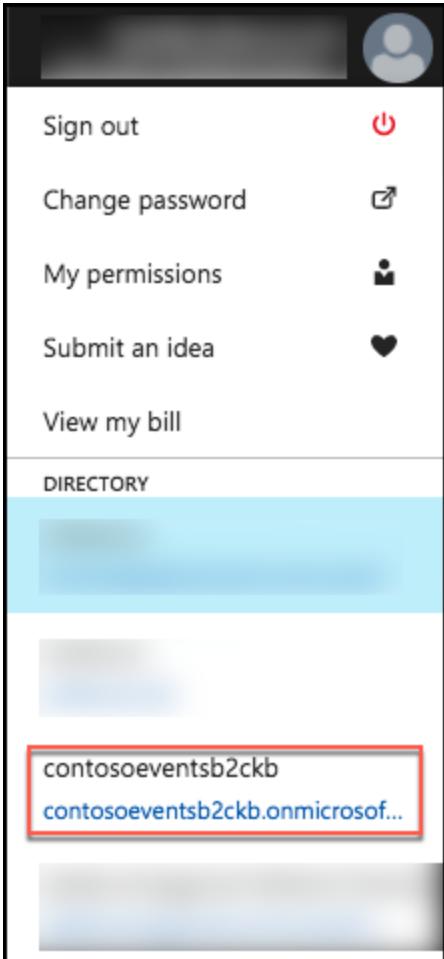
- * Organization name: contosoeventsb2ckb (highlighted by a purple box)
- * Initial domain name: contosoeventsb2ckb (highlighted by a purple box)
contosoeventsb2ckb.onmicrosoft.com
- Country or region: United States (highlighted by a purple box)

A large blue 'Create' button is at the bottom.

- When the tenant finishes provisioning, you can switch to your new tenant by select the manage your new directory link which will appear in the Azure AD B2C Create Tenant blade.



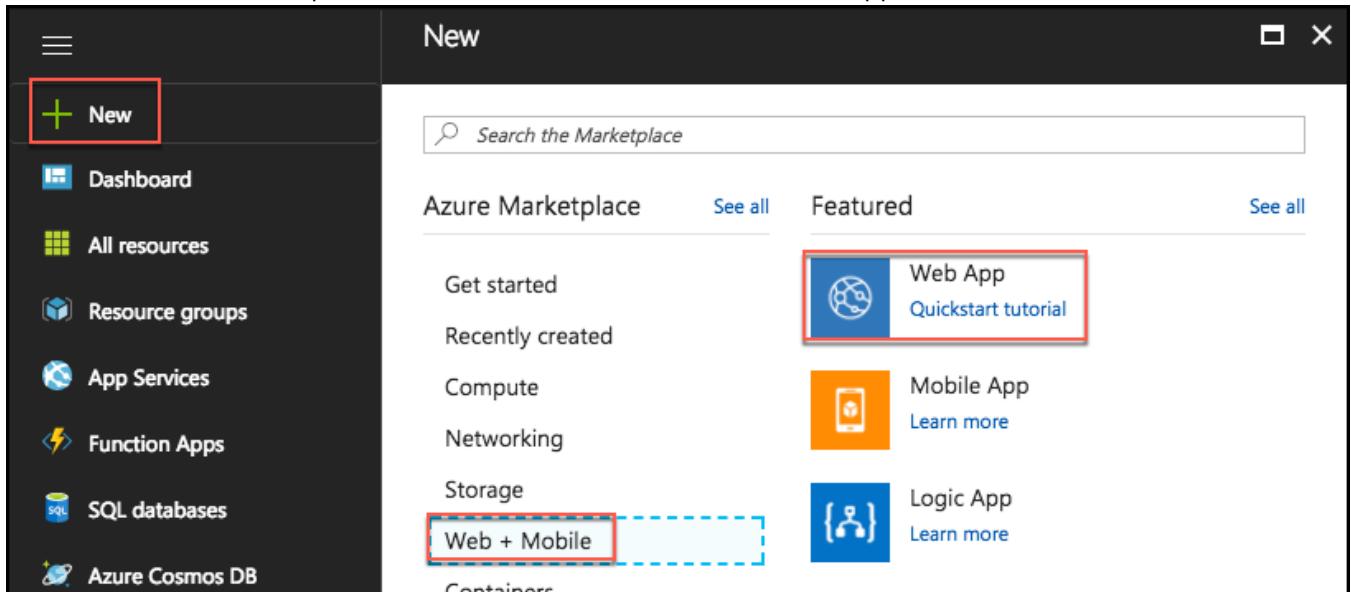
- It may take a few minutes to provision the new directory. In the meantime, feel free to move on to Task 4, and return later to verify. To find the directory later, you may need to refresh the portal. The directory can be found by clicking your logged in user name in the upper-right corner of the page.



Task 4: Web app

In these steps, you will provision a web app in a new App Service Plan.

1. Select +New in the Azure portal, select Web + Mobile, then select Web App.



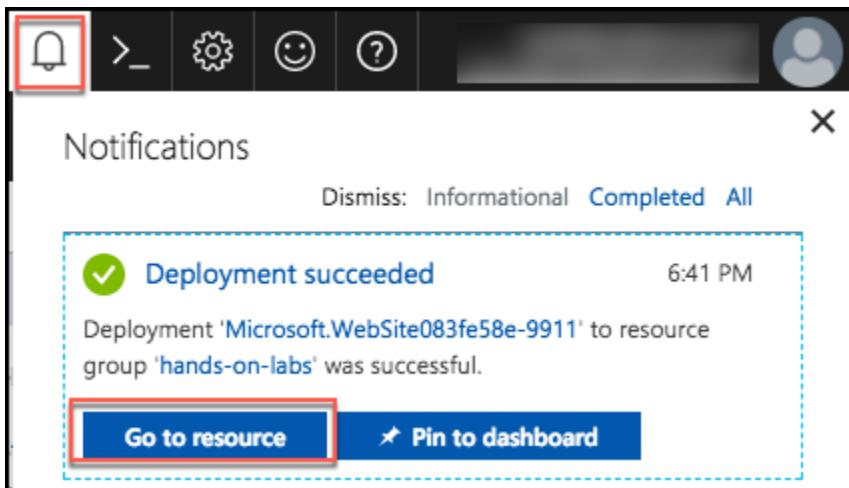
2. On the Create Web App blade, enter the following:

- a. App name: Enter a unique name, such as contosoeventsweb-SUFFIX
- b. Subscription: Select your subscription
- c. Resource group: Select Use existing, and select the hands-on-labs resource group created previously.
- d. OS: Select Windows
- e. App Service plan/location: Select this, select Create new
 - i. App service plan: Enter contosoeventsplan-SUFFIX
 - ii. Location: Select the same location you have been using for other resources in this lab
 - iii. Pricing tier: Select S1 Standard
 - iv. Select OK

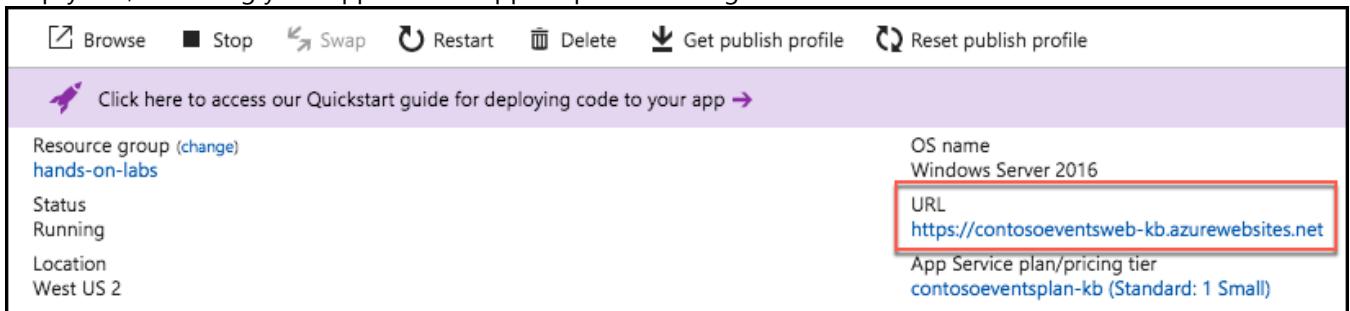
- f. Select Create to provision the web app

The screenshot shows the 'Create Web App' blade. On the left, the 'Web App' blade shows fields for App name (contosoeventsweb-kb), Subscription (Soliance MVP MSDN), Resource Group (hands-on-labs), OS (Windows), and App Service plan/Location (DefaultTier1ServerFarm(West US)). The 'App Service plan' tab is selected in the center, showing existing plans like ServicePlan69507b4-8322(S1) and DemoAppPlan(S1). On the right, the 'New App Service Plan' blade is open, showing fields for App Service plan (contosoeventsplan-kb), Location (West US 2), and Pricing tier (S1 Standard). The 'OK' button is at the bottom right of the 'New App Service Plan' blade.

3. You will receive a notification in the Azure portal when the web app deployed completes. From this, select Go to resource.



4. On the Web Apps Overview blade, you can URL used to access your web app. If you select this, it will open an empty site, indicating your App Services app is up and running.



Task 5: Function app

In this task, you will provision a Function app using a Consumption Plan. By using a Consumption Plan, you enable dynamic scaling of your Functions.

1. Select +New in the Azure portal, and enter "Function App" in the Search the Marketplace box, then select Function App from the results.

The screenshot shows the Azure Marketplace search results for 'Function App'. The search bar contains 'Function App'. The results table has columns for NAME, PUBLISHER, and more. The first result, 'Function App' by Microsoft, is highlighted with a red box. Other results include 'Functions Bot' by Microsoft.

NAME	PUBLISHER
Function App	Microsoft
Functions Bot	Microsoft

2. Select Create on the Function App blade.
 3. On the Create Function App blade, enter the following:
 a. App name: Enter a unique name, such as contosoeventsfn-SUFFIX
 b. Subscription: Select your subscription
 c. Resource group: Select Use existing, and select the hands-on-labs resource group created previously

- d. OS: Select Windows
- e. Hosting Plan: Select Consumption Plan
- f. Location: Select the same location as the hands-on-labs resource group
- g. Storage: Leave Create new selected, and accept the default name
- h. Select Create to provision the new function app

Function App X

Create

* App name
contosoeventsfn-kb .azurewebsites.net

* Subscription

* Resource Group i
 Create new Use existing
hands-on-labs ▼

* OS Windows Linux (Preview)

* Hosting Plan i
Consumption Plan ▼

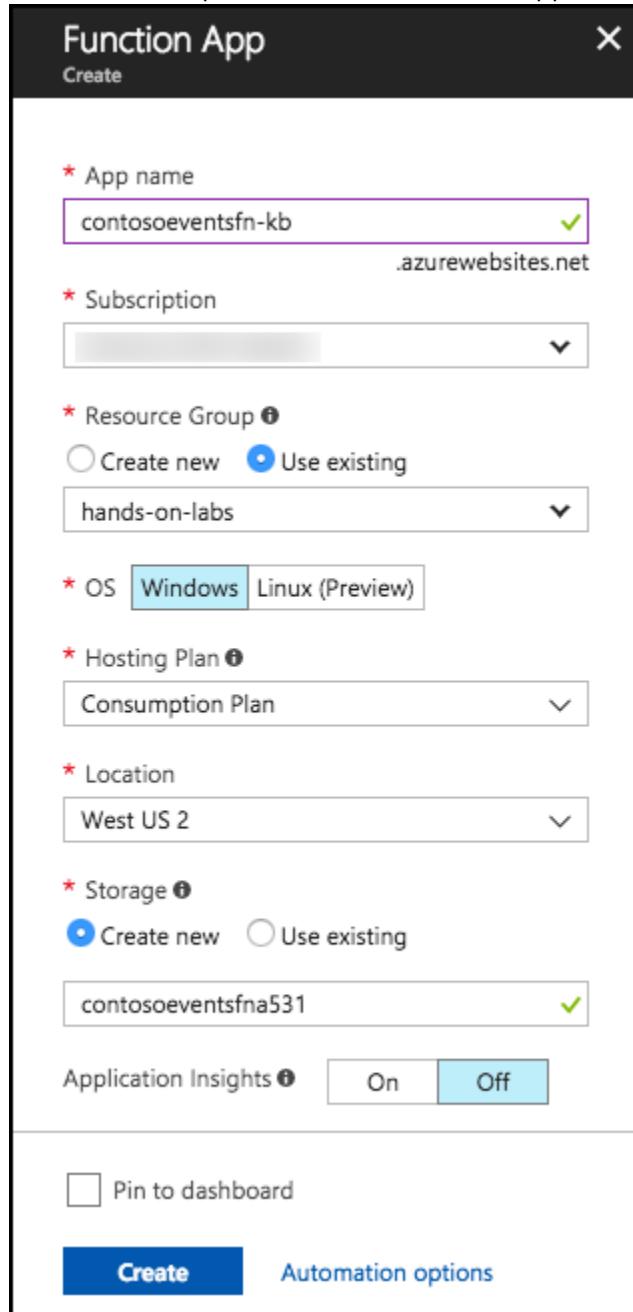
* Location
West US 2 ▼

* Storage i
 Create new Use existing
contosoeventsfn531 .contosoeventsfn531.blob.core.windows.net

Application Insights i On Off

Pin to dashboard

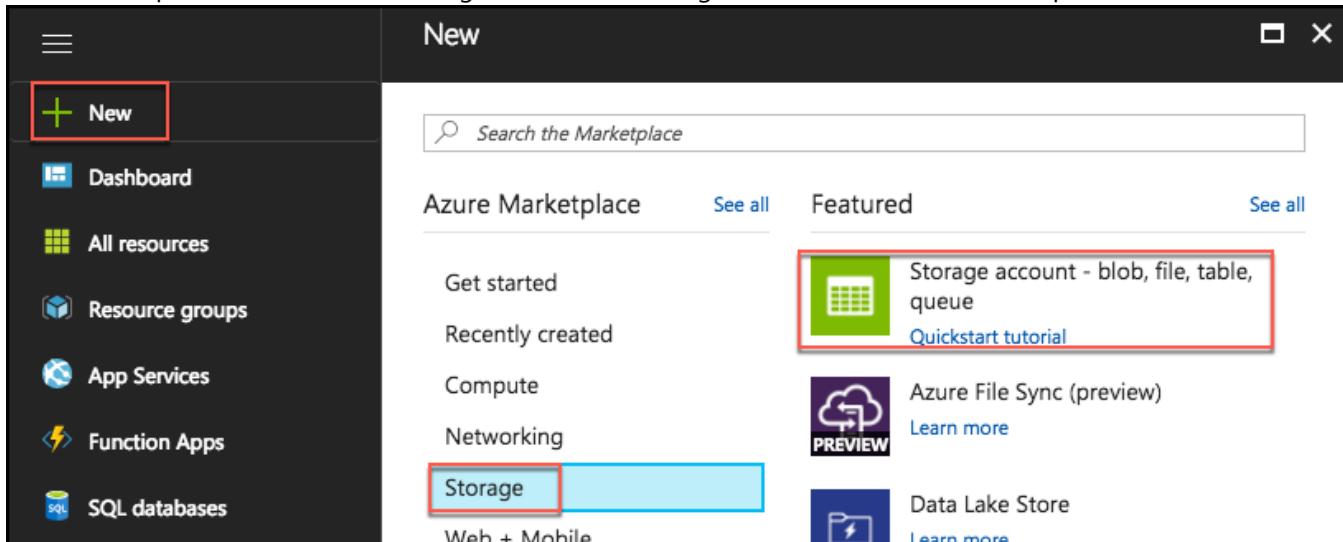
Create [Automation options](#)



Task 6: Storage account

In this section, you will create a Storage account for the application to create and use queues required by the solution.

1. In the Azure portal, select +New, Storage, then select Storage account – blob, file, table, queue under Featured.



2. In the Create Storage account blade, enter the following:

- Name: Enter a unique name, such as contosoeventsSUFFIX
- Deployment model: Select Resource manager
- Account kind: Select Storage (general purpose v1)
- Performance: Select Standard
- Replication: Select Locally-redundant storage (LRS)
- Secure transfer required: Leave Disabled
- Subscription: Select your subscription
- Resource group: Select Use existing, and select the hands-on-labs resource group created previously
- Location: Select the same region used for the hands-on-labs resource group
- Virtual networks (Preview): Leave Disabled

k. Select Create

Create storage account

The cost of your storage account depends on the usage and the options you choose below.
[Learn more](#)

*** Name** contosoeventskb .core.windows.net

Deployment model Resource manager Classic

Account kind Storage (general purpose v1)

Performance Standard Premium

Replication Locally-redundant storage (LRS)

*** Secure transfer required** Disabled Enabled

*** Subscription** [dropdown]

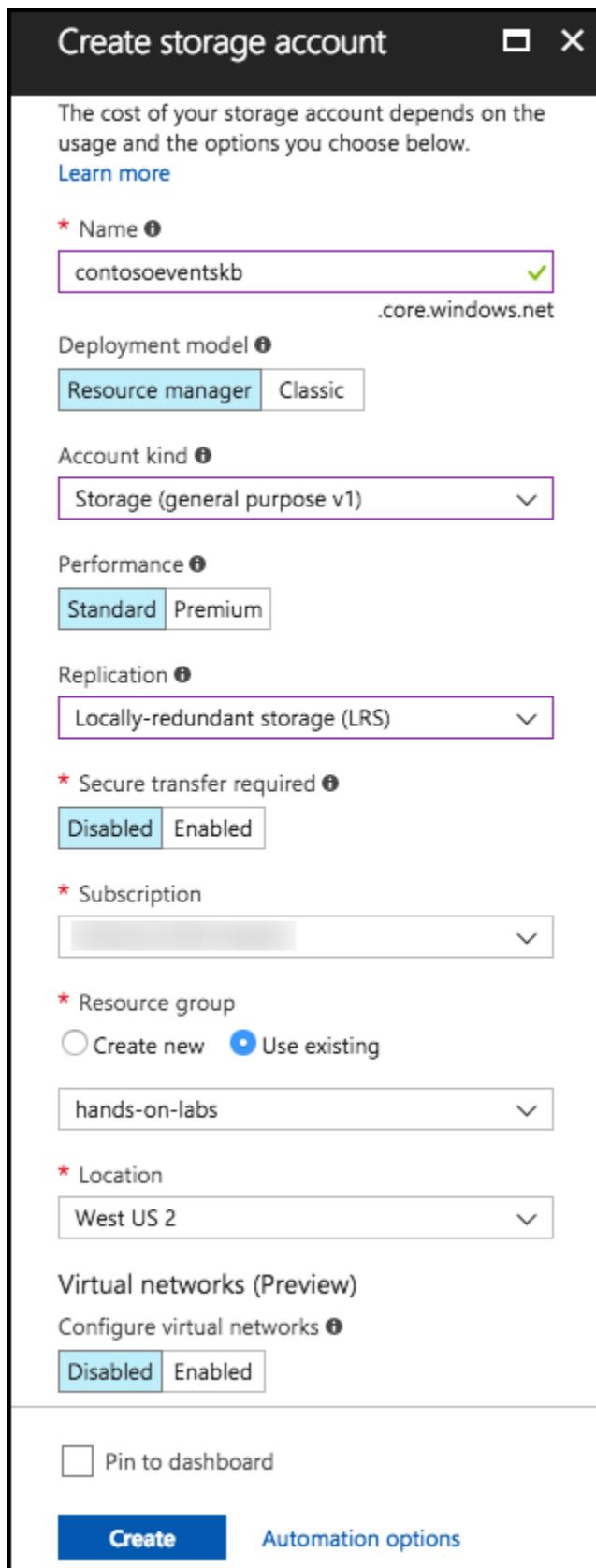
*** Resource group** Create new Use existing hands-on-labs

*** Location** West US 2

Virtual networks (Preview)
Configure virtual networks
Disabled Enabled

Pin to dashboard

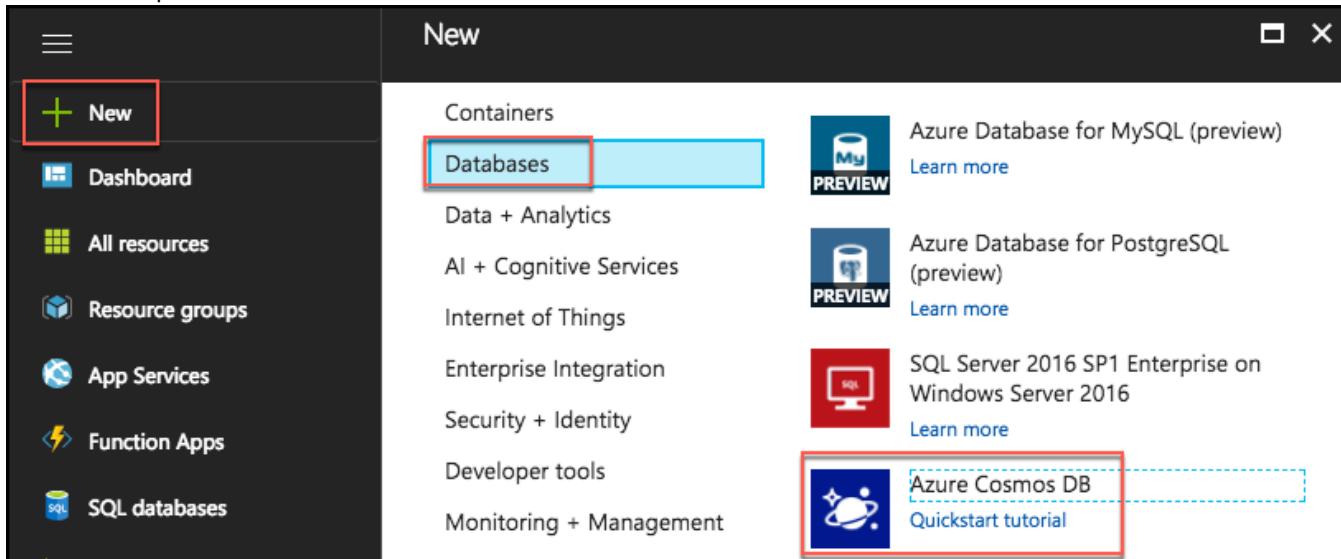
Create Automation options



Task 7: Cosmos DB

In this section, you will provision a Cosmos DB account, a Cosmos DB Database, and a Cosmos DB collection that will be used to collect ticket orders.

1. In the Azure portal, select +New, Databases, then select Azure Cosmos DB.



2. On the Azure Cosmos DB blade, enter the following:
 - a. ID: Enter a unique value, such as contosoeventsdb-SUFFIX
 - b. API: Select SQL
 - c. Subscription: Select your subscription
 - d. Resource group: Select Use existing, and select the hands-on-labs resource group previously created
 - e. Location: Select the location used for the hands-on-lab resource group. If this location is not available, select one close to that location that is available.
 - f. Enable geo-redundancy: Leave checked

- g. Select Create to provision the Cosmos DB

The screenshot shows the 'Azure Cosmos DB' creation dialog for a new account. The fields filled are:

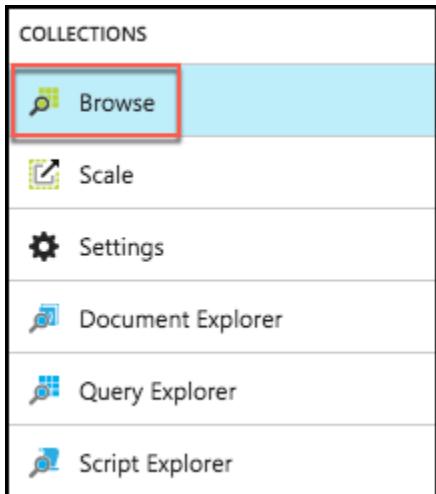
- ID:** contosoeventsdb-kb (highlighted with a green checkmark)
- API:** SQL
- Subscription:** (dropdown menu)
- Resource Group:** Use existing (radio button selected) hands-on-labs
- Location:** West US
- Enable geo-redundancy:** checked
- Pin to dashboard:** unchecked

At the bottom are two buttons: **Create** (blue) and **Automation options**.

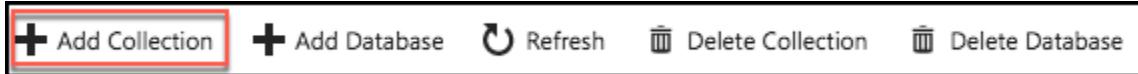
3. When the Cosmos DB account is ready, navigate to the hands-on-labs Resource Group, and select your Cosmos DB account from the list.

	NAME	TYPE	LOCATION
<input type="checkbox"/>	contosoeventsdb-kb	Azure Cosmos DB account	West US
<input type="checkbox"/>	contosoeventskb	Storage account	West US 2
<input type="checkbox"/>	contosoevents-kb	API Management service	West US 2
<input type="checkbox"/>	contosoeventsplan-kb	App Service plan	West US 2

4. On the Cosmos DB account blade, under Collections in the left-hand menu, select Browse.

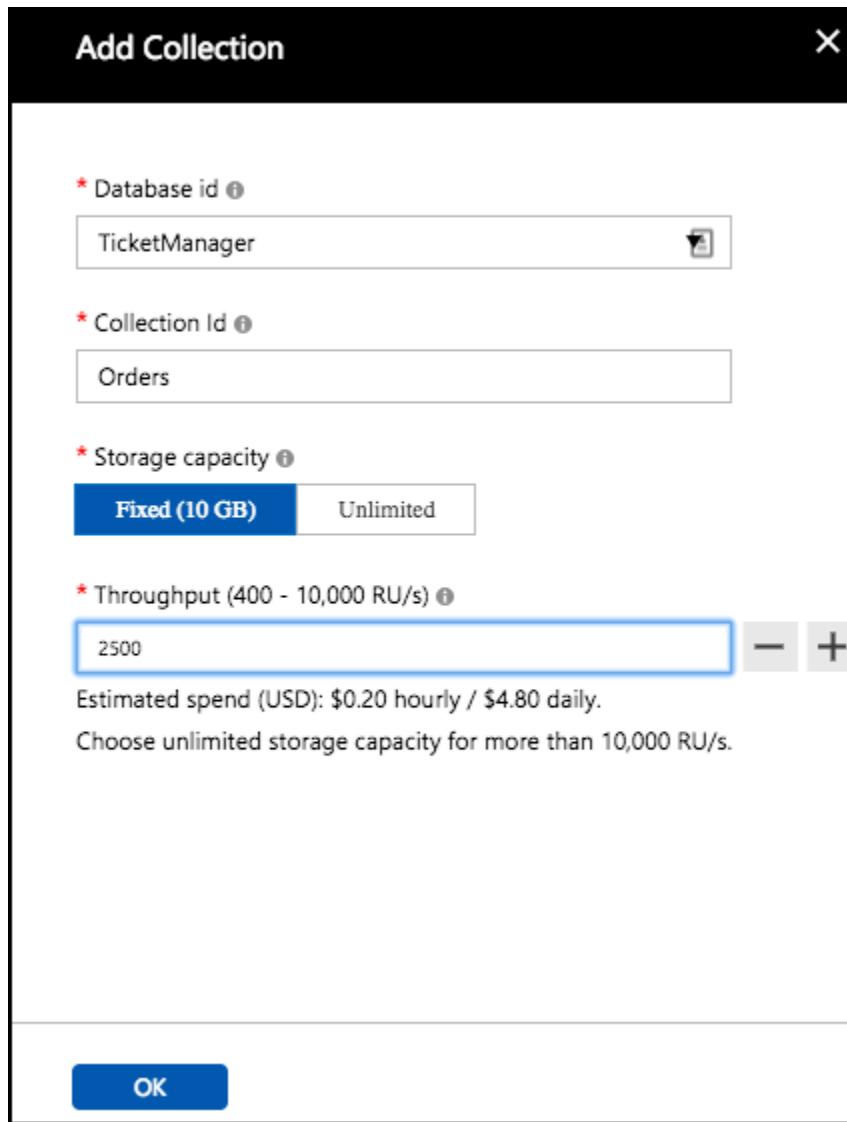


5. On the Browse blade, select +Add Collection.

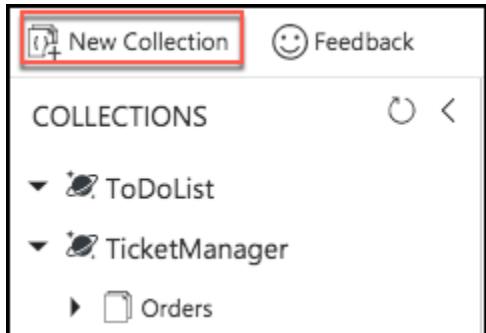


6. On the Add Collection dialog, enter the following:
- Database id: Enter TicketManager
 - Collection id: Enter Orders
 - Storage capacity: Select Fixed (10 GB)
 - Throughput: Enter 2500

- e. Select OK to create the new collection



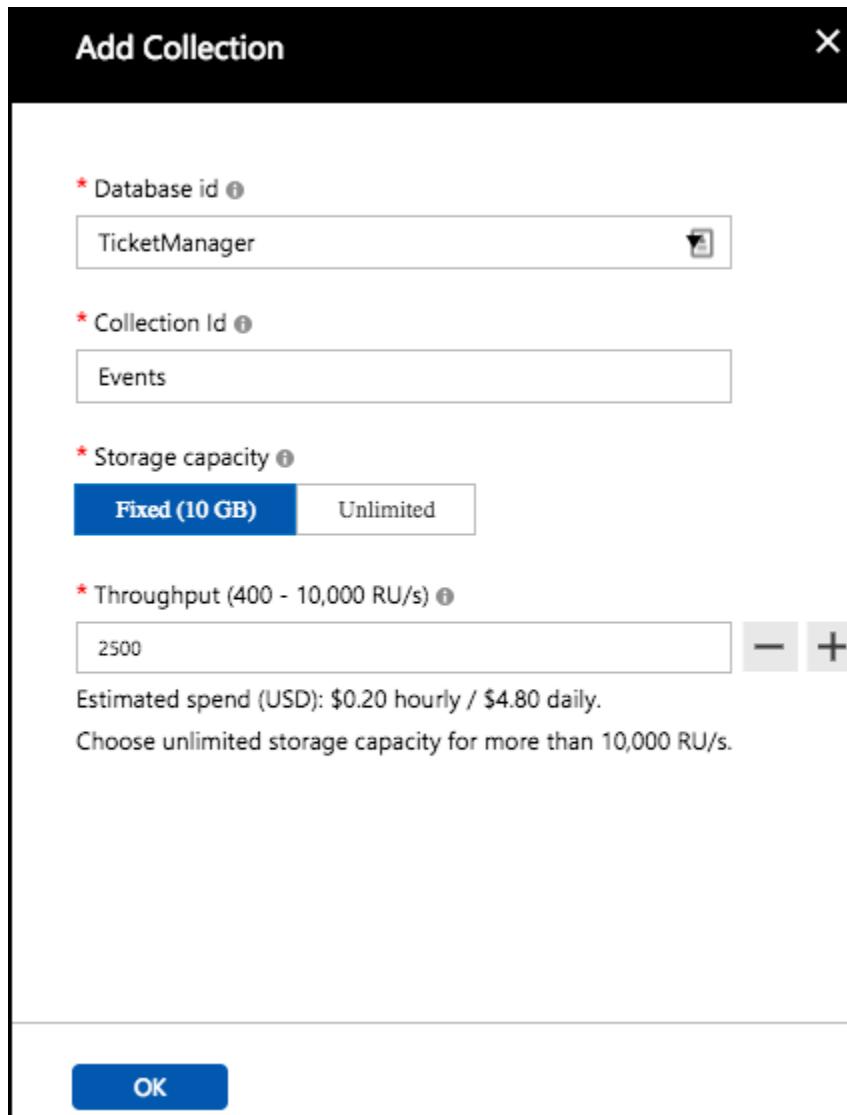
7. Select New collection from the new screen that appears.



8. In the Add Collection dialog, enter the following:

- Database id: Enter TicketManager
- Collection id: Enter Events
- Storage capacity: Select Fixed (10 GB)
- Throughput: Enter 2500

- e. Select OK to create the new collection



9. You will be able to see that the two collections exist in the new database.

The screenshot shows the Azure portal interface for a database. It lists 'COLLECTIONS' under 'TicketManager'. Two collections are shown: 'Orders' and 'Events', both highlighted with a red border.

New Collection Feedback

COLLECTIONS

ToDoList

TicketManager

Orders

Events

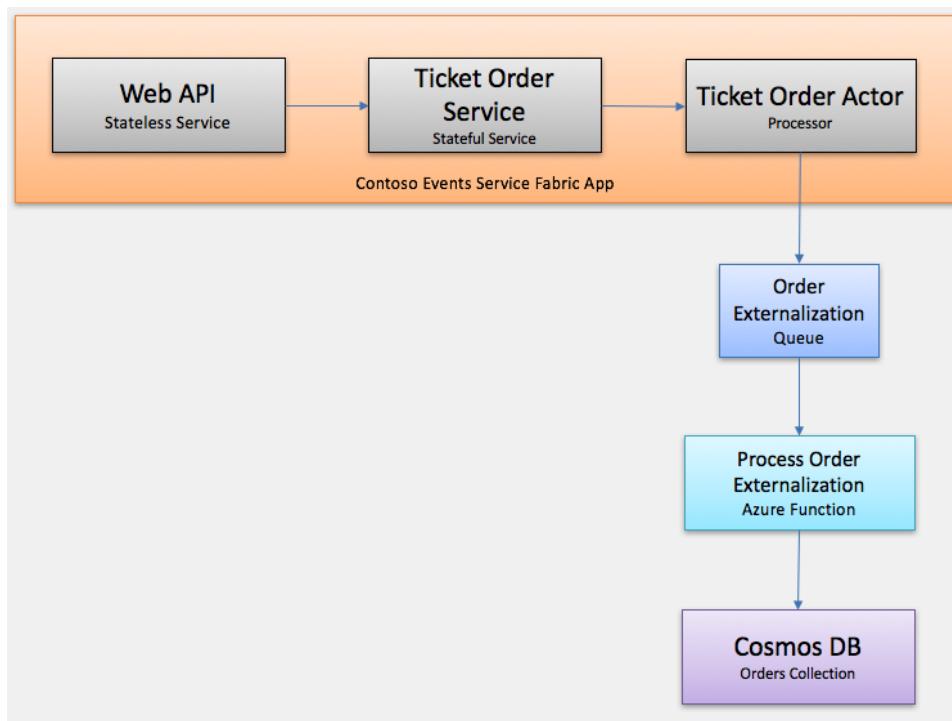
Exercise 2: Placing ticket orders

Duration: 30 minutes

The agreed-upon design with Contoso Events involves queuing ticket orders, and executing them asynchronously. A stateless Web API service receives the request, and queues it to a stateful service. An actor processes the request, and persists the order in its state.

The design also calls for saving the state of the ticket order to a Cosmos DB collection for ad hoc queries. This exercise will guide you through adding configurations that will light up the actor code that externalizes its state to a storage queue. In addition, you will set up the Function App to read from the queue, and persist the order to the Orders collection of the Cosmos DB instance you created.

Note: The code to write to the storage queue is already in place within the actor, so setting up configuration keys is the only requirement to lighting up that feature.



Task 1: Run the solution

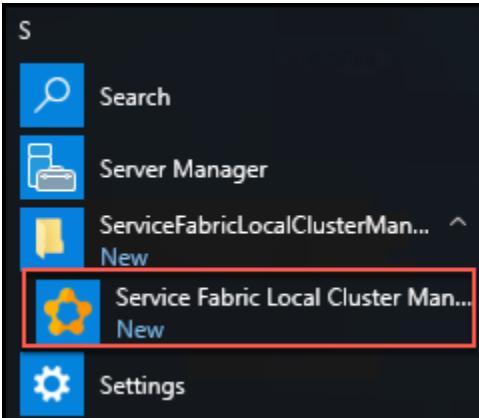
The purpose of this task is to make sure that the source code compiles, and that you can publish to a local cluster. To make sure that the app is running flawlessly, you will run some API tests and access the Service Fabric explorer.

Note: Not all features are in place, but you will be able to see that the application can run.

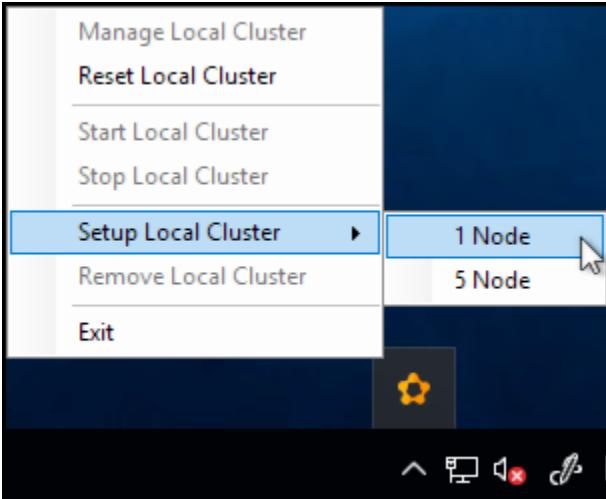
1. On the Lab VM, make sure the local Service Fabric environment is running by selecting the arrow in the system tray, and checking for the appearance of the Service Fabric icon. Note: You may need to restart your VM if you've recently installed the Service Fabric Local Cluster Manager, for the icon to appear.



2. If it is not there, you will need to start the local Service Fabric cluster. To do this, select the Start menu, scroll down to the apps listed under "S," and select Service Fabric Local Cluster Manager. The icon (Step 1) should now appear.



3. Right-click the system tray icon, and select Setup Local Cluster, 1 Node.

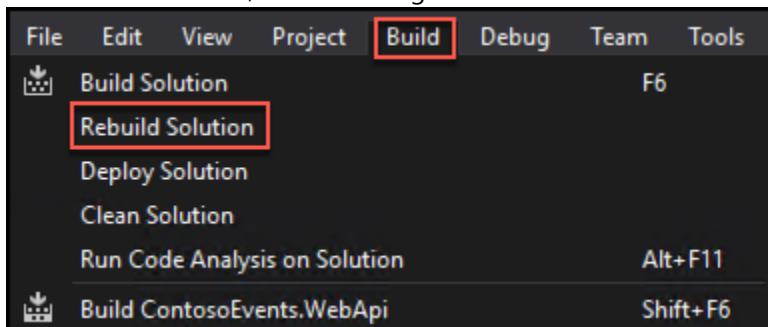


4. You should see a message that the local cluster is setting up. **Note:** Sometimes you will see an error message appear indicating the cluster setup is retrying.

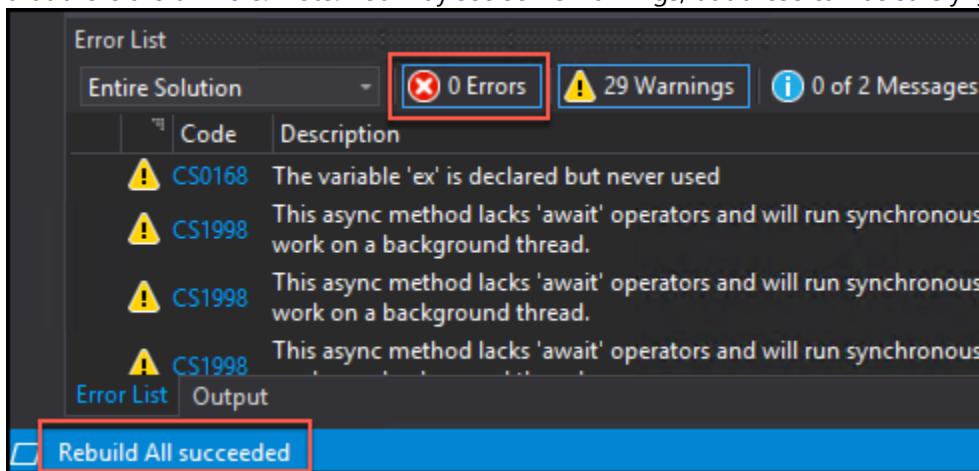


5. Open the ContosoEventsPoc solution in Visual Studio, if it is not still open from the previous exercise.

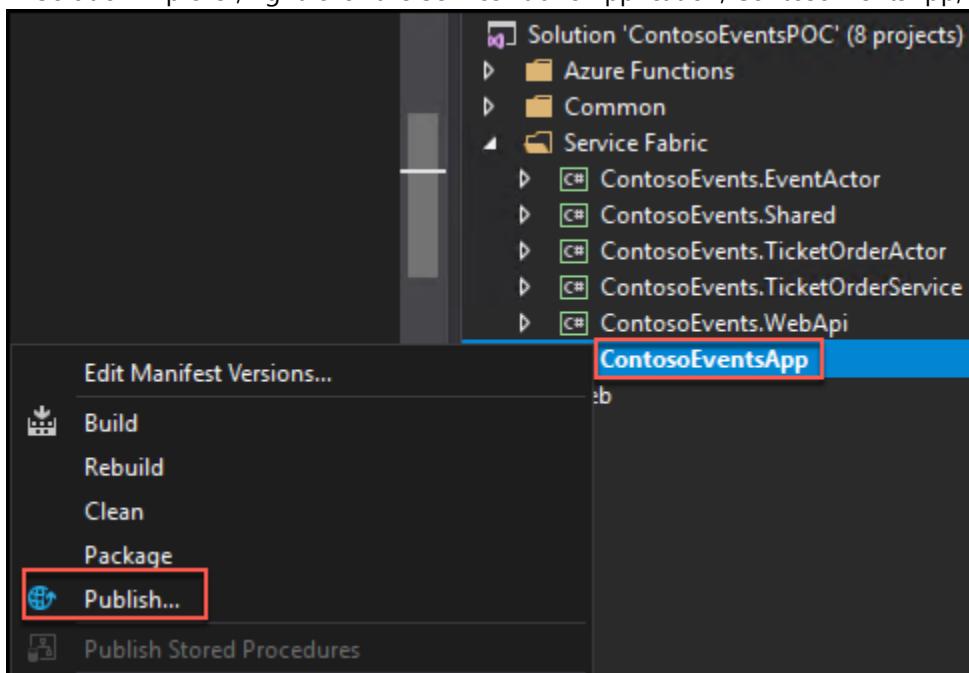
6. Rebuild the solution to resolve all NuGet packages, and to make sure there are no compilation errors, by selecting Build from the menu, then selecting Rebuild Solution.



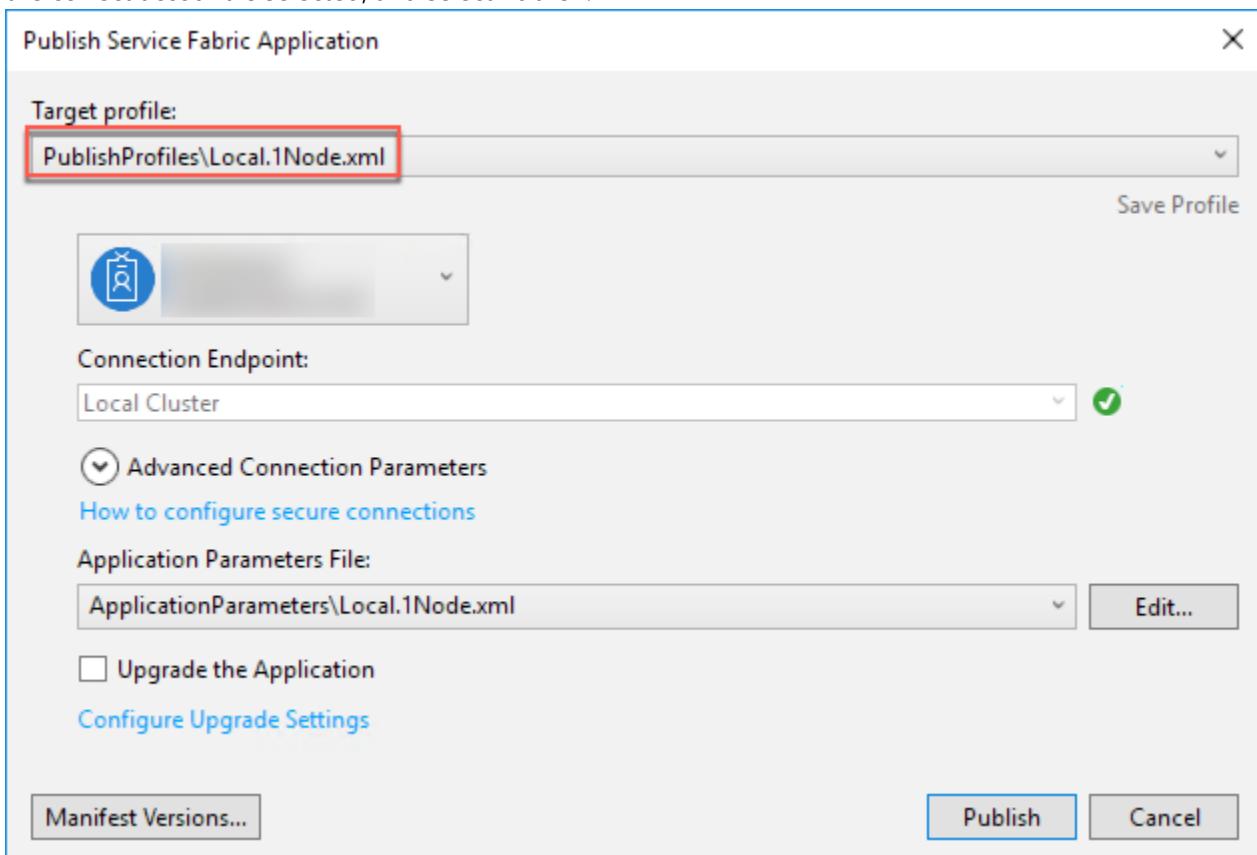
7. After the rebuild, you should see a Rebuild All succeeded message in the bottom left corner of Visual Studio, and that there are 0 Errors. Note: You may see some warnings, but these can be safely ignored.



8. Now, you will Publish the Service Fabric app to the local cluster.
9. In Solution Explorer, right-click the Service Fabric Application, ContosoEventsApp, and select Publish.

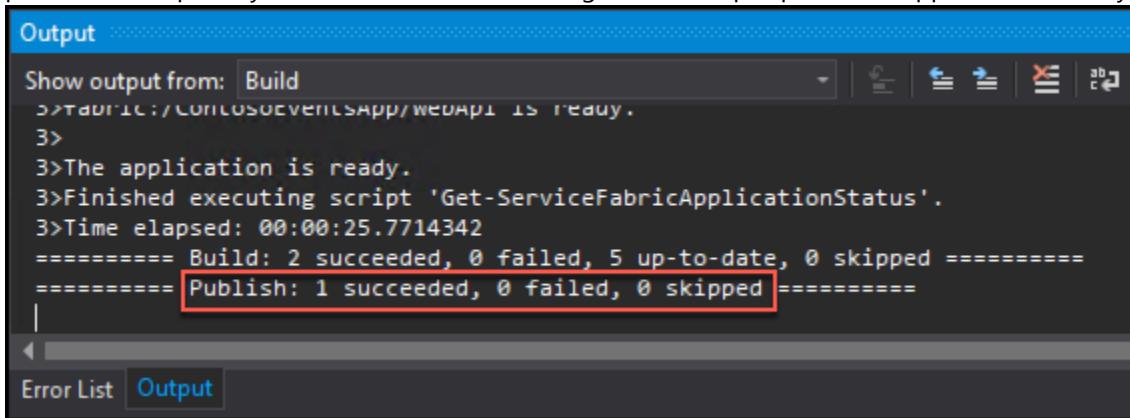


10. In the Public Service Fabric Application dialog, select PublishProfiles\Local.1Node.xml for the Target profile, ensure the correct account is selected, and select Publish.



Note: If you have an error such as: "The project does not have a package action set." you can restart Visual Studio, re-open the solution, and this will resolve the problem.

11. You can see the publish status in the Visual Studio output pane, at the bottom of the window. When the publish process is complete, you will see a success message in the output pane. The application is ready to be used.



Task 2: Test the application

The Service Fabric Application includes a front-end Web API as the public-facing window to internal stateful services. In this task, you will check that you can call the Web API now that you have the application published to the local cluster. Because the app is not yet fully configured, not all Web API methods will be fully functional.

1. On the Lab VM, open a Chrome browser, navigate to the Swagger endpoint for the Web API at: <http://localhost:8082/swagger/ui/index>. Note the three API endpoints: Admin, Events, and Orders.

The screenshot shows the Swagger UI interface for the **ContosoEvents.WebApi**. At the top, there's a navigation bar with the Swagger logo, the URL <http://localhost:8082/swagger/docs/v1>, an **api_key** input field, and a **Explore** button. Below the header, the title **ContosoEvents.WebApi** is displayed. The main content area lists three categories: **Admin**, **Events**, and **Orders**. Each category has a "Show/Hide" link, a "List Operations" link, and an "Expand Operations" link. The **Orders** category is highlighted with a red border. At the bottom left, there's a note: [BASE URL: , API VERSION: V1].

2. Select the Admin API, and observe the list of methods available.

The screenshot shows the Swagger UI interface for the **Admin** API endpoint. At the top, it shows the title **ContosoEvents.WebApi** and the URL <http://localhost:8082/swagger/docs/v1>. The **Admin** category is selected and highlighted with a red border. Below the title, there's a table listing various API operations:

Method	URL
GET	/api/admin/partitions
GET	/api/admin/applicationhealth
GET	/api/admin/servicehealth/{servicename}
GET	/api/admin/actorhealth/{actorname}
POST	/api/admin/simulate/orders
PUT	/api/admin/health/service
DELETE	/api/admin/events
DELETE	/api/admin/orders
DELETE	/api/admin/logmessages

Each row in the table contains a method name (e.g., GET, POST) and its corresponding URL path. The rows alternate colors (light blue, light green, pink). Below the table, there are sections for **Events** and **Orders**, each with a "Show/Hide" link, a "List Operations" link, and an "Expand Operations" link. At the bottom left, there's a note: [BASE URL: , API VERSION: V1].

3. Select /api/admin/partitions, then select Try it out.

Admin

GET **/api/admin/partitions**

Show/Hide | List Operations | Expand Operations

Response Class (Status 200)

Ticket Order Partitions

Model | Model Schema

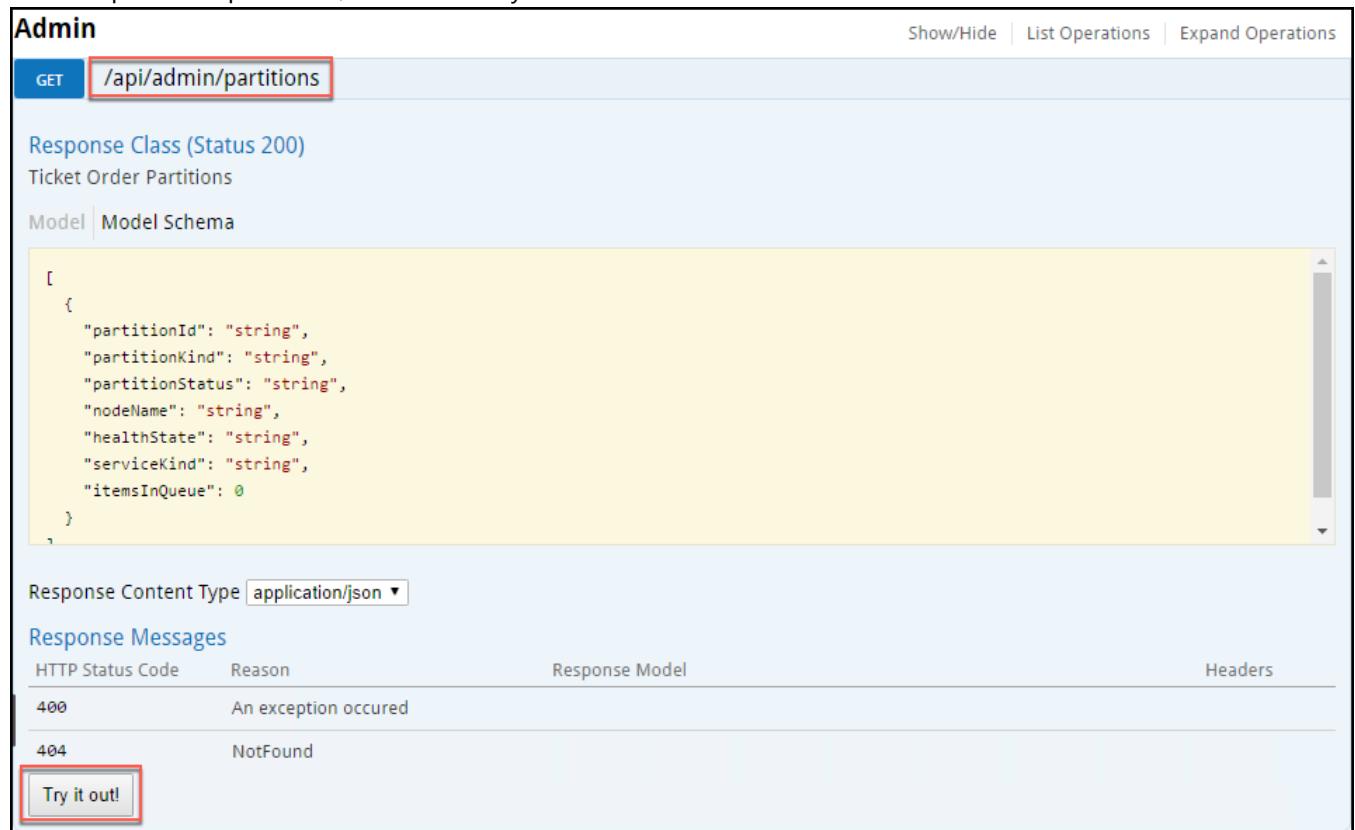
```
[  
  {  
    "partitionId": "string",  
    "partitionKind": "string",  
    "partitionStatus": "string",  
    "nodeName": "string",  
    "healthState": "string",  
    "serviceKind": "string",  
    "itemsInQueue": 0  
  }  
,
```

Response Content Type **application/json ▾**

Response Messages

HTTP Status Code	Reason	Response Model	Headers
400	An exception occurred		
404	NotFound		

Try it out!



4. Make sure it returns a Response Code of 200, as shown in the following screen shot. This API endpoint returns the number of tickets queued across all partitions.

The screenshot shows a Swagger UI interface with the following sections:

- Curl**: A code block containing the command: `curl -X GET --header 'Accept: application/json' 'http://localhost:8082/api/admin/partitions'`.
- Request URL**: A text input field containing `http://localhost:8082/api/admin/partitions`.
- Response Body**: A JSON array representing partition data:

```
[  
  {  
    "partitionId": "88f92ad4-6cd6-41f7-bb9b-0bb755048712",  
    "partitionKind": "Int64Range",  
    "partitionStatus": "Ready",  
    "nodeName": "_Node_0",  
    "healthState": "Ok",  
    "serviceKind": "Stateful",  
    "itemsInQueue": 0  
  }  
]
```
- Response Code**: A text input field containing `200`, which is highlighted with a red box.
- Response Headers**: A JSON object representing the response headers:

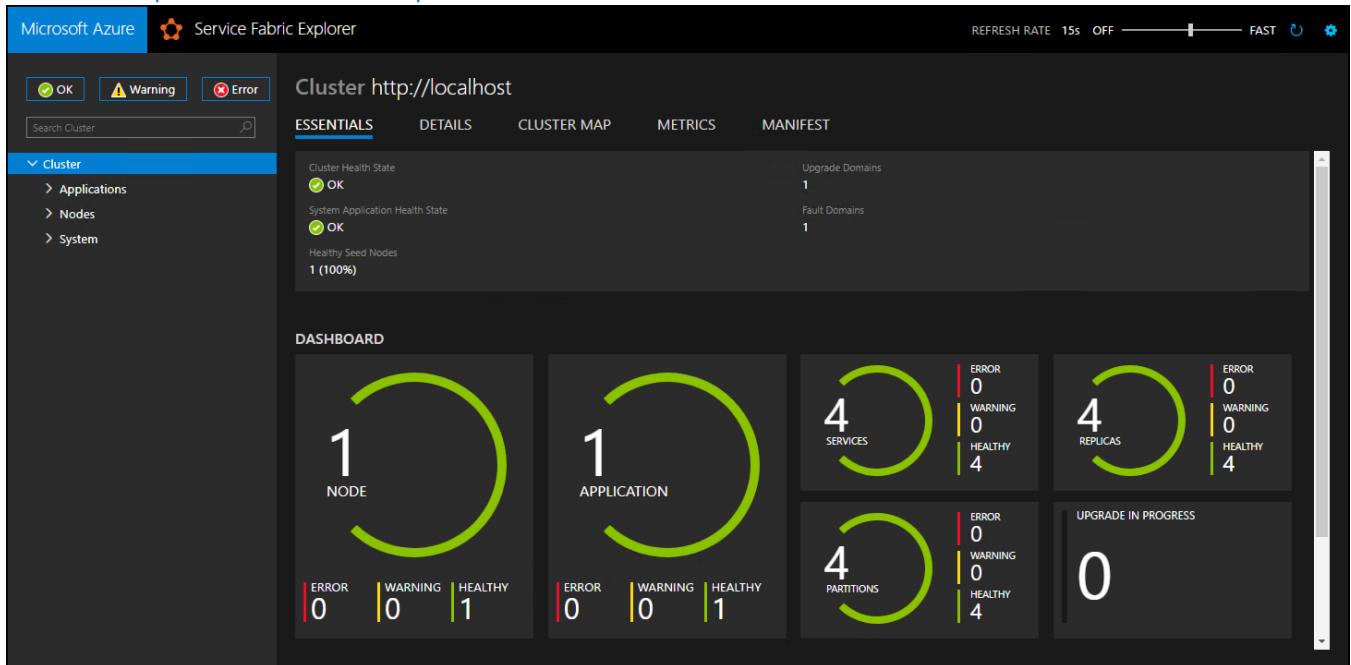
```
{  
  "date": "Thu, 21 Dec 2017 14:53:32 GMT",  
  "server": "Microsoft-HTTPAPI/2.0",  
  "content-length": "193",  
  "content-type": "application/json; charset=utf-8"  
}
```

5. After viewing the Swagger definition, and receiving a success response code from the partitions method, your environment is in a good state to continue.

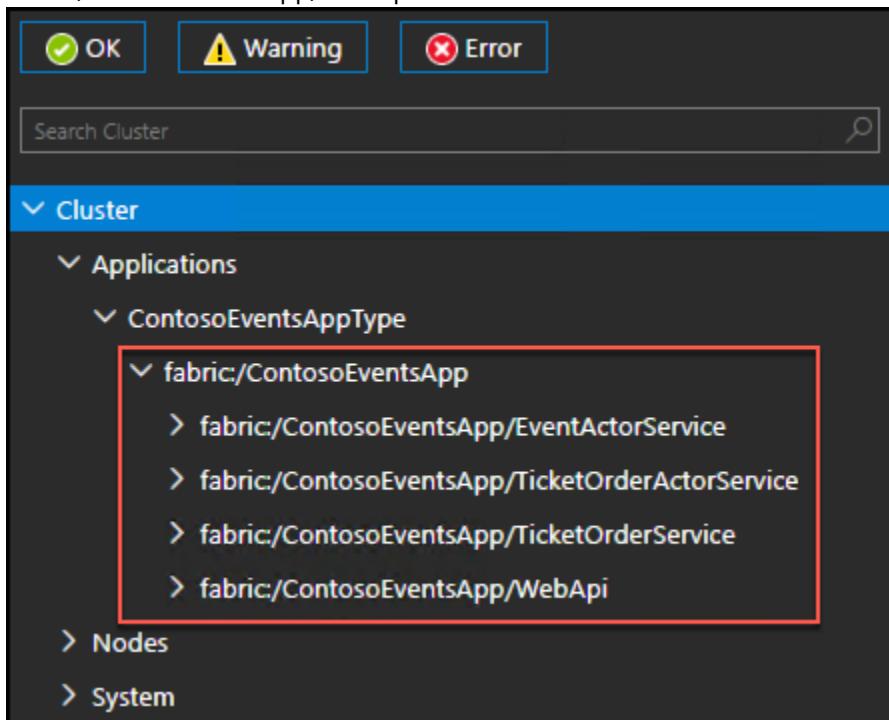
Task 3: Service Fabric Explorer

In this task, you will browse to the Service Fabric Explorer, and view the local cluster.

1. On your Lab VM, open a new tab in your Chrome browser, and navigate to the Service Fabric Explorer for the local cluster at <http://localhost:19080/Explorer/index.html>.



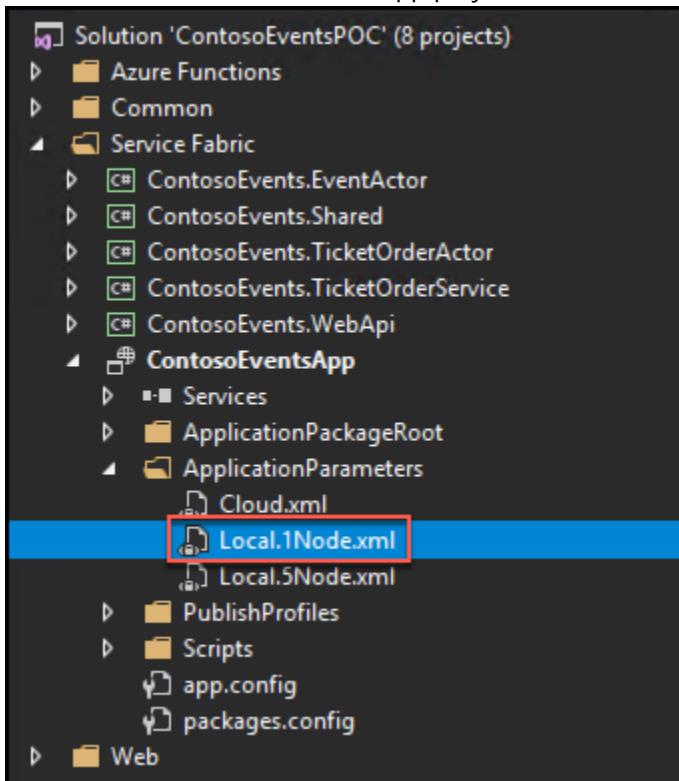
2. Observe that the ContosoEventsApp is deployed with the following services:
 - a. fabric:/ContosoEventsApp/EventActorService
 - b. fabric:/ContosoEventsApp/TicketOrderActorService
 - c. fabric:/ContosoEventsApp/TicketOrderService
 - d. fabric:/ContosoEventsApp/WebApi



Task 4: Set up the Ticket Order Sync queue

In this task, you will complete features of the Contoso Events POC so that placing an order also syncs with the back-end data store. You will also update the configuration settings to reference the Azure resources you previously created correctly.

1. Return to Visual Studio, and from Solution Explorer, open Local.1Node.xml, located in the ApplicationParameters folder under the ContosoEventsApp project in the Service Fabric folder.



2. Locate the following Parameter block in the file. In the steps below, you will retrieve the values needed to update Local.1Node.xml with your own configuration parameters.

```
<Parameter Name="DataStorageEndpointUri" Value="" />  
<Parameter Name="DataStoragePrimaryKey" Value="" />  
<Parameter Name="DataStorageDatabaseName" Value="TicketManager" />  
<Parameter Name="DataStorageEventsCollectionName" Value="Events" />  
<Parameter Name="DataStorageOrdersCollectionName" Value="Orders" />  
<Parameter Name="StorageConnectionString" Value="" />  
<Parameter Name="ExternalizationQueueName" Value="contosoevents-externalization-requests"/>  
<Parameter Name="SimulationQueueName" Value="contosoevents-simulation-requests" />
```

Cosmos DB settings:

1. In the Azure Portal, browse to the Azure Cosmos DB account you created in [Exercise 1, Task 7](#), and select Keys, under Settings in the left-hand menu.

The screenshot shows the 'contosoeventsdb-kb - Keys' blade in the Azure portal. The left sidebar has a 'Keys' item selected, indicated by a red box. The main area shows the 'Read-write Keys' tab selected. The 'URI' field contains the value 'https://contosoeventsdb-kb.documents.azure.com:443/' and the 'PRIMARY KEY' field contains a long string of characters. Both of these fields are highlighted with red boxes.

2. Select the Copy button next to the URI value, return to the Local.1Node.xml file in Visual Studio, and paste the URI value into the value for the DataStorageEndpointUri parameter.
3. Now, on your Cosmos DB keys blade, copy the Primary Key value, return to the Local.1Node.xml file in Visual Studio, and paste the Primary Key value into the value for the DataStoragePrimaryKey parameter.
4. Back in the Azure portal, select Browse under Collections in the left-hand menu for your Cosmos DB. The database (TicketManager) and collection (Orders and Events) names are pre-set in the Local.1Node.xml file, so verify these match the collection and database names you see in the Azure portal.

The screenshot shows the 'contosoeventsdb-kb - Browse' blade in the Azure portal. The left sidebar has a 'Browse' item selected, indicated by a red box. The main area shows a table with two rows: 'Orders' and 'Events'. Both rows are highlighted with red boxes. The table columns include Collection ID, Database, Throughput, Storage, and Est. Hourly.

COLLECTION ID	DATABASE	THROUGHPUT...	STORAGE	EST. HOURLY ...
Orders	TicketManager	2500	0 kB	0.20
Events	TicketManager	2500	0 kB	0.20
TOTAL		5000	0 kB	0.4

5. If you used a different database name, modify the value of the DataStorageDatabaseName parameter to reflect the Cosmos DB database name.
6. If you used a different collection names for the Orders or Events collections, modify the values of the DataStorageEventsCollectionName and DataStorageOrdersCollectionName parameters to match your names.
7. Save the Local.1Node.xml file. The Cosmos DB parameters should now resemble the following:

```

10  <Parameter Name="DataStorageEndpointUri" Value="https://contosoeventsdb-kb.documents.azure.com:443/" />
11  <Parameter Name="DataStoragePrimaryKey" Value="V592d2J3TTCZld0aaPZVAJ05rx4bMLPh4SHWHfrFUfxDxSCHpngodqd72vBMFUO3CCBs5lLkPieM4TCSuojlwig==" />
12  <Parameter Name="DataStorageDatabaseName" Value="TicketManager" />
13  <Parameter Name="DataStorageEventsCollectionName" Value="Events" />
14  <Parameter Name="DataStorageOrdersCollectionName" Value="Orders" />

```

Storage settings:

1. In the Azure Portal, browse to the Storage account you created in [Exercise 1, Task 6](#), and select Access keys under Settings in the left-hand menu.
2. Select the copy button to the right of key1 Connection String to copy the value.

The screenshot shows the 'Access keys' page for the storage account 'contosoeventskb'. On the left, a sidebar lists various settings like Configuration, Shared access signature, Firewalls and virtual networks, Metrics (preview), Properties, Locks, and Automation script. The 'Access keys' option is selected and highlighted with a red box. On the right, there's a brief description about using access keys for authentication and a note about regenerating them. Below that, the 'Storage account name' is listed as 'contosoeventskb'. A table titled 'Default keys' shows two entries: 'key1' and 'key2'. The 'key1' row is highlighted with a red box around its 'KEY' and 'CONNECTION STRING' columns. The 'KEY' column contains a long string of characters, and the 'CONNECTION STRING' column starts with 'DefaultEndpointsProtocol=https;AccountName=contosoeventskb;AccountKey=' followed by a long key value.

NAME	KEY	CONNECTION STRING
key1	Ibjzzc1WHAyZR8wWjX2KG...	DefaultEndpointsProtocol=https;AccountName=contosoeventskb;AccountKey=Ibjzzc1WHAyZR8wWjX2KG...
key2	2q6o2mryLO38gi9UFYY0vn...	DefaultEndpointsProtocol=https;AccountName=contosoeventskb;AccountKey=2q6o2mryLO38gi9UFYY0vn...

3. Return to the Local.1Node.xml file in Visual Studio, and paste the key 1 Connection String into the value for the StorageConnectionString parameter, and save Local.1Node.xml.

```
16 | <Parameter Name="StorageConnectionString" Value="DefaultEndpointsProtocol=https;AccountName=contosoeventskb;AccountKey=Ibjzzc1WHAyZR8wWjX2KG...">
```

Test configurations:

1. In Visual Studio, rebuild and publish the application to the local cluster using the steps you followed previously.
2. After the application is fully published, use Chrome to browse to the Swagger endpoint at <http://localhost:8082/swagger/ui/index>.

3. Select the Orders API methods, and select the POST operation for /api/orders:

The screenshot shows the Microsoft Azure API Management interface. In the top navigation bar, the 'Orders' section is highlighted with a red box. Below it, a list of operations is shown, with the 'POST /api/orders' operation highlighted with a green box. The 'Response Class (Status 200)' section is visible, along with the 'Parameters' table and a JSON schema preview.

Parameter	Value	Description	Parameter Type	Data Type
order	(required)		body	Model Model Schema

```
{  
  "id": "string",  
  "orderDate": "2017-12-21T15:56:25.413Z",  
  "fulfillDate": "2017-12-21T15:56:25.413Z",  
  "cancellationDate": "2017-12-21T15:56:25.413Z",  
  "userName": "string",  
  "email": "string",  
  "tag": "string",  
  "eventId": "string",  
  "paymentProcessorTokenId": "string",  
  "paymentProcessorConfirmation": "string",  
  "tickets": 0,  
}  
Click to set as parameter value
```

4. POST an order to the Contoso Events application. Copy the order request JSON below into the order value box, and select Try it out.

```
{  
  "UserName": "johnsmith",  
  "Email": "john.smith@gmail.com",  
  "EventId": "EVENT1-ID-00001",  
  "PaymentProcessorTokenId": "YYYTT6565661652612516125",  
  "Tickets": 3  
}
```

POST /api/orders

Response Class (Status 200)

Response Content Type **application/json ▾**

Parameters

Parameter	Value	Description	Parameter Type	Data Type
order	<pre>{ "UserName": "johnsmith", "Email": "john.smith@gmail.com", "EventId": "EVENT1-ID-00001", "PaymentProcessorTokenId": "YYTT6565661652612516125", "Tickets": 3 }</pre>		body	Model Model Schema <pre>{ "id": "string", "orderDate": "2017-12-21T15:56:25.413Z", "fulfillDate": "2017-12-21T15:56:25.413Z", "cancellationDate": "2017-12-21T15:56:25.413Z", "userName": "string", "email": "string", "tag": "string", "eventId": "string", "paymentProcessorTokenId": "string", "paymentProcessorConfirmation": "string", "tickets": 0 }</pre>

Parameter content type: **application/json ▾**

Response Messages

HTTP Status Code	Reason	Response Model	Headers
400	An exception occurred		
404	NotFound		

Try it out!

5. This should return successfully, with an HTTP 200 response code. The Response Body contains a unique order id that clients could use to track the order.

Try it out! Hide Response

Curl

```
curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/json' -d '{
    "UserName": "johnsmith",
    "Email": "john.smith@gmail.com",
    "EventId": "EVENT1-ID-00001",
    "PaymentProcessorTokenId": "YYTT6565661652612516125",
    "Tickets": 3
}' 'http://localhost:8082/api/orders'
```

Request URL

```
http://localhost:8082/api/orders
```

Response Body

```
"dcf79eeb-c3f4-49c4-982d-5086a49104b9"
```

Response Code

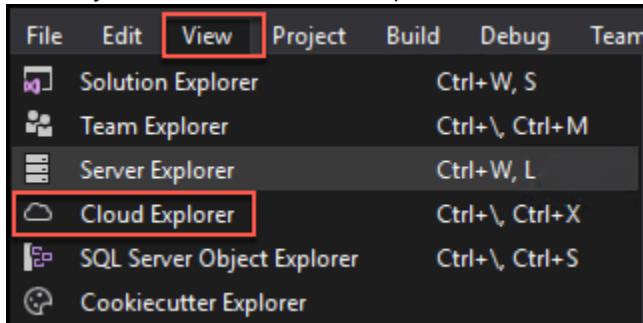
```
200
```

Response Headers

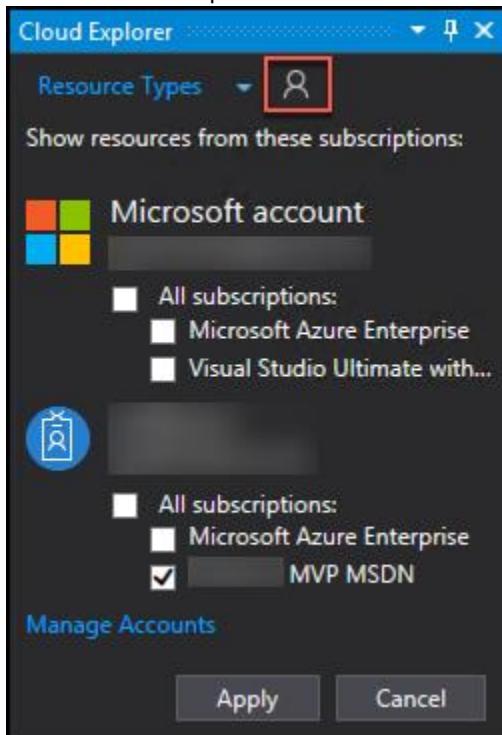
```
{
    "access-control-allow-origin": "*",
    "date": "Thu, 21 Dec 2017 16:03:15 GMT",
    "server": "Microsoft-HTTPAPI/2.0",
    "content-length": "38",
    "content-type": "application/json; charset=utf-8"
}
```

Note: This sends the order to the Web API, which in turn queues the order with the Ticket Order Processing queue. Ultimately, the Ticket Order Actor will pick up the message and process the order.

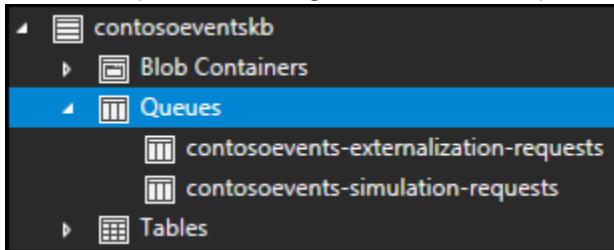
6. Now, you should have an order in the Service Fabric App, and it is being processed. In addition, the Ticket Order Actor should have sent the order to the externalization queue you set up in configuration earlier. The actor has pre-existing logic in place to write to this queue.
7. To verify that the order is in the queue, select View -> Cloud Explorer from the menu in Visual Studio.



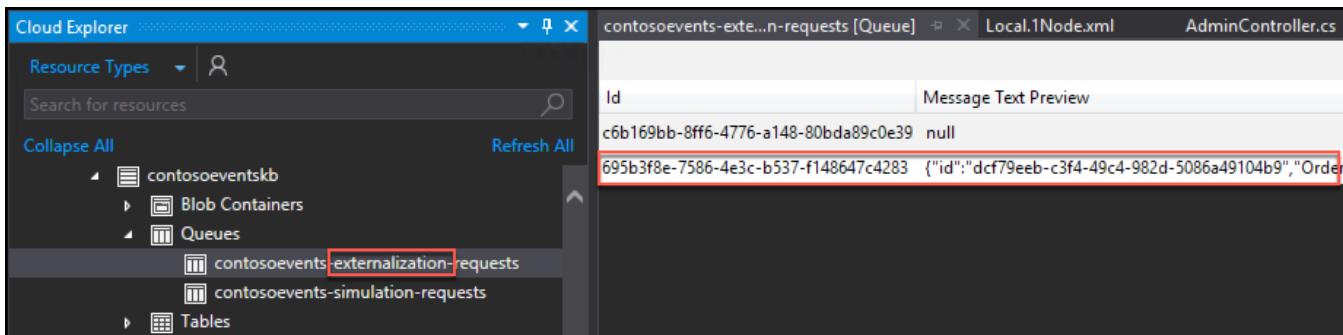
8. In the Cloud Explorer, select the Account Management icon, and select the check box next to the subscription you are using for this lab, then select Apply. You may need to re-enter your account credentials to see the resources under the subscription.



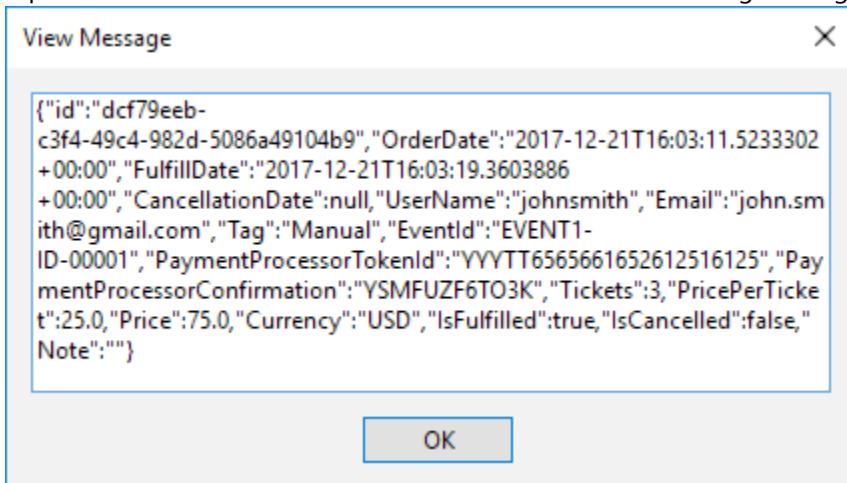
9. Now, expand Storage Accounts under your subscription in Cloud Explorer, and locate the Storage account you set up in [Exercise 1, Task 6](#). You may need to select Load more at the bottom of the list, if you don't see the storage account. Expand the storage account, then expand Queues.



10. Double-click on the externalization queue, and you should see the message you just sent from Swagger in the document window.



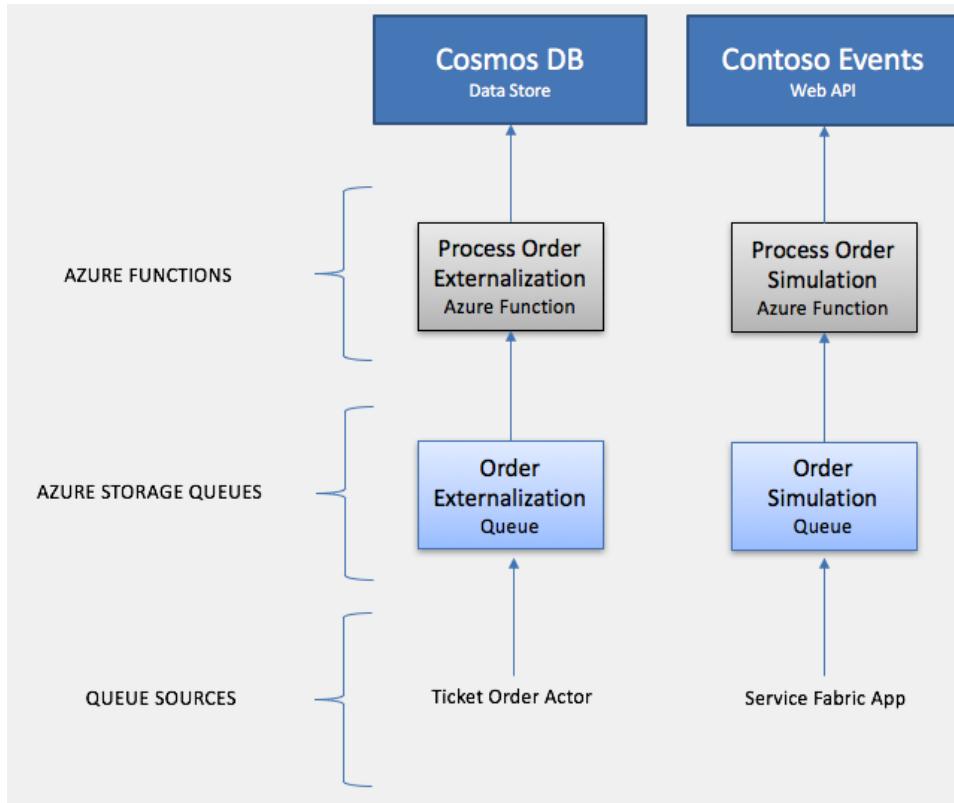
11. Now, double-click on the message, and you will see the contents of the message including the JSON representation of the order. Select OK to close the View Message dialog.



Task 5: Set up the functions

In this task, you will create a function that will be triggered by the externalization queue we created for the app. Each order that is deposited to the queue by the TicketOrderActor type will trigger the ProcessOrderExternalizations function. The function then persists the order to the Orders collection of the Cosmos DB instance.

You will also create a second function that will be used to generate load against the system at runtime. Overall, the ContosoEvents app has the following queue and function architecture. This should help you visualize how the queues and functions are related.



1. There appears to be an issue with Azure Functions detecting Storage accounts, so before creating your function, you will manually add your Storage account connection string to the Application Settings for your Function App.
2. In the Azure portal, browse to the Storage Account you created in [Exercise 1, Step 6](#), then select Access keys under Settings on the left-hand menu, and copy the key1 Connection String value, as you did previously.

NAME	KEY	CONNECTION STRING
key1	Ibjzzc1WHAyZR8wWjX2KG707+w6sDQwYEqkqjS4...	DefaultEndpointsProtocol=https;AccountName=co...
key2	2q6o2mryLO38gi9UFYY0vnLeCv4Uk7iR5/ie3iVUEVu...	DefaultEndpointsProtocol=https;AccountName=co...

The screenshot shows the 'Access keys' section of the Azure Storage Account settings. The 'key1' row is highlighted with a red box around its connection string value. The connection string is: DefaultEndpointsProtocol=https;AccountName=co... . The 'key2' row is also visible below it.

3. Now, browse to the Function App you created in [Exercise 1, Step 5](#).

4. Select your Function App in the left-hand menu, then select Application settings under Configured features.

The screenshot shows the Azure portal's 'Function Apps' blade. In the left sidebar, 'contosoeventsfn-kb' is selected. The main area shows the 'Overview' tab with the function app's status as 'Running'. Below this, the 'Configured features' section is expanded, and the 'Application settings' tab is selected. A red box highlights the 'Application settings' tab.

5. On the Application Settings tab, scroll down and select +Add new setting under Application Settings, then enter contosoeventsstore in the name textbox, and paste the key1 Connection String value you copied from your Storage account into the value textbox.

The screenshot shows the 'Application settings' tab in the Azure portal. It lists several environment variables:

AzureWebJobsDashboard	DefaultEndpointsProtocol=https;AccountName=contosoeventsfnb153;AccountKey=jJH...
AzureWebJobsStorage	DefaultEndpointsProtocol=https;AccountName=contosoeventsfnb153;AccountKey=jJH...
FUNCTIONS_EXTENSION_VERSION	~1
WEBSITE_CONTENTAZUREFILECONNEC...	DefaultEndpointsProtocol=https;AccountName=contosoeventsfnb153;AccountKey=jJH...
WEBSITE_CONTENTSHARE	contosoeventsfn-kbb153
WEBSITE_NODE_DEFAULT_VERSION	6.5.0
contosoeventsstore	DefaultEndpointsProtocol=https;AccountName=contosoeventskb;AccountKey=IbIjzzc1...

A new setting is being added:

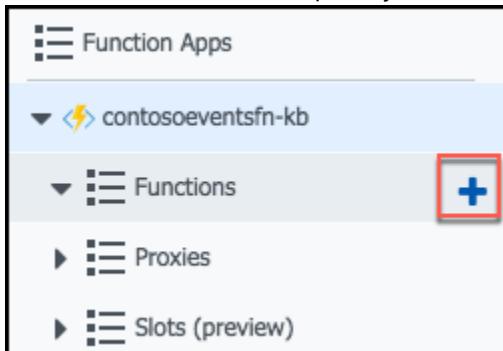
+ Add new setting	
-------------------	--

The 'contosoeventsstore' entry is highlighted with a red box, and the '+ Add new setting' button is also highlighted with a red box.

6. Scroll back to the top of the Application Settings tab, and select Save to apply the change.



7. From the left-hand menu, place your mouse cursor over Functions, then select the + to the right of Functions.



8. Select Data processing under the Get started quickly... header, leave CSharp selected for the language, and select the Create this function button.

Get started quickly with a premade function

1. Choose a scenario

Webhook + API Timer Data processing

2. Choose a language

CSharp JavaScript FSharp Java

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

Create this function

9. First, let's rename the function, so it has a more meaningful name.

10. In the left-hand menu, select your Function app, then select the Platform features tab, and select Console under Development Tools.

The screenshot shows the Azure portal interface for a function app named 'contosoeventsfn-kb'. On the left, there's a sidebar with 'Function Apps' selected. Under 'Function Apps', 'contosoeventsfn-kb' is expanded, and 'Console' is highlighted with a red box. The main content area has tabs: 'Overview' and 'Platform features', with 'Platform features' also highlighted with a red box. The 'Platform features' tab contains sections for General Settings, Networking, API, App Service Plan, Resource Management, and Monitoring. Under 'Development Tools', 'Console' is listed and highlighted with a red box. Other items like 'Advanced tools (Kudu)', 'App Service Editor', 'Resource Explorer', and 'Extensions' are also visible.

11. At the Console command prompt, type "ls" to view the list of functions and files in the wwwroot folder.

```
Manage your web app environment by running common commands. Run 'help' for supported commands.  
require elevated privileges won't work.  
  
D:\home\site\wwwroot  
  
> ls  
D:\home\site\wwwroot  
QueueTriggerCSharp1  
host.json
```

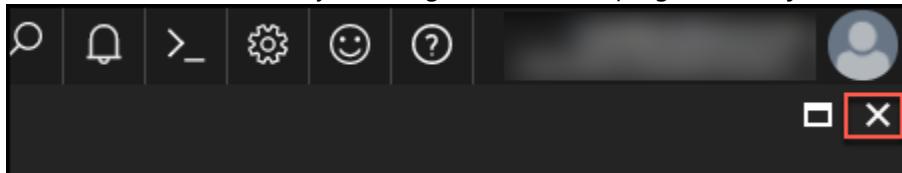
12. Next, paste the following command into the console to rename the QueueTriggerCSharp1 function to ProcessOrderExternalizations.

```
rename QueueTriggerCSharp1 ProcessOrderExternalizations
```

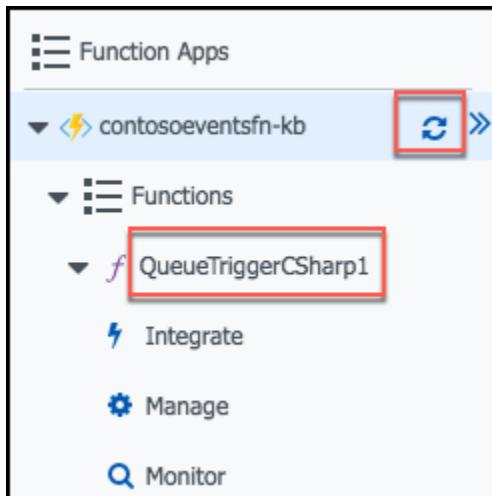
13. Typing "ls" at the command prompt again will show the function has been renamed.

```
> rename QueueTriggerCSharp1 ProcessOrderExternalizations  
D:\home\site\wwwroot  
  
> ls  
D:\home\site\wwwroot  
ProcessOrderExternalizations  
host.json
```

14. Exit the Console window by selecting the X in the top right corner, just below your account name.



15. Back on the Function app page, move your mouse cursor over your function app in the left-hand menu, and select the Refresh icon. Note the function name is still QueueTriggerCSharp1 before refreshing.



16. After the refresh, the function name will change in the display to ProcessOrderExternalizations.

The screenshot shows the Azure Functions blade. At the top, there's a header with a gear icon and the text 'Function Apps'. Below it, a dropdown menu is open, showing 'contosoeventsfn-kb' as the selected item. Under 'Functions', there is a list with one item: 'ProcessOrderExternalizations'. This item is highlighted with a red rectangular border. To the right of the function name are three buttons: 'Integrate', 'Manage', and 'Monitor'. At the bottom of the list is a 'Proxies' section with a right-pointing arrow icon.

17. Under the ProcessOrderExternalizations function, select Integrate.

This screenshot shows the 'ProcessOrderExternalizations' function blade. It has a similar structure to the previous screenshot, with a 'Functions' header and a 'ProcessOrderExternalizations' item under it. However, the 'Integrate' button is now highlighted with a red rectangular border, indicating it is the selected action.

18. On the Integrate screen, set the following:

- Message parameter name: Enter orderItem
- Storage account connection: Select the arrows in the box, then select the contosoevents-store connection from the list. This is the Application Setting you added above.

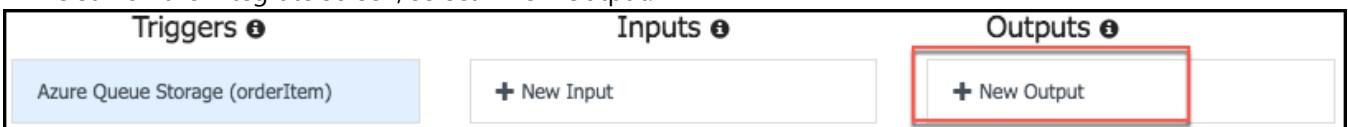
This screenshot shows a dropdown menu for selecting a storage account connection. The options listed are 'AzureWebJobsDashboard', 'AzureWebJobsStorage', 'WEBSITE_CONTENTAZUREFILECONNECTIONSTRING', and 'contosoeventsstore'. The 'contosoeventsstore' option is highlighted with a red rectangular border and has a checkmark next to it, indicating it is selected.

- Queue name: Enter the name of your externalization queue (from Cloud explorer in Visual Studio). This should be contosoevents-externalization-requests.

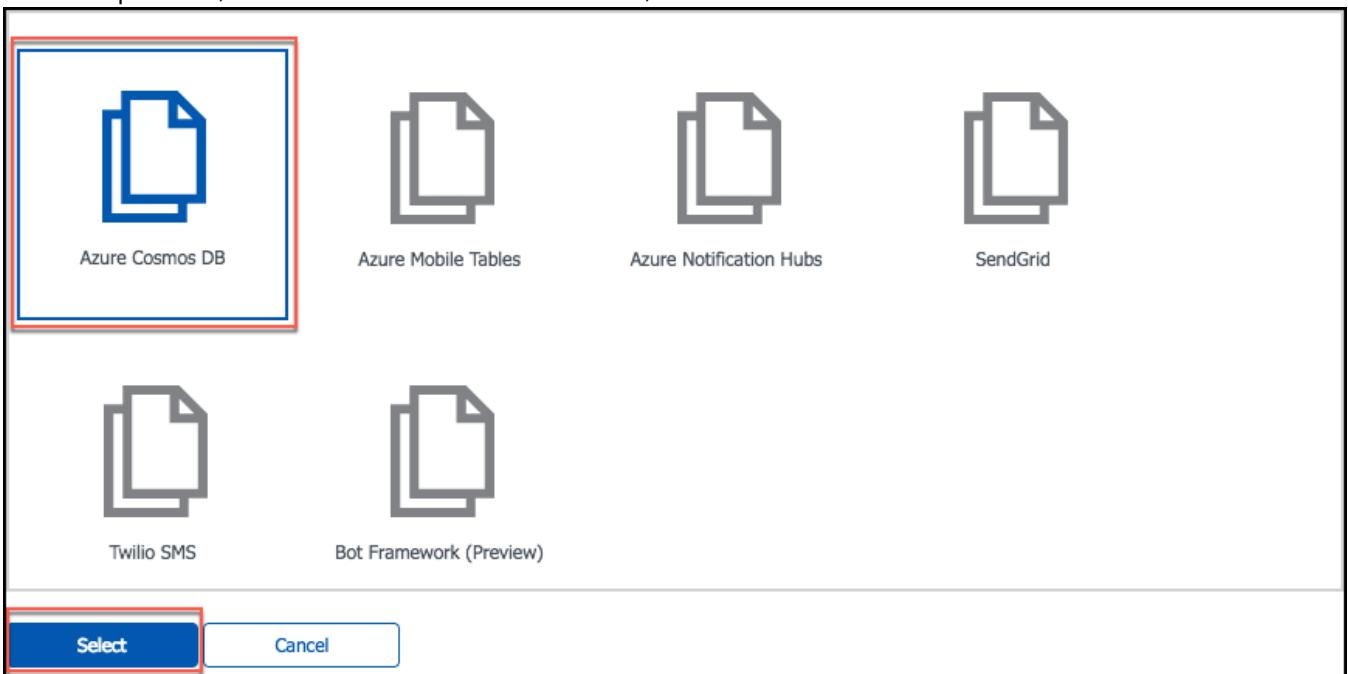
This screenshot shows the 'Azure Queue Storage trigger' configuration dialog. It includes fields for 'Message parameter name' (set to 'orderItem'), 'Queue name' (set to 'contosoevents-externalization-requests'), and 'Storage account connection' (set to 'contosoeventsstore'). There are 'Save' and 'Cancel' buttons at the bottom.

d. Select Save.

19. While still on the Integrate screen, select +New Output.



20. In the outputs box, locate and select Azure Cosmos DB, then select.



21. On the Azure Cosmos DB output screen, enter the following:

- Document parameter name: Enter orderDocument
- Collection name: Enter Orders
- Partition key: Leave empty
- Database name: Enter TicketManager
- Azure Cosmos DB account connection: Select new next to the text box, and select the Cosmos DB you created in [Exercise 1, Task 7](#).

The screenshot shows the 'Azure Cosmos DB output' configuration screen. It includes fields for 'Document parameter name' (set to 'orderDocument'), 'Database name' (set to 'TicketManager'), 'Collection Name' (set to 'Orders'), 'Partition key (optional)' (empty), and 'Azure Cosmos DB account connection' (set to 'contosoeventsdb-kb_DOCUMENTDB'). A red box highlights the 'new' button next to the connection dropdown.

- f. Select Save. You should now see an Azure Queue Storage trigger and an Azure Cosmos DB output on the Integrate screen.

The screenshot shows the 'Integrate' screen in the Azure portal. It has three main sections: 'Triggers' (with 'Azure Queue Storage (orderItem)' selected), 'Inputs' (with '+ New Input'), and 'Outputs' (with 'Azure Cosmos DB (orderDocument)' selected). Both the 'orderItem' trigger and the 'orderDocument' output are highlighted with red boxes.

22. Next, select your function from the left-hand menu.

The screenshot shows the 'Functions' menu in the Azure portal. Under the 'ProcessOrderExternalizations' function, the 'Integrate' option is selected and highlighted with a red box.

23. Now, you will retrieve the code for the function from a file in Visual Studio.

24. In Visual Studio, go to Solution Explorer, and locate ProcessTicketOrderExternalizationEvent.cs in the Azure Functions folder.

The screenshot shows the 'Solution Explorer' in Visual Studio. It lists several projects and files under 'ContosoEventsPOC'. The 'ProcessTicketOrderExternalizationEvent.cs' file is selected and highlighted with a red box.

25. Select all the code in that file (CTRL+A) and copy (CTRL+C) it.

26. Return to your function's page in the Azure portal, and replace the code in the run.csx block with the code you just copied from Visual Studio, and select Save. The run.csx code should now look like the following. Note: The ProcessOrdersExternalization function will enable you to process another order, and see that it is saved to the Orders collection of the Cosmos DB.

The screenshot shows the 'run.csx' code editor in the Azure portal. The code block contains the following C# code:

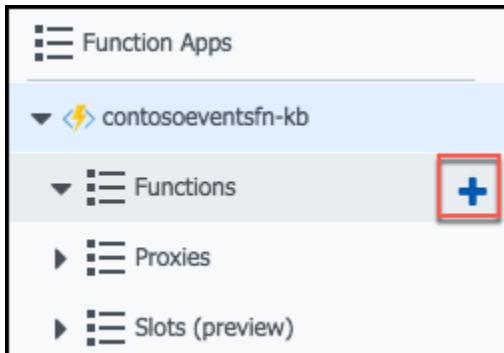
```

1 using System;
2
3
4 public static void Run(object orderItem, out object orderDocument, TraceWriter log)
5 {
6     log.Info($"Ticket Order received: {orderItem}");
7
8     // Store in DocDB so we can query it for orders
9     orderDocument = orderItem;
10
11 }

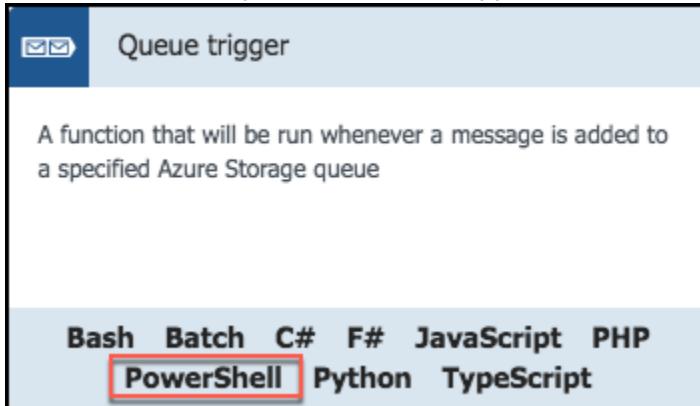
```

27. You will now create another function for the load simulation you will use later in this hands-on lab.

28. As before, select + next to Functions in the left-hand menu.



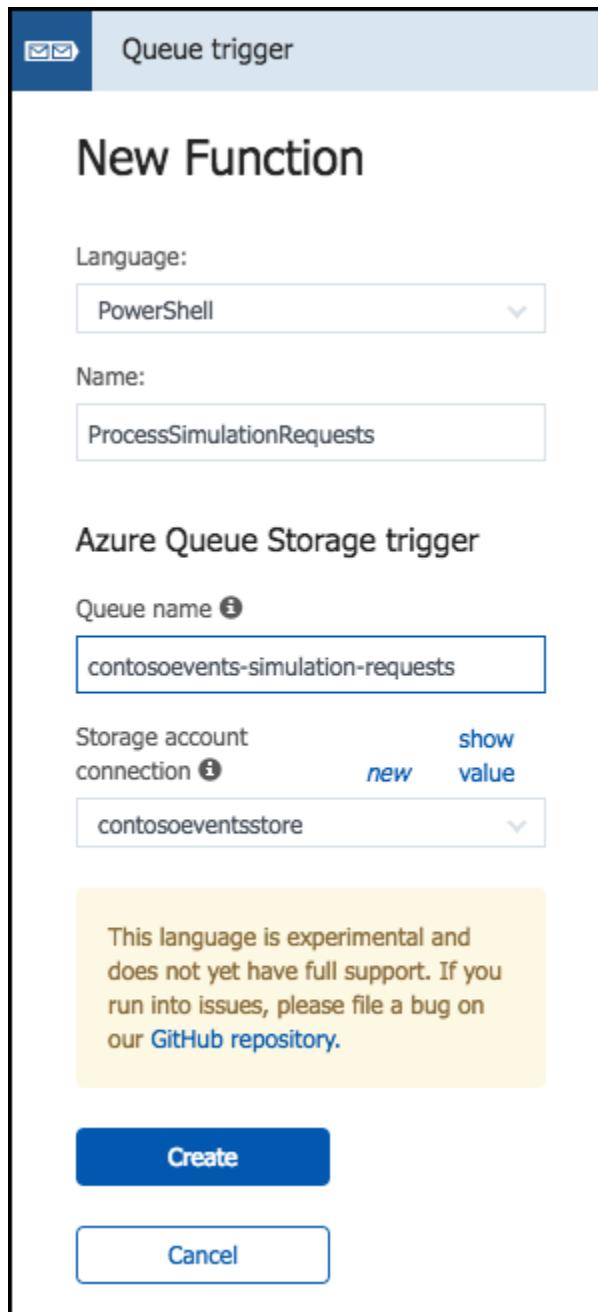
29. In the Choose a template... screen that appears, locate the Queue trigger box, and select PowerShell.



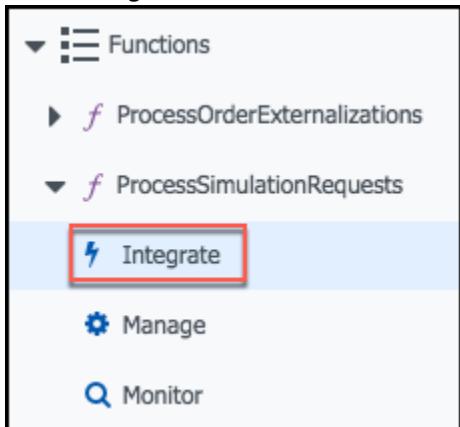
30. In the Queue trigger dialog, enter the following:

- Language: Leave PowerShell selected
- Name: Enter ProcessSimulationRequests
- Queue name: Enter your simulation queue name, from Cloud explorer in Visual Studio. The value should be contosoevents-simulation-requests.
- Storage account connection: Select contosoeventsstore from the drop down

- e. Select Create to create the new function



31. Select Integrate under the new ProcessSimulationRequests function in the left-hand menu.

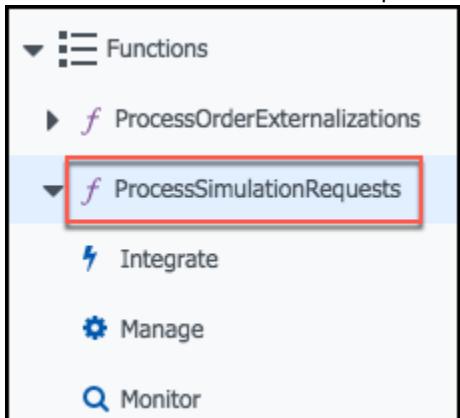


32. Make sure Azure Queue Storage is selected under Triggers, then enter the following:

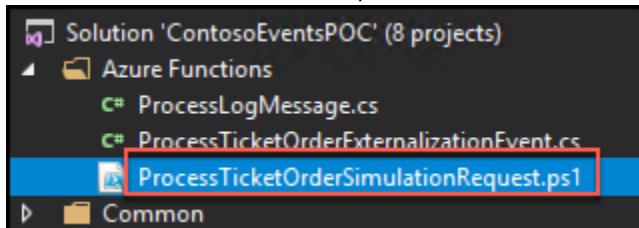
- Message parameter name: Enter simulationRequest
- Storage account connection: Leave set to contosoeventsstore
- Queue name: Leave as contosoevents-simulation-requests
- Select Save.

The screenshot shows the configuration dialog for an Azure Queue Storage trigger. The 'Triggers' tab is selected, showing one trigger: 'Azure Queue Storage (triggerInput)'. The 'Inputs' tab shows a 'New Input' button. The 'Outputs' tab shows a 'New Output' button. Below the triggers, the 'Azure Queue Storage trigger' configuration is shown. It includes fields for 'Message parameter name' (set to 'simulationRequest'), 'Queue name' (set to 'contosoevents-simulation-requests'), and 'Storage account connection' (set to 'contosoeventsstore'). At the bottom are 'Save' and 'Cancel' buttons.

33. Select the ProcessSimulationRequests function in the left-hand menu.



34. Return to Visual Studio, and open the ProcessTicketOrderSimulationRequest.ps1 file in the Azure Functions folder.



35. Copy all the contents of this file (CTRL+A, then CTRL+C), then return to your function page in the Azure portal, and paste the code into the run.ps1 code block.
36. Select Save.
37. The final setting to update is the API Management key. You will return to set this up when you set up the API Management service.

Task 6: Test order data sync

In this task, you will test the ticket order processing back-end, to validate that orders are queued and processed by the ProcessOrderExternalizations function – ultimately saving the order to the Orders collection of the Cosmos DB instance.

1. In the Azure portal, navigate to the Function App.

2. Select the ProcessOrderExternalizations function (may be listed as the default QueueTriggerCSharp1 name if you did not rename it), and select Logs at the bottom of the screen. Leave the window open with the Logs visible.

The screenshot shows the Azure Functions developer tools interface. At the top, there are tabs for "run.csx" (selected), "Save", and "Run". Below the tabs is the code editor containing the following C# code:

```
1 using System;
2
3
4 public static void Run(object orderItem, out object orderDocument, TraceWriter log)
5 {
6     log.Info($"Ticket Order received: {orderItem}");
7
8     // Store in DocDB so we can query it for orders
9     orderDocument = orderItem;
10
11 }
```

At the bottom of the interface, there is a "Logs" section. The word "Logs" is highlighted with a red box. To the right of the logs are several buttons: "Pause", "Clear", "Copy logs", "Expand", and a dropdown menu. The logs themselves show the following message:

```
2017-12-21T20:59:32 Welcome, you are now connected to log-streaming service.
```

3. Repeat the steps to post an order via Swagger to the /api/orders endpoint ([Task 4, Test configurations](#), Steps 2 – 5).

4. As orders are processed, you will see activity in the function logs in the Azure portal.

```

2017-12-21T21:01:30.406 Function started (Id=90218900-d179-46c5-8499-2668a51779d7)
2017-12-21T21:01:30.972 Function started (Id=559af742-8df7-48b9-b4ce-ca27f3fff767)
2017-12-21T21:01:31.675 Ticket Order received:
2017-12-21T21:01:31.675 Ticket Order received: {
    "id": "56dab32a-4154-4ea5-befa-2eb324e142ee",
    "OrderDate": "2017-12-21T21:01:28.388345+00:00",
    "FulfillDate": "2017-12-21T21:01:29.4258911+00:00",
    "CancellationDate": null,
    "UserName": "johnsmith",
    "Email": "john.smith@gmail.com",
    "Tag": "Manual",
    "EventId": "EVENT1-ID-00001",
    "PaymentProcessorTokenId": "YYTT6565661652612516125",
    "PaymentProcessorConfirmation": "8TE0PJ7PSPGE",
    "Tickets": 3,
    "PricePerTicket": 25.0,
    "Price": 75.0,
    "Currency": "USD",
    "IsFulfilled": true,
    "IsCancelled": false,
    "Note": ""
}
2017-12-21T21:01:31.675 Function completed (Success, Id=90218900-d179-46c5-8499-2668a51779d7, Duration=1248ms)
2017-12-21T21:01:31.910 Function completed (Success, Id=559af742-8df7-48b9-b4ce-ca27f3fff767, Duration=940ms)

```

5. Copy the order id of the function just processed. You will use this information below to verify the order was persisted to the Orders collection in Cosmos DB.
 6. If logs are not appearing in this view, click the Monitor tab under the ProcessOrderExternalizations function, then select a log item that has an "id" parameter value passed to the function, as shown below:

Function	Status	Details: Last ran (duration)
ProcessOrderExternalizations (null, {"DatabaseNa ...})	✓	7 minutes ago (0 ms)
ProcessOrderExternalizations ({"id": "eb6ea952-9d ...")	✓	7 minutes ago (94 ms)
ProcessOrderExternalizations ({"id": "56dab32a-41 ...})	✓	7 minutes ago (938 ms)
ProcessOrderExternalizations (null, {"DatabaseNa ...})	✓	7 minutes ago (1,300 ms)

Invocation details

Parameter

```

orderItem: {"id": "eb6ea952-9d26-4d0e-86c8-f538b91d6707", "OrderDate": "2017-12-21T21:01:33.331687+00:00", "FulfillDate": "2017-12-21T21:01:32.1573122+00:00", "CancellationDate": null, "UserName": "johnsmith", "Email": "john.smith@gmail.com", "Tag": "Manual", "EventId": "EVENT1-ID-00001", "PaymentProcessorTokenId": "YYTT6565661652612516125", "PaymentProcessorConfirmation": "8TE0PJ7PSPGE", "Tickets": 3}
orderDocument: {"DatabaseName": "TicketManager", "CollectionName": "Orders", "PartitionKey": "2017-12-21T21:01:32.1573122+00:00", "RowKey": "eb6ea952-9d26-4d0e-86c8-f538b91d6707", "OrderID": "eb6ea952-9d26-4d0e-86c8-f538b91d6707", "OrderDate": "2017-12-21T21:01:32.1573122+00:00", "FulfillDate": "2017-12-21T21:01:33.331687+00:00", "CancellationDate": null, "UserName": "johnsmith", "Email": "john.smith@gmail.com", "Tag": "Manual", "EventId": "EVENT1-ID-00001", "PaymentProcessorTokenId": "YYTT6565661652612516125", "PaymentProcessorConfirmation": "8TE0PJ7PSPGE", "Tickets": 3}
log: 4b7fd8e4-e74c-491c-9662-913dbf74b507
_context: 4b7fd8e4-e74c-491c-9662-913dbf74b507
  
```

Logs

```

Ticket Order received: {
    "id": "eb6ea952-9d26-4d0e-86c8-f538b91d6707",
    "OrderDate": "2017-12-21T21:01:32.1573122+00:00",
    "FulfillDate": "2017-12-21T21:01:33.331687+00:00",
    "CancellationDate": null,
    "UserName": "johnsmith",
    "Email": "john.smith@gmail.com",
    "Tag": "Manual",
    "EventId": "EVENT1-ID-00001",
    "PaymentProcessorTokenId": "YYTT6565661652612516125",
    "PaymentProcessorConfirmation": "8TE0PJ7PSPGE",
    "Tickets": 3
}
  
```

7. In the Azure portal, navigate to your Cosmos DB account, and from the top menu of the Overview blade, select Data Explorer.

The screenshot shows the Azure Cosmos DB Overview blade for the 'contosoeventsdb-kb' account. The left sidebar includes links for Overview, Activity log, Access control (IAM), and Tags. The main area displays account status (Online), resource group (hands-on-labs), subscription information, and a 'Subscription ID' field. The top navigation bar features 'Add Collection', 'Refresh', 'Move', 'Delete Account', and the 'Data Explorer' button, which is highlighted with a red box.

8. In Cosmos DB Data Explorer, select Orders under the TicketManager database, then select New SQL Query from the toolbar, and enter the following query into the Query 1 window, replacing the ID in red with the order id you copied above.

```
SELECT * FROM c WHERE c.id = '56dab32a-4154-4ea5-befa-2eb324e142ee'
```

9. Select Execute Query.

The screenshot shows the Azure Cosmos DB Data Explorer interface. The left sidebar has a 'Data Explorer' link highlighted with a red box. The main area shows the 'TicketManager' database with its collections: 'Orders' (highlighted with a red box) and 'ToDoList'. The 'Orders' collection has sub-links for 'Documents', 'Scale & Settings', 'Stored Procedures', 'User Defined Functions', 'Triggers', and 'Events'. The 'Query 1' window contains the SQL query: 'SELECT * FROM c WHERE c.id = '56dab32a-4154-4ea5-befa-2eb324e142ee''. A red box highlights the 'Execute Query' button. Below the query window, the results are displayed: 'Results: 1 - 1 | Request Charge: 2.32 RUs | →'. The result pane shows a single document with the following JSON structure:

```
{
  "id": "56dab32a-4154-4ea5-befa-2eb324e142ee",
  "OrderDate": "2017-12-21T21:01:28.388345+00:00",
  "FulfillDate": "2017-12-21T21:01:29.4258911+00:00",
  "CancellationDate": null
}
```

10. If the Cosmos DB query returns the order id specified, the order has been fully processed through to the Cosmos DB.

Exercise 3: Publish the Service Fabric Application

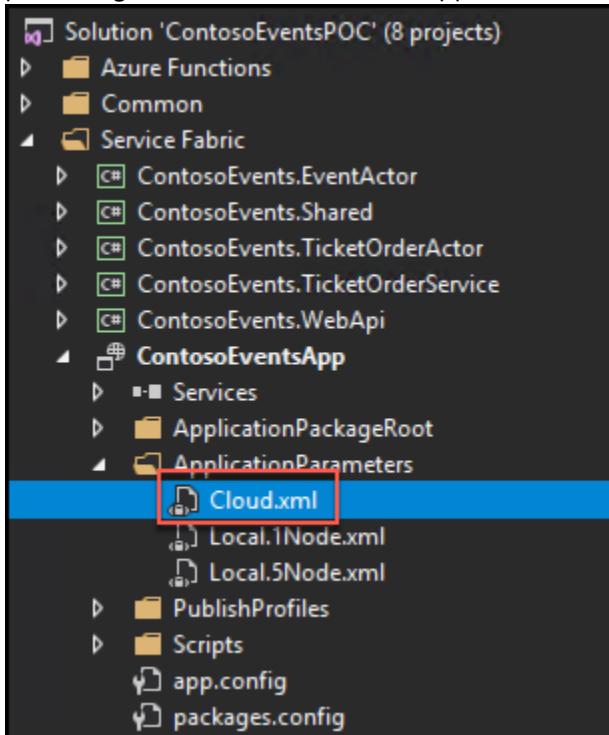
Duration: 15 minutes

In this exercise, you will publish the Service Fabric Application to the Azure cluster you created previously. After it is deployed, you will validate it by sending orders through the Web API endpoints exposed from the cluster.

Task 1: Publish the application

In this task, you will deploy the application to a hosted Service Fabric Cluster.

1. In Visual Studio on your Lab VM, within Solution Explorer, open Cloud.xml from the ApplicationParameters folder of the ContosoEventsApp project, under the Service Fabric folder. This file contains parameters we can use for publishing to the hosted cluster, as opposed to the local cluster.



2. Open Local.1Node.XML from the same folder.
3. To make sure you are using the same parameters you setup earlier for the Local cluster, copy just the following parameters from Local.1Node.xml to Cloud.xml, overwriting the existing parameters in Cloud.xml, then save Cloud.xml.

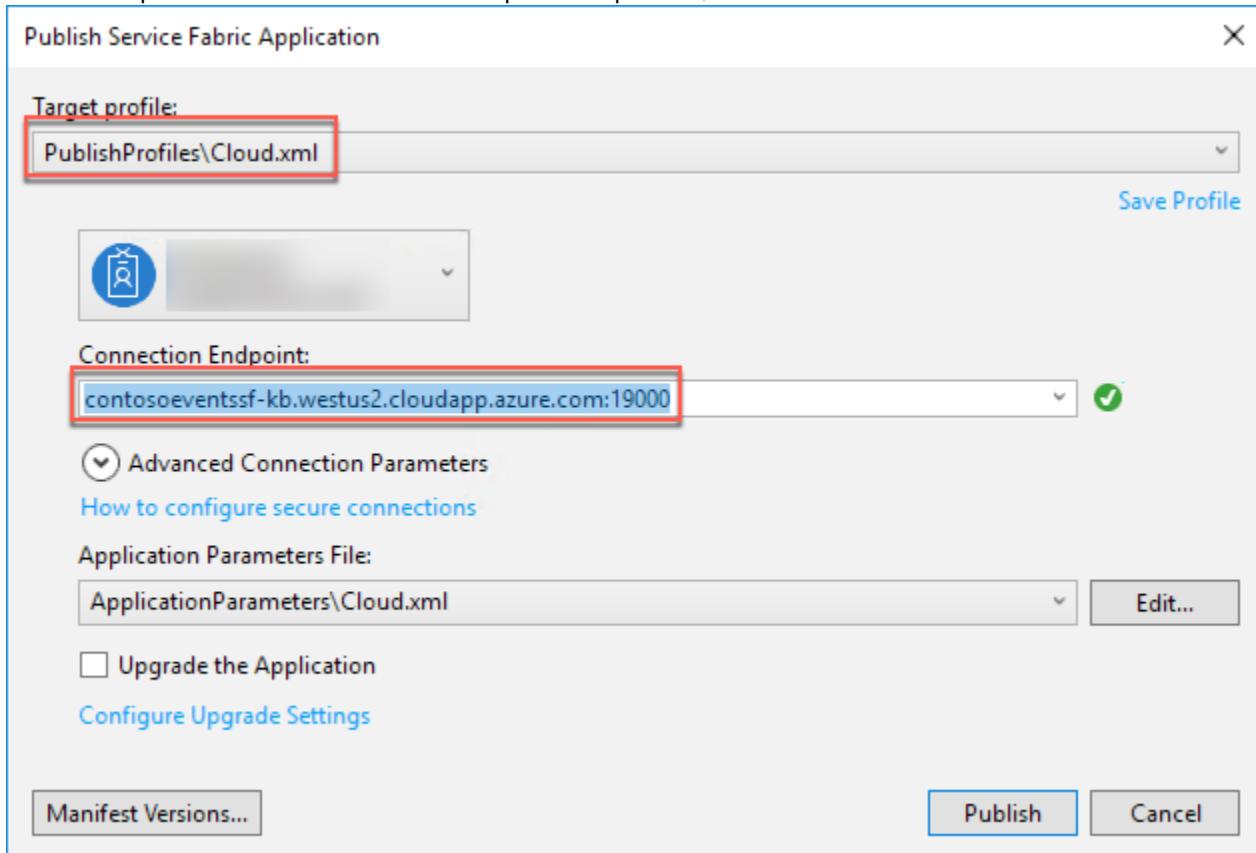
```
<Parameter Name="DataStorageEndpointUri" Value="" />
<Parameter Name="DataStoragePrimaryKey" Value="" />
<Parameter Name="DataStorageDatabaseName" Value="TicketManager" />
<Parameter Name="DataStorageEventsCollectionName" Value="Events" />
<Parameter Name="DataStorageOrdersCollectionName" Value="Orders" />
<Parameter Name="DataStorageLogMessagesCollectionName" Value="LogMessages" />
<Parameter Name="StorageConnectionString" Value="" />
```

```
<Parameter Name="ExternalizationQueueName" Value="contosoevents-externalization-requests"/>  
<Parameter Name="SimulationQueueName" Value="contosoevents-simulation-requests" />
```

4. Review the settings related specifically to cloud publishing.
5. In Cloud.xml, verify the WebAPI_InstanceCount parameter is set to -1. This instructs the cluster to create as many instances of the Web API as there are nodes in the cluster.
6. In Cloud.xml, verify the TicketOrderService_PartitionCount parameter is set to 5. You will look more closely at this in the load test section.

```
<Parameter Name="TicketOrderService_PartitionCount" Value="5" />  
<Parameter Name="TicketOrderService_MinReplicaSetSize" Value="3" />  
<Parameter Name="TicketOrderService_TargetReplicaSetSize" Value="3" />  
<Parameter Name="WebApi_InstanceCount" Value="-1" />  
<Parameter Name="TicketOrderActorService_PartitionCount" Value="5" />  
<Parameter Name="EventActorService_PartitionCount" Value="1" />
```

7. From Solution Explorer, right-click the ContosoEventsApp project and select Publish.
8. In the Publish Service Fabric Application dialog, set the Target profile to Cloud.xml, and select your Service Fabric Cluster endpoint from the Connection Endpoint drop down, then select Publish.



9. Publishing to the hosted Service Fabric Cluster takes about 5 minutes. It follows the same steps as a local publish step with an alternate configuration. The Visual Studio output window keeps you updated of progress.
10. From the Visual Studio output window, validate that the deployment has completed successfully before moving on to the next task.

Task 2: Test an order from the cluster

In this task, you will test an order against your application deployed to the hosted Service Fabric Cluster.

1. In a Chrome browser on your Lab VM, navigate to the Swagger endpoint for the Web API exposed by the hosted Service Fabric cluster. The URL is made of: <http://<your-cluster-name>.<your-region>.cloudapp.azure.com:8082/swagger/ui/index>. For example: <http://contosoeventssf-SUFFIX.eastus.cloudapp.azure.com:8082/swagger/ui/index>
2. Expand the Orders API and expand the POST method of the /api/orders endpoint.

The screenshot shows the Swagger UI interface for the **ContosoEvents.WebApi**. The main navigation bar includes a 'swagger' icon, the URL <http://contosoeventssf-kb.westus2.cloudapp.azure.com:8082/swagger/ui/index>, and an 'api_key' input field. The 'Explore' button is also visible.

The API structure is organized into three main sections: **Admin**, **Events**, and **Orders**. The **Orders** section is currently selected and has a red border around it. It contains several methods:

- GET /api/orders/user/{username}**
- GET /api/orders/event/{eventid}**
- GET /api/orders/{id}**
- GET /api/orders/stats**
- POST /api/orders**

Below the methods, there is a section titled **Response Class (Status 200)** which specifies the **Response Content Type** as `application/json`.

The **Parameters** table lists the required parameters for the **POST /api/orders** method. The **order** parameter is highlighted with a red box. The table columns are: Parameter, Value, Description, Parameter Type, and Data Type. The **order** parameter is defined as a **body** parameter with a **Model Schema**:

```
{  
    "id": "string",  
    "orderDate": "2017-12-21T21:42:33.940Z",  
    "fulfillDate": "2017-12-21T21:42:33.940Z",  
    "cancellationDate": "2017-12-21T21:42:33.940Z",  
    "userName": "string",  
    "email": "string",  
    "tag": "string",  
    "eventId": "string",  
    "paymentProcessorTokenId": "string",  
    "paymentProcessorConfirmation": "string",  
    "tickets": 0.  
}  
Click to set as parameter value
```

3. Copy the JSON below, and paste it into the order field, highlighted in the screen shot above, then select Try it out.

```
{  
  "UserName": "johnsmith",  
  "Email": "john.smith@gmail.com",  
  "Tag": "Manual",  
  "EventId": "EVENT1-ID-00001",  
  "PaymentProcessorTokenId": "YYYTT6565661652612516125",  
  "Tickets": 3  
}
```

The screenshot shows the `/api/orders` endpoint configuration in a Swagger UI. The `order` parameter is highlighted with a red box, indicating it is the field to which the provided JSON should be pasted. The `Try it out!` button at the bottom left is also highlighted with a red box, instructing the user to click it after pasting the JSON.

POST /api/orders

Response Class (Status 200)

Response Content Type application/json ▾

Parameters

Parameter	Value	Description	Parameter Type	Data Type
order	{ "UserName": "johnsmith", "Email": "john.smith@gmail.com", "Tag": "Manual", "EventId": "EVENT1-ID-00001", "PaymentProcessorTokenId": "YYYTT6565661652612516125", "Tickets": 3 }		body	Model Model Schema

Parameter content type: application/json ▾

Model Schema

```
{  
  "id": "string",  
  "orderDate": "2017-12-21T21:42:33.940Z",  
  "fulfillDate": "2017-12-21T21:42:33.940Z",  
  "cancellationDate": "2017-12-21T21:42:33.940Z",  
  "userName": "string",  
  "email": "string",  
  "tag": "string",  
  "eventId": "string",  
  "paymentProcessorTokenId": "string",  
  "paymentProcessorConfirmation": "string",  
  "tickets": 0.  
}
```

Click to set as parameter value

Response Messages

HTTP Status Code	Reason	Response Model	Headers
400	An exception occurred		
404	NotFound		

Try it out!

4. This should return with HTTP 200 response code. The Response Body includes a unique order id that can be used to track the order. Copy the Response Body value. It will be used to verify the order was persisted in Cosmos DB.

The screenshot shows a POST request to `http://contosoeventssf-kb.westus2.cloudapp.azure.com:8082/api/orders`. The response body contains the order ID `"3bb85105-55d4-4d04-b689-8085b0dfe9ee"`. The response code is 200.

```

curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/json' -d '{
  "UserName": "johnsmith",
  "Email": "john.smith@gmail.com",
  "Tag": "Manual",
  "EventId": "EVENT1-ID-00001",
  "PaymentProcessorTokenId": "YYTT6565661652612516125",
  "Tickets": 3
}' 'http://contosoeventssf-kb.westus2.cloudapp.azure.com:8082/api/orders'

```

Request URL: `http://contosoeventssf-kb.westus2.cloudapp.azure.com:8082/api/orders`

Response Body: `"3bb85105-55d4-4d04-b689-8085b0dfe9ee"`

Response Code: `200`

Response Headers:

```

{
  "access-control-allow-origin": "*",
  "date": "Thu, 21 Dec 2017 21:48:15 GMT",
  "server": "Microsoft-HTTPAPI/2.0",
  "content-length": "38",
  "content-type": "application/json; charset=utf-8"
}

```

5. Verify that the order was persisted to the Orders collection.
 6. In the Azure portal, navigate to your Cosmos DB account.
 7. Perform a query against the Orders collection, as you did previously, to verify the order exists in the collection. Replace the id in the query with the order id you copied from the Response Body above.

The screenshot shows a SQL query being executed against the `Orders` collection in the `TicketManager` database. The query is `SELECT * FROM c WHERE c.id = '3bb85105-55d4-4d04-b689-8085b0dfe9ee'`. The results show one document with the specified ID.

COLLECTIONS

- TicketManager**
 - Orders** (highlighted with a red box)
 - Documents
 - Scale & Settings
 - Stored Procedures
 - User Defined Functions
 - Triggers
 - Events
- ToDoList

Query 1

Execute Query

```

1 SELECT * FROM c WHERE c.id = '3bb85105-55d4-4d04-b689-8085b0dfe9ee'

```

Results: 1 - 1 | Request Charge: 2.32 RUs | →

```

{
  "id": "3bb85105-55d4-4d04-b689-8085b0dfe9ee",
  "OrderDate": "2017-12-21T21:48:11.6790634+00:00",
  "FulfillDate": "2017-12-21T21:48:17.7491575+00:00",
  "CancellationDate": null,
}

```

Exercise 4: API Management

Duration: 15 minutes

In this exercise, you will configure the API Management service in the Azure portal.

Task 1: Import API

In this task, you will import the Web API description to your API Management service to create an endpoint.

1. In the Azure portal, navigate to the hands-on-labs resource group, and select your API Management Service from the list of resources.

<input type="checkbox"/>	contosoeventskb	Storage account	West US 2
<input type="checkbox"/>	contosoevents-kb	API Management service	West US 2
<input type="checkbox"/>	contosoeventsplan-kb	App Service plan	West US 2

2. From the API Management blade, select Publisher portal from the toolbar.

The screenshot shows the Azure API Management service blade for the 'contosoevents-kb' service. The 'Publisher portal' button in the top navigation bar is highlighted with a red box. The left sidebar menu has 'Overview' selected. The main content area displays the 'Essentials' section with details about the resource group, status, location, developer portal URL, and gateway URL.

Resource group (change) hands-on-labs	Developer portal URL https://contosoevents-kb.portal.azure-api.net
Status Online	Gateway URL https://contosoevents-kb.azure-api.net
Location West US 2	Tier Developer

3. In the Publisher portal, select APIs from the left-hand menu, then select Import API.

The screenshot shows the 'APIs' page in the Publisher portal. The left sidebar menu has 'API MANAGEMENT' selected. The main content area shows a list of APIs, with the 'Echo API' listed. The 'IMPORT API' button in the center of the page is highlighted with a red box.

4. Return to your Swagger browser window, and copy the URL from the textbox at the top of the screen, next to the Swagger logo, as shown in the screen shot below.

The screenshot shows a Swagger browser window for the 'ContosoEvents.WebApi'. The URL 'http://contosoeventssf-kb.westus2.cloudapp.azure.com:8082/swagger' is highlighted with a red box in the address bar. The interface includes sections for 'Admin' and 'Events'.

5. Return to the Import API page in the Publisher portal, and do the following:
 - a. Select From URL
 - b. Paste the URL copied from Swagger into the Specification document URL textbox
 - c. Select Swagger for the Specification format
 - d. Leave New API selected
 - e. Enter events in the Web API URL suffix textbox
 - f. Note the URL under "This is what the URL is going to look like." You will use this URL in your website configuration in the next exercise.
 - g. Enter Unlimited in the Products textbox.
 - h. Select Save.

Import API

From clipboard Specification document URL Specification format

From file http://contosoevents-kb.westus2.cloudapp.azure.com: WADL
From URL Swagger WSDL

New API Existing API

Web API URL suffix Last part of the API's public URL. This URL will be used by API consumers for sending requests to the web service.

events

Web API URL scheme

HTTP HTTPS

This is what the URL is going to look like:

<https://contosoevents-kb.azure-api.net/events>

Products (optional) Add this API to one or more existing products.

Unlimited

Save **Cancel**

Note: You would typically create a new product for each environment in a scenario like this one. For example, Development, Testing, Acceptance and Production (DTAP) and issue a key for your internal application usage for each environment, managed accordingly.

6. You will see your API listed under APIs.

The screenshot shows the 'contosoevents-kb' API Management service overview. On the left, there's a sidebar with 'API MANAGEMENT' and links for 'Dashboard', 'APIs', 'Products', 'Policies', and 'Analytics'. The main area has tabs for 'Summary', 'Settings', 'Operations', 'Security', 'Issues', and 'Products'. A red box highlights the 'ContosoEvents.WebApi' entry in the 'APIs' list, which includes the URL 'https://contosoevents-kb.azure-api.net/events'. At the bottom, there are date range filters: 'Today', 'Yesterday', 'Last 7 Days', 'Last 30 Days', and 'Last 90 Days'. On the far right, there's an 'EXPORT API' button.

Task 2: Retrieve the user subscription key

In this task, you will retrieve the subscription key for the client applications to call the new API Management endpoint.

1. In the Azure portal, navigate to your API Management service, and from the Overview blade, select Developer portal from the toolbar. This will open a new browser tab, and log you into the Developer portal as an administrator, giving you the rights you need to complete the following steps.

The screenshot shows the 'contosoevents-kb' API Management service overview. The 'Developer portal' button in the top navigation bar is highlighted with a red box. Below it, the 'Essentials' section displays resource group information: 'hands-on-labs', 'Status Online', 'Location West US', and developer portal URLs: 'Developer portal URL https://contosoevents-kb.portal.azure-api.net' and 'Gateway URL https://contosoevents-kb.azure-api.net'. There's also a 'Tier Developer' entry.

2. In the Developer portal, expand the Administrator menu, and then select Profile.

The screenshot shows the 'Contoso Events API' developer portal. The top navigation bar includes 'HOME', 'APIS', 'PRODUCTS', 'APPLICATIONS', 'ISSUES', and 'ADMINISTRATOR'. The 'ADMINISTRATOR' dropdown is highlighted with a red box. The main content area features a blue banner with the text 'Welcome to the developer portal! Administrators can manage the APIs and customize the content in this portal. Learn more'. On the right side, there's a 'MANAGE PROFILE' button, which is also highlighted with a red box.

3. Select Show for the Primary Key of the Unlimited subscription to reveal it.

Your subscriptions		Product
Subscription details		
Subscription name	Starter (default)	Rename Starter
Primary key	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	Show Regenerate
Secondary key	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	Show Regenerate
Subscription name	Unlimited (default)	Rename Unlimited
Primary key	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	Show Regenerate
Secondary key	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	Show Regenerate

4. Save this key for next steps.

Subscription name	Unlimited (default)	Rename
Primary key	499c129b97fa448e94e78f48cbac3858	Hide Regenerate
Secondary key	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	Show Regenerate

5. You now have API Management application key you will need to configure the Function App settings for order load test simulation.

Task 3: Configure the Function App with the API Management key

In this task, you will provide the API Management key in a setting for the Function App, so it can reach the Web API through the API Management service.

1. From the Azure Portal, browse to the Function App.
2. You will create an Application setting for the function to provide it with the API Management consumer key.

3. Select your Function App in the left-hand menu, then select Application settings under Configured features.

The screenshot shows the Azure Functions Overview page for the function app 'contosoeventsfn-kb'. The sidebar on the left lists 'Functions', 'Proxies', and 'Slots (preview)'. The main area displays the app's status as 'Running' and its subscription information ('Solliance MVP MSDN'). Below this, the 'Configured features' section is shown, with 'Application settings' highlighted by a red box.

4. Scroll down to the Application settings section, select +Add new setting, and enter contosoeventsapimgrkey into the name field, and paste the API key you copied from the Developer portal above into the value field.

The screenshot shows the 'Application settings' page for the function app 'contosoeventsfn-kb'. A new application setting named 'contosoeventsapimgrkey' is being added. The 'Name' field contains 'contosoeventsapimgrkey' and the 'Value' field contains '499c129b97fa448e94e78f48cbac3858'. The 'Save' button at the bottom left is highlighted with a red box.

AzureWebJobsDashboard	DefaultEndpointsProtocol=https;AccountName=contosoeventsfnb153;AccountKey=jJH...
AzureWebJobsStorage	DefaultEndpointsProtocol=https;AccountName=contosoeventsfnb153;AccountKey=jJH...
FUNCTIONS_EXTENSION_VERSION	~1
WEBSITE_CONTENTAZUREFILECONNEX...	DefaultEndpointsProtocol=https;AccountName=contosoeventsfnb153;AccountKey=jJH...
WEBSITE_CONTENTSHARE	contosoeventsfn-kbb153
WEBSITE_NODE_DEFAULT_VERSION	6.5.0
contosoeventsstore	DefaultEndpointsProtocol=https;AccountName=contosoeventskb;AccountKey=IbIjzzc1...
contosoeventsdb-kb_DOCUMENTDB	AccountEndpoint=https://contosoeventsdb-kb.documents.azure.com:443/;AccountKey...
contosoeventsapimgrkey	499c129b97fa448e94e78f48cbac3858

5. Scroll back to the top of the Application Settings tab, and select Save to apply the change.

The screenshot shows the bottom navigation bar with two buttons: 'Save' and 'Discard'. The 'Save' button is highlighted with a red box.

6. You will be able to issue a load test from the website in Exercise 8, and see that orders have been processed through the function – because it will have successfully called the API and you will see results in the load test status page.

Exercise 5: Configure and publish the web application

Duration: 15 minutes

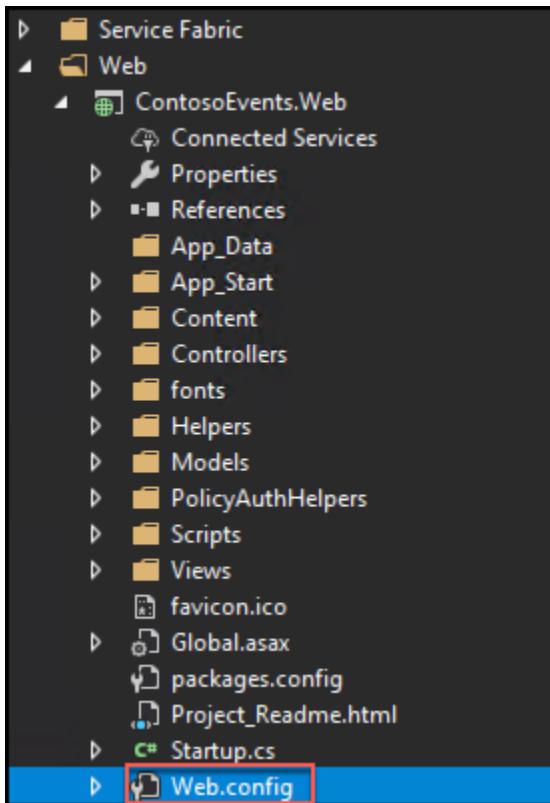
In this exercise, you will configure the website to communicate with the API Management service, deploy the application, and create an order.

Task 1: Configure the Web App settings

In this task, you will update configuration settings to communicate with the API Management service. You will be guided through the instructions to find the information necessary to populate the configuration settings.

1. Within Visual Studio Solution Explorer on your Lab VM, expand the Web folder, then expand the ContosoEvents.Web project, and open Web.config. You will update these appSettings in this file:

```
<add key="apimng:BaseUrl" value="" />  
<add key="apimng:SubscriptionKey" value="" />
```



2. For the apimng:BaseUrl key, enter the base URL of the API you created in the API Management Publisher Portal ([Exercise 5, Task 1, Step 7](#)), such as <https://contosoeventsSUFFIX.azure-api.net/events/>.

Note: Make sure to include a trailing "/" or the exercise will not work.

3. For the apimng:SubscriptionKey key, enter the subscription key you revealed in API Management developer portal ([Exercise 5, Task 2, Step 4](#)).

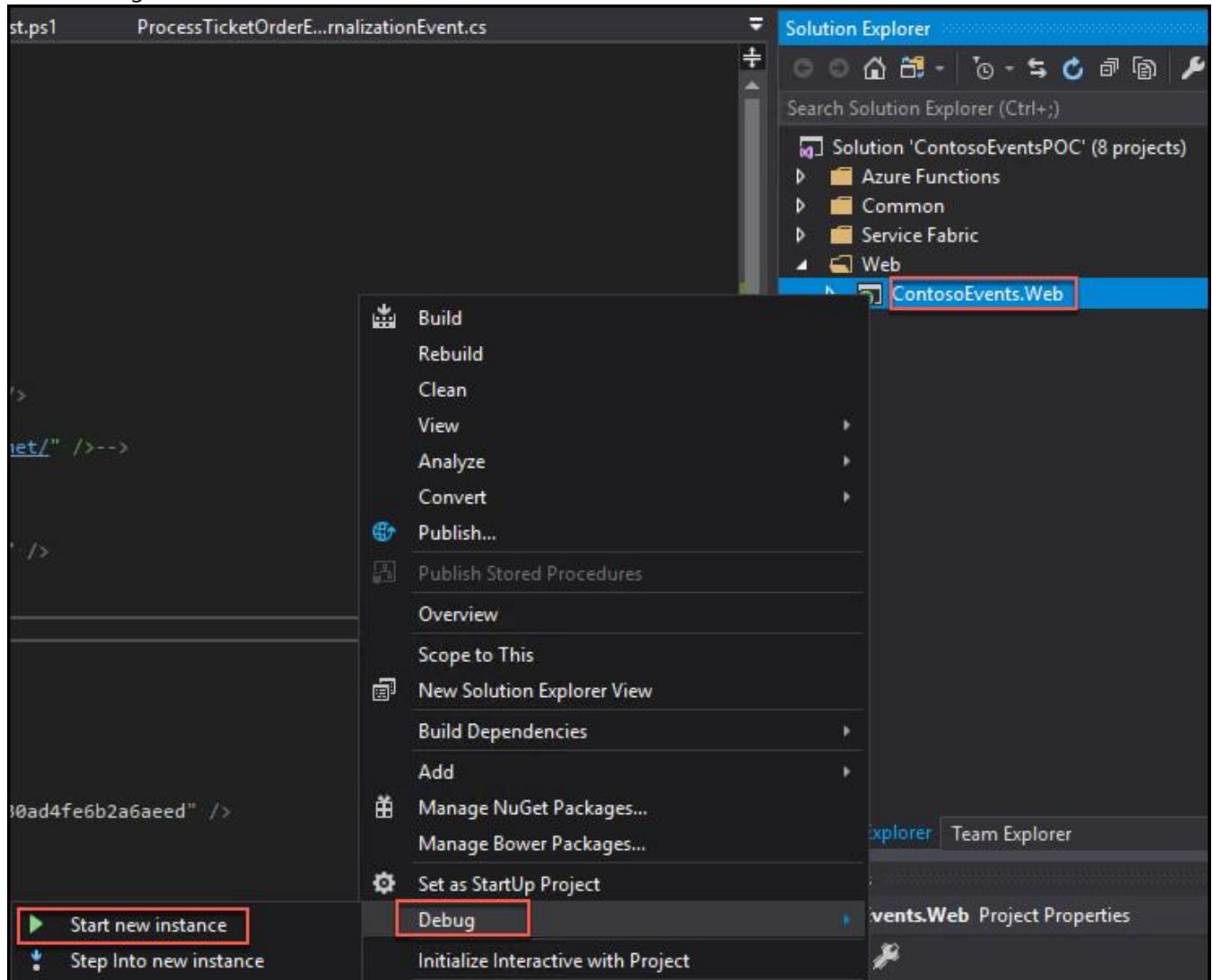
- Save Web.config. You should have values for two of the API Management app settings.

```
20 |     <add key="apimng:BaseUrl" value="https://contosoevents-kb.azure-api.net/events/" />
21 |     <add key="apimng:SubscriptionKey" value="499c129b97fa448e94e78f48cbac3858" />
22 |   </appSettings>
```

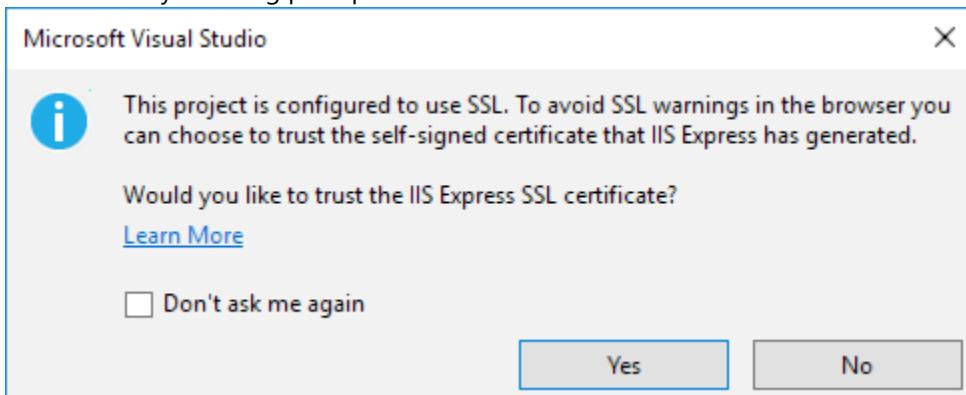
Task 2: Running the web app and creating an order

In this task, you will test the web application calls to API Management by creating an order through the UI.

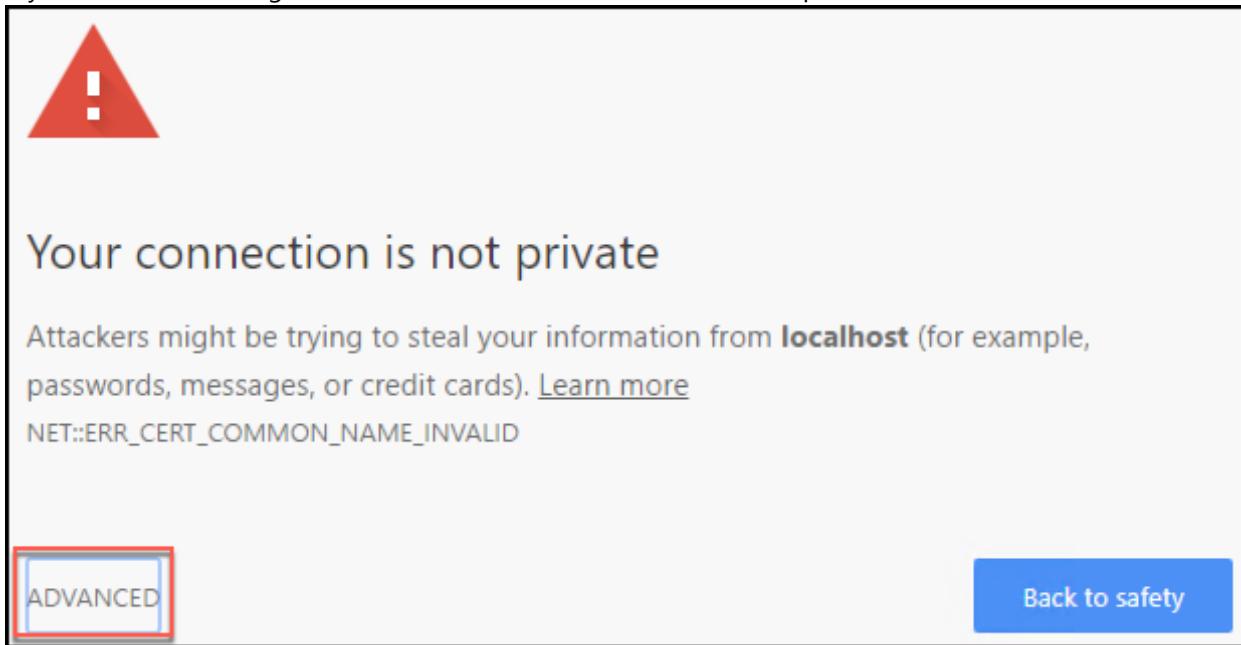
- Using Solution Explorer in Visual Studio, expand the Web folder, then right-click the ContosoEvents.Web project, select Debug, and then Start new instance.



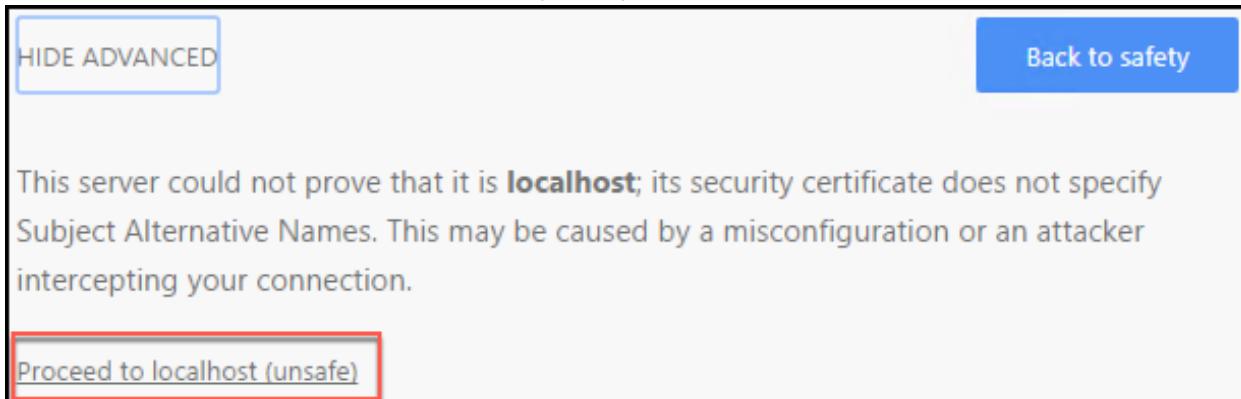
2. If prompted about whether you would like to trust the IIS Express SSL certificate, select Yes, then select Yes again at the Security Warning prompt.



3. If you receive a warning in the browser that "Your connection is not private," select Advanced.



4. Under Advanced, select Proceed to localhost (unsafe).



5. When the application launches you will see the website home page as shown in the following screen shot.

The screenshot shows the homepage of the Contoso Events website. At the top, there is a navigation bar with links for "Contoso Events", "Orders", "LoadTest", and "LoadTest Status". The main header features the text "Contoso Events" and "Welcome to the Contoso Events." Below this, a sub-header says "Browse and book your favorite events in a few clicks." The background of the header and main content area is a photograph of a concert crowd with performers on stage. Below the header, the text "On Sale Now: Seattle Rock and Rollers" is displayed. A thumbnail image of a band performing on stage is shown, with a green button labeled "25 USD" overlaid. To the right of the thumbnail, the event details are listed: "Seattle Rock and Rollers", "Seattle", "22/OCT/2017", and "04:03". At the bottom of the card, there is a blue button labeled "Order tickets now".

6. Note the event presented on the home page has an Order Tickets Now button. Click that to place an order.

7. Choose the number of tickets for the order, and then scroll down to see the billing fields.

Seattle Rock and Rollers



Order

Number of tickets :

Total Price : 25 USD

Location : Seattle
Date : 25/mai/2016
Time : 04:36
PricePerTicket : 25 USD

8. Since you have not yet integrated Azure Active Directory B2C, the application does not know who you are. You will have to enter the empty fields for your email, first name, last name, and Cardholder name.

Billing	Credit Card
Email : <input type="text" value="johnsmith@contoso.com"/>	Cardholder Name : <input type="text" value="John Smith"/>
Phone Number : <input type="text" value="425 123 4567"/>	Card Number : <input type="text" value="1234567890123456"/>
First Name : <input type="text" value="John"/>	Expiration Month : <input type="text" value="abril"/> <input type="button" value="▼"/>
Last Name : <input type="text" value="Smith"/>	Expiration Year : <input type="text" value="2017"/> <input type="button" value="▼"/>
Address : <input type="text" value="1 Microsoft Way"/>	Security Code : <input type="text" value="123"/>
City : <input type="text" value="Redmond"/>	<input type="button" value="Place Order"/>
Postal Code : <input type="text" value="WA 98052"/>	
Country : <input type="text" value="US"/>	

9. Select Place Order.

10. Once the order is queued for processing, you will be redirected to a results page as shown in the following screen shot. It should indicate Success and show you the order id that was queued as confirmation.

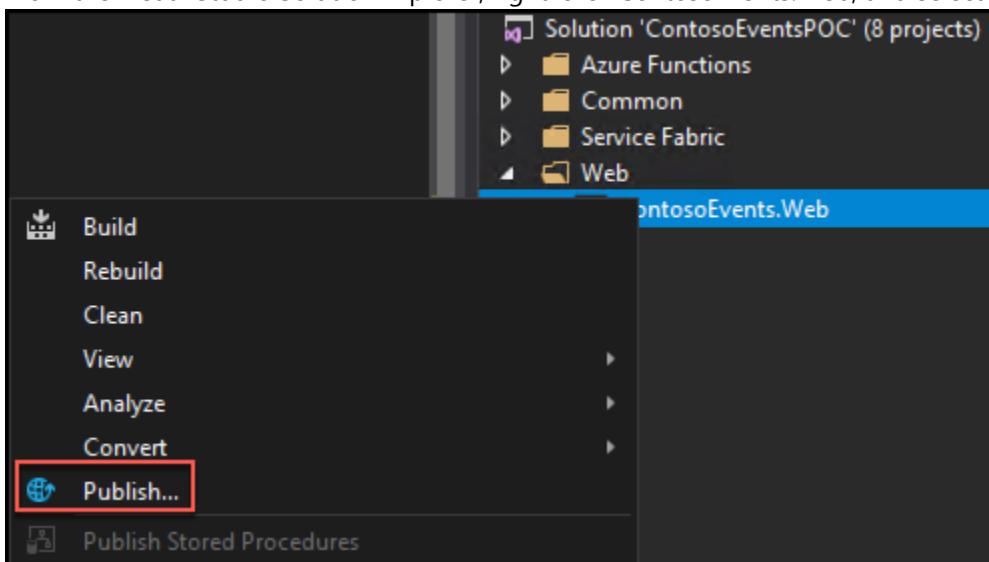


11. Close the web browser to stop debugging the application.

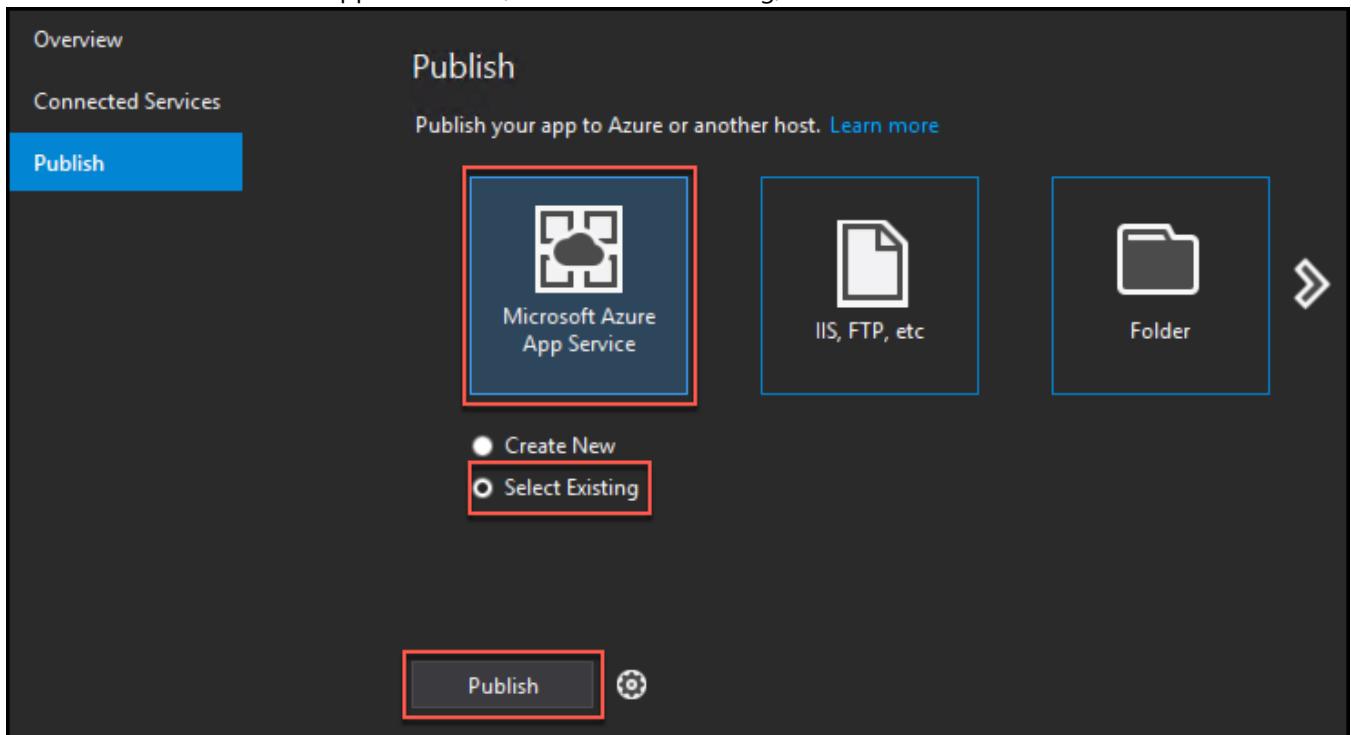
Task 3: Publish the web app

In this task, you will publish the web application to Azure.

1. From the Visual Studio Solution Explorer, right-click ContosoEvents.Web, and select Publish.

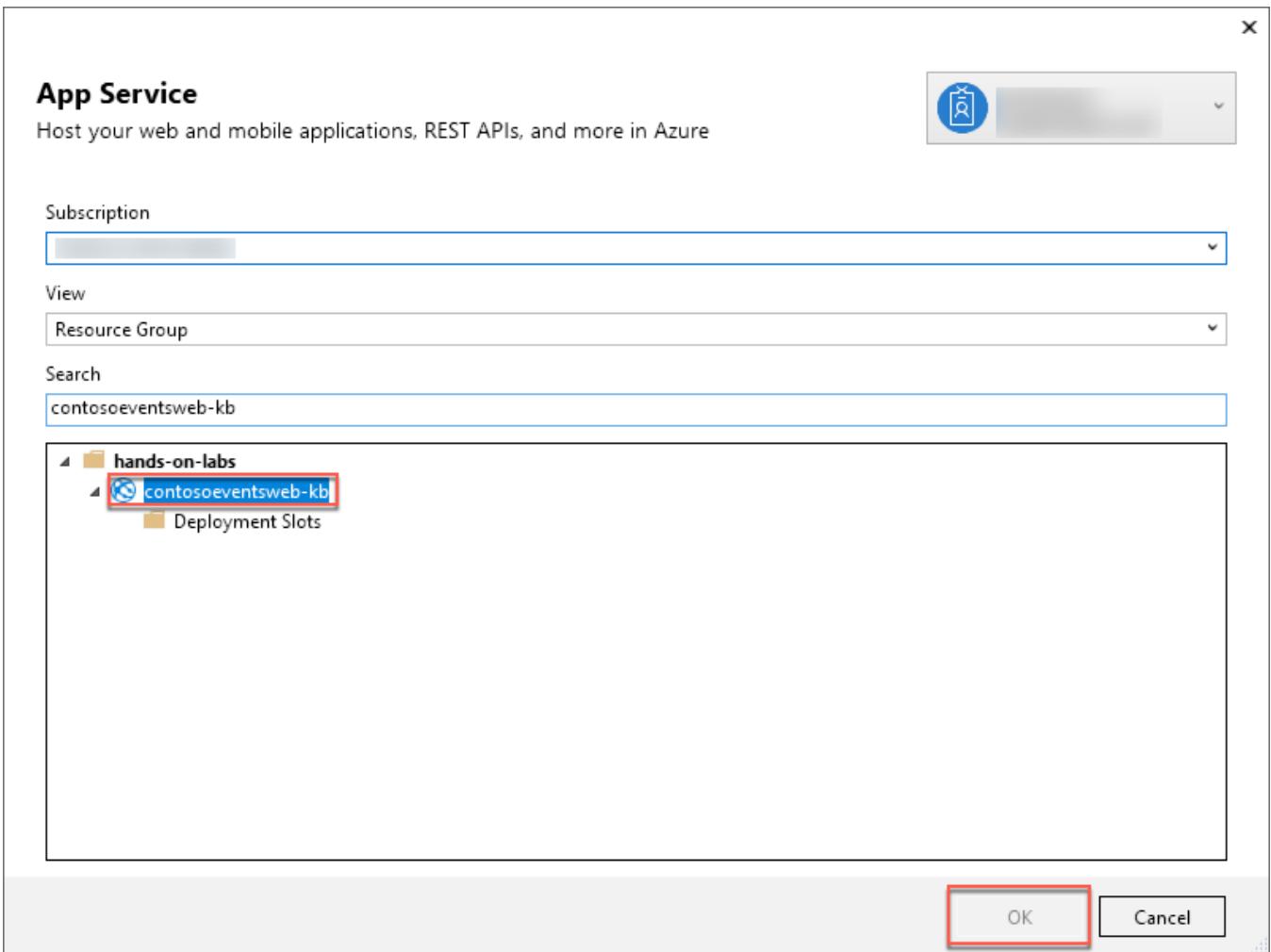


2. Select the Microsoft Azure App Service tile, choose Select Existing, then select Publish.

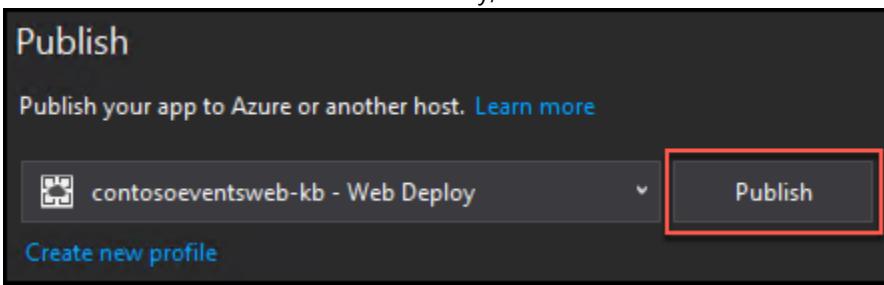


3. You may be prompted to log in to your Microsoft Account with access to the subscription where you created the resources for this hands-on lab. After logging in, you can select the subscription in the App Service screen.

4. From the list below, expand the Resource Group you created previously (hands-on-labs), and select the web app contosoeventsweb-SUFFIX. Select OK.



5. If the Publish does not start automatically, select Publish next to the Web Deploy publishing profile.



6. When publishing is complete, your browser will launch, and navigate to the deployed web app home page. You can optionally submit another order to validate functionality works as in Task 2.

Exercise 6: Upgrading

Duration: 30 minutes

In this task, you will make changes to the code, and deploy an update to the application to enhance functionality. Specifically, the update addresses the area of concern related to changes in the ticket order model, and the impact on the system.

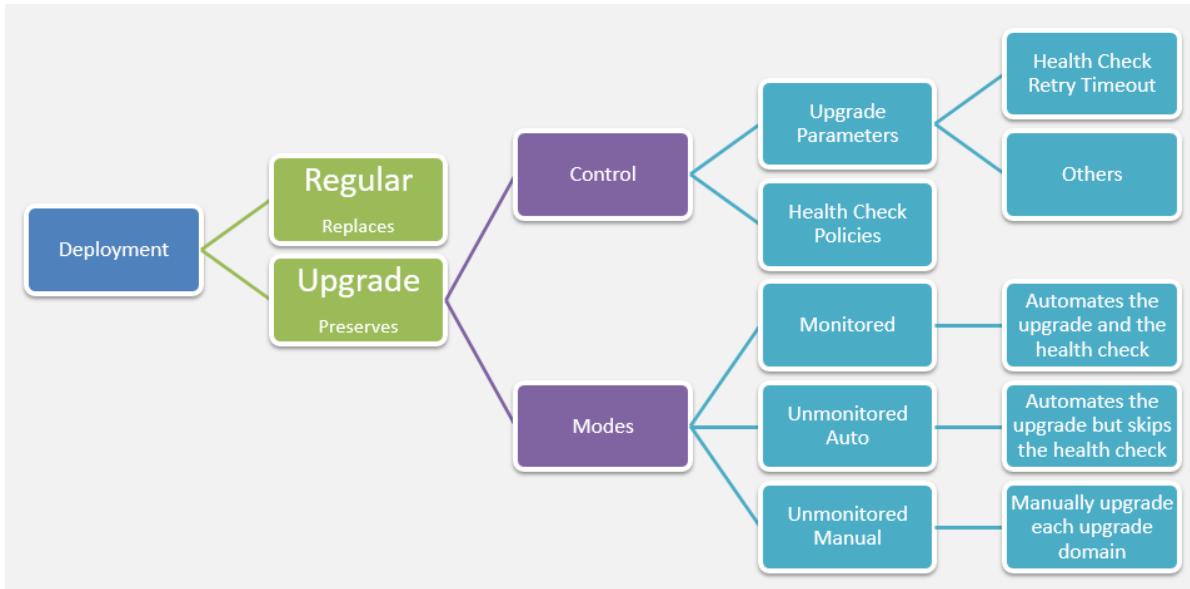
This task will illustrate the mechanism that Service Fabric provides for upgrading an application in production.

Task 1: How upgrade works

In Service Fabric, deployments can be either regular or upgrade. A regular deployment erases any previous deployment data while the upgrade deployment preserves it. There are advantages to upgrades:

- Stateful service data will not be lost during upgrade
- Availability remains during the upgrade

The following figure illustrates the deployment hierarchy:

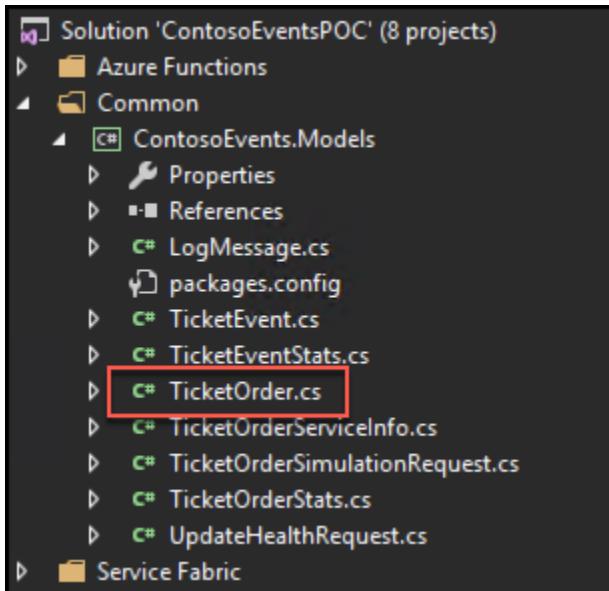


If you set the upgrade mode to Monitored, Service Fabric will be in full control of the upgrade process. There is a configurable time to wait after the upgrade has finished before Service Fabric evaluates the health of the application. This value defaults to 600 seconds.

Task 2: Update an actor state

Currently, the TicketOrderActor does not have a status property to make it easier to check on the actor order status quickly. In this task, you will modify the Ticket Order State model to add a new status property.

1. In Visual Studio on the Lab VM, open TicketOrder.cs in the ContosoEvents.Models project, under the Common folder.



2. Edit the TicketOrder type to include a status field based on an Enum. Uncomment all TODO: Exercise 6 – Task 1 – there are two places as shown below:

```
//TODO: Exercise 6 - Task 1  
[DataMember]  
  
public OrderStatuses Status { get; set; }  
  
and  
  
//TODO: Exercise 6 - Task 1  
  
public enum OrderStatuses  
{  
    Fulfilled,  
    TicketsExhausted,  
    CreditCardDenied,  
    Cancelled,  
    Invalid  
}
```

3. Save TicketOrder.cs.
4. From Solution Explorer, open TicketOrderActor.cs in the ContosoEvents.TicketOrderActor project, under the Service Fabric folder.

5. Edit the TicketOrderActor to add the new order status. Uncomment all TODO: Exercise 6 – Task 1. The change will uncomment several lines that set the new Status field to one of the OrderStatuses enumeration values. Be sure to find all of the following comments (there are 6 total):

```
//TODO: Exercise 6 - Task 1  
state.Status = OrderStatuses.Invalid;  
  
//TODO: Exercise 6 - Task 1  
state.Status = OrderStatuses.Fulfilled;  
  
//TODO: Exercise 6 - Task 1  
state.Status = OrderStatuses.CreditCardDenied;  
  
//TODO: Exercise 6 - Task 1  
state.Status = OrderStatuses.TicketsExhausted;  
  
//TODO: Exercise 6 - Task 1  
state.Status = OrderStatuses.Invalid;  
  
//TODO: Exercise 6 - Task 1  
state.Status = OrderStatuses.Cancelled;
```

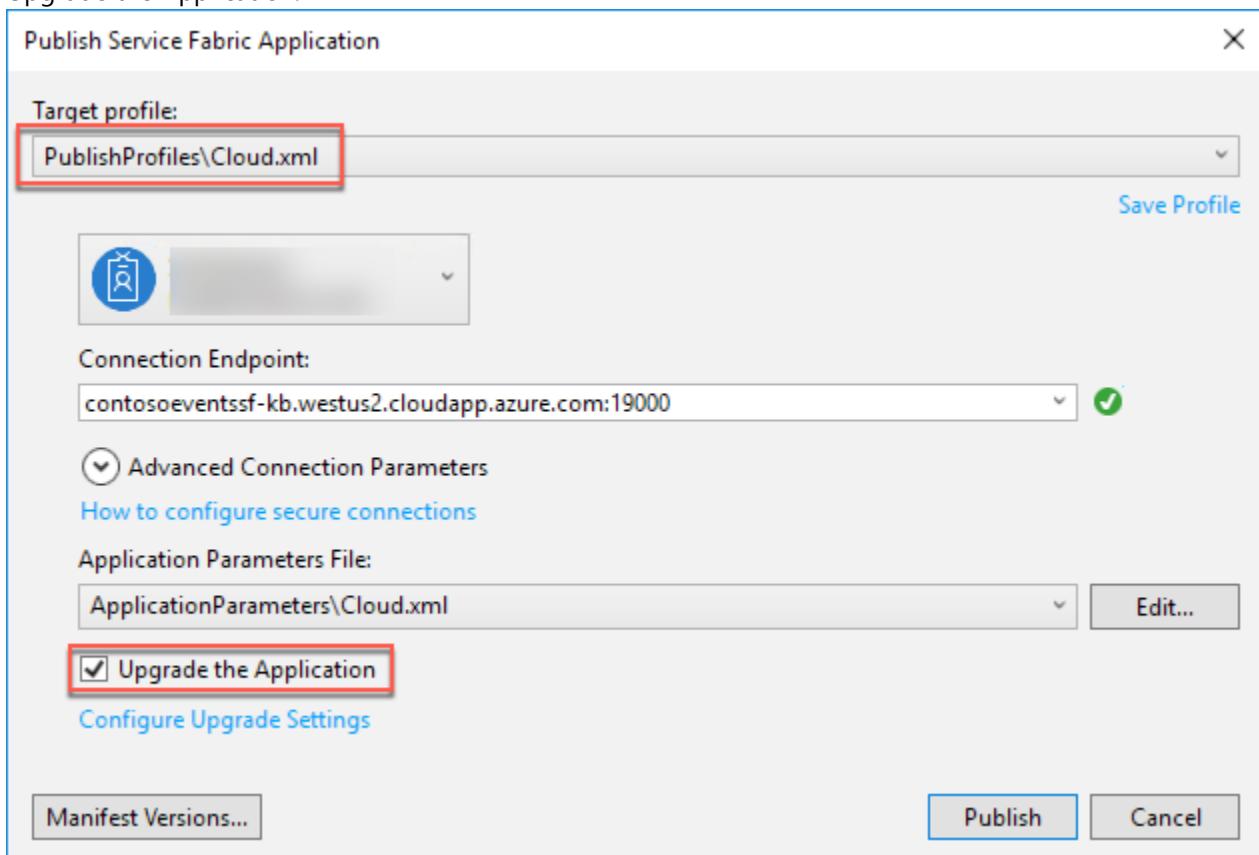
6. Save TicketOrderActor.cs.
7. After adding this field, the actor will now save it with each ticket order.
8. After making this change, rebuild the solution (Build menu, Rebuild solution), and verify that there are no errors.

Task 3: Perform a smooth upgrade

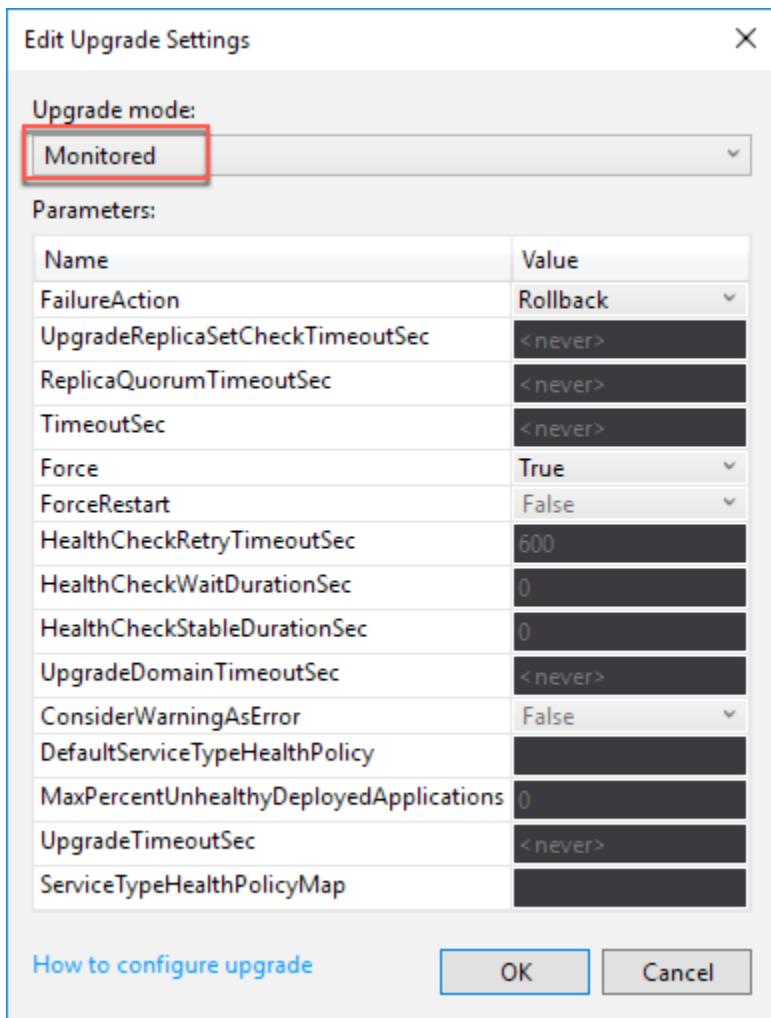
In this task, you will configure settings for the Service Fabric application to perform an upgrade.

1. From the Visual Studio Solution Explorer, expand the Service Fabric folder, then right-click ContosoEventsApp, and select Publish.

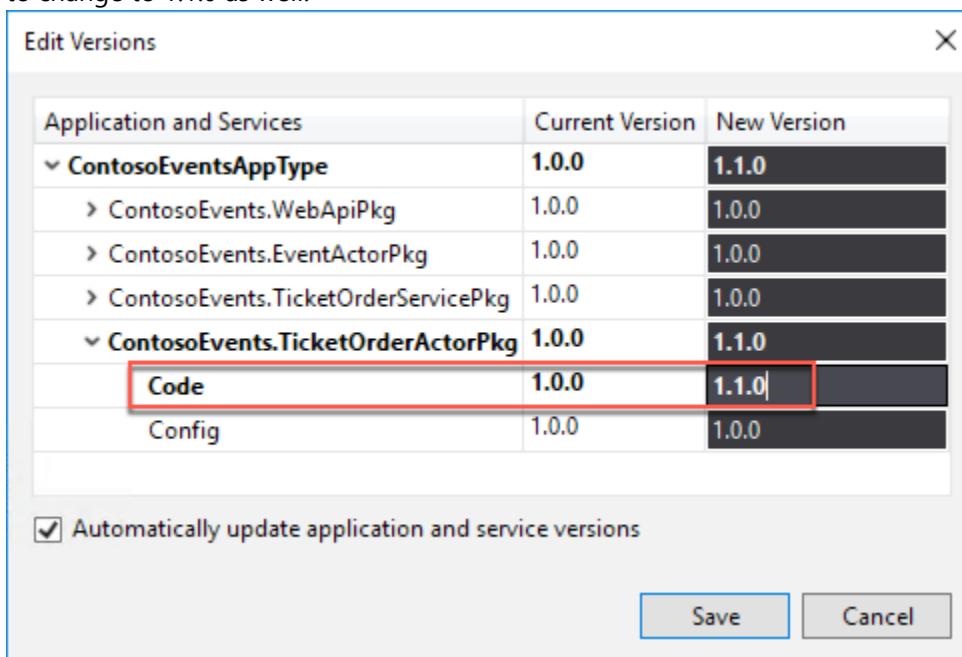
2. In the Public Service Fabric Application dialog, select PublishProfiles\Cloud.xml for the target profile, and check Upgrade the Application.



3. Select Configure Upgrade Settings, under Upgrade the Application. Select Monitored for the upgrade mode, then select OK.



4. From the Publish Service Fabric Application dialog, select Manifest Versions.
5. Change the TicketOrderActorPkg\Code New Version to 1.1.0. This change will force the actor package and the app to change to 1.1.0 as well.

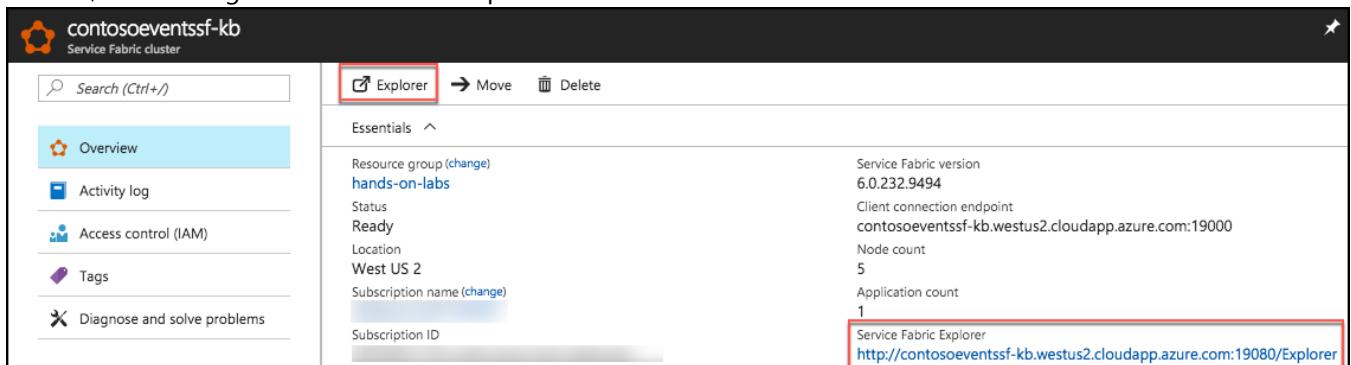


6. Select Save.
7. Now that the upgrade configuration is set, select Publish.
8. Observe the Visual Studio Output window going through the upgrade process, which can take 5 minutes or more.

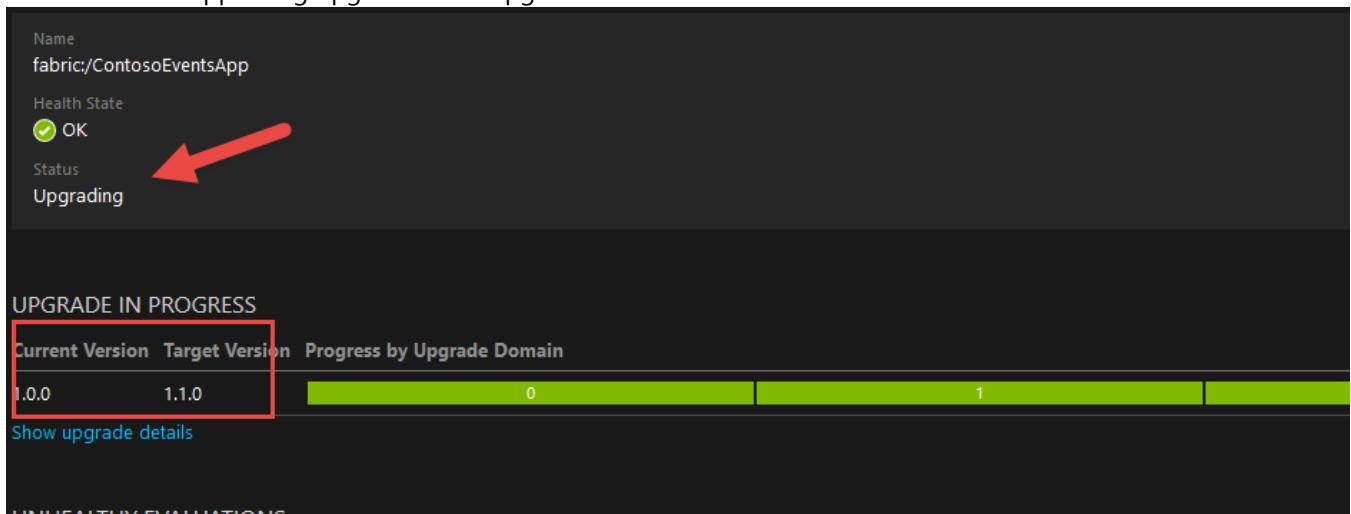
The screenshot shows the Visual Studio Output window with the title 'Output' and a dropdown menu 'Show output from: Build'. The log content is as follows:

```
8>Waiting for upgrade...
8>Upgrade completed successfully.
8>
```

9. Navigate to the Service Fabric Explorer for the deployment using your URL, something like: <http://contosoeventssf-SUFFIX.eastus.cloudapp.azure.com:19080/Explorer/index.html>. You can retrieve this from the Azure portal, by navigating to your Service Fabric cluster's blade by selecting the Explorer button on the toolbar, or selecting the Service Fabric Explorer link in the Essentials area.



10. It will show the app being upgraded one upgrade domain at a time.



11. When the upgrade is complete, from Service Fabric Explorer observe the new application version number.

The screenshot shows the Service Fabric Explorer interface for the application `fabric:/ContosoEventsApp`. The **ESSENTIALS** tab is active. In the top right, a red box highlights the **Version** field, which displays `1.1.0`. Below this, the **SERVICES** section lists four services with their respective service types and versions:

Name	Service Type	Version	Service Kind	Health State
<code>fabric:/ContosoEventsApp/EventActorService</code>	<code>EventActorServiceType</code>	1.0.0	Stateful	OK
<code>fabric:/ContosoEventsApp/TicketOrderActorService</code>	<code>TicketOrderActorServiceType</code>	1.1.0	Stateful	OK
<code>fabric:/ContosoEventsApp/TicketOrderService</code>	<code>TicketOrderServiceType</code>	1.0.0	Stateful	OK
<code>fabric:/ContosoEventsApp/WebApi</code>	<code>WebApiType</code>	1.0.0	Stateless	OK

Task 4: Submit a new order

Now that the upgrade is completed successfully, you will submit a new order, and make sure that the newly submitted order has the extended state.

1. This time, you will post an order using the Web API for the deployed service. The order can be something like this:

```
{  
    "userName": "testupgrade",  
    "email": "test.upgrade@gmail.com",  
    "eventId": "EVENT1-ID-00001",  
    "paymentProcessorTokenId": "ggashwh565-uiewuu87-ujdsk",  
    "tickets": 3  
}
```

2. Access the Swagger UI for your published Service Fabric Web API at a URL that looks like this:

<http://contosoeventssf-SUFFIX.eastus.cloudapp.azure.com:8082/swagger/ui/index>

3. Access the Orders API and select the POST method for the /api/orders endpoint. From there you can submit a new order using the JSON above.

POST /api/orders

Response Class (Status 200)

Response Content Type application/json ▾

Parameters

Parameter	Value	Description	Parameter Type	Data Type
order	{ "userName": "testupgrade", "email": "test.upgrade@gmail.com", "eventId": "EVENT1-ID-00001", "paymentProcessorTokenId": "ggashwh565-uiewuu87-ujdsk", "tickets": 3		body	Model Model Schema

Parameter content type: application/json ▾

Model Schema:

```
{
  "id": "string",
  "orderDate": "2017-12-21T21:42:33.940Z",
  "fulfillDate": "2017-12-21T21:42:33.940Z",
  "cancellationDate": "2017-12-21T21:42:33.940Z",
  "userName": "string",
  "email": "string",
  "tag": "string",
  "eventId": "string",
  "paymentProcessorTokenId": "string",
  "paymentProcessorConfirmation": "string",
  "tickets": 0,
}
```

Click to set as parameter value

Response Messages

HTTP Status Code	Reason	Response Model	Headers
400	An exception occurred		
404	NotFound		

Try it out!

4. Once you get back a 200 response code, the order id will be returned in the Response Body.

Try it out!
[Hide Response](#)

Curl

```
curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/json' -d '{
  "userName": "testupgrade",
  "email": "test.upgrade@gmail.com",
  "eventId": "EVENT1-ID-00001",
  "paymentProcessorTokenId": "ggashwh565-uiewuu87-ujdsk",
  "tickets": 3
}' 'http://contosoeventssf-kb.westus2.cloudapp.azure.com:8082/api/orders'
```

Request URL

```
http://contosoeventssf-kb.westus2.cloudapp.azure.com:8082/api/orders
```

Response Body

"4c90df2d-5f9b-4261-996a-894e25147514"

Response Code

```
200
```

Response Headers

```
{
  "access-control-allow-origin": "*",
  "date": "Fri, 22 Dec 2017 16:06:50 GMT",
  "server": "Microsoft-HTTPAPI/2.0",
  "content-length": "38",
  "content-type": "application/json; charset=utf-8"
}
```

5. In the Azure portal, navigate to your Cosmos DB account.
 6. Go to Cosmos DB Data Explorer, and query for the order id to see the new extended state actually persisted to the database, as you did in [Exercise 3, Task 6, Steps 7-9](#).

[New Collection](#)
[New SQL Query](#)
[New Stored Procedure](#)
[New UDF](#)
[New Trigger](#)
[Delete Collection](#)
[Delete Database](#)

COLLECTIONS

- ▼ **TicketManager**
 - ▼ **Orders**
 - Documents
 - Scale & Settings
 - ▶ Stored Procedures
 - ▶ User Defined Functions
 - ▶ Triggers
 - ▶ Events
- ▼ **ToDoList**

Query 1
X

Execute Query

```
1 SELECT * FROM c WHERE c.id = '4c90df2d-5f9b-4261-996a-894e25147514'
```

Results: 1 - 1 | Request Charge: 2.32 RUs | Next →

```
[{"id": "4c90df2d-5f9b-4261-996a-894e25147514", "OrderDate": "2017-12-22T16:06:48.5242242+00:00", "FulfillDate": "2017-12-22T16:06:53.0466188+00:00", "CancellationDate": null},
```

Exercise 7: Rollback

Duration: 30 minutes

In this exercise, you will deploy an update that causes issues with the application, and will trigger an automatic rollback of the deployment. This exercise will illustrate the mechanism Service Fabric provides for preserving application health and availability in production.

Task 1: Add health checks

In this task, you will add code to produce health checks that force the monitored upgrade to roll back to the original version of the service.

1. In Visual Studio, open TicketOrderService.cs located in the ContosoEvents.TicketOrderService project in the Service Fabric folder.
2. Locate TODO: Exercise 7 – Task 1 and uncomment:

```
//TODO: Exercise 7 - Task 1  
this.HealthReporterService.SendReportForService(HealthState.Error, "Simulated Error");
```

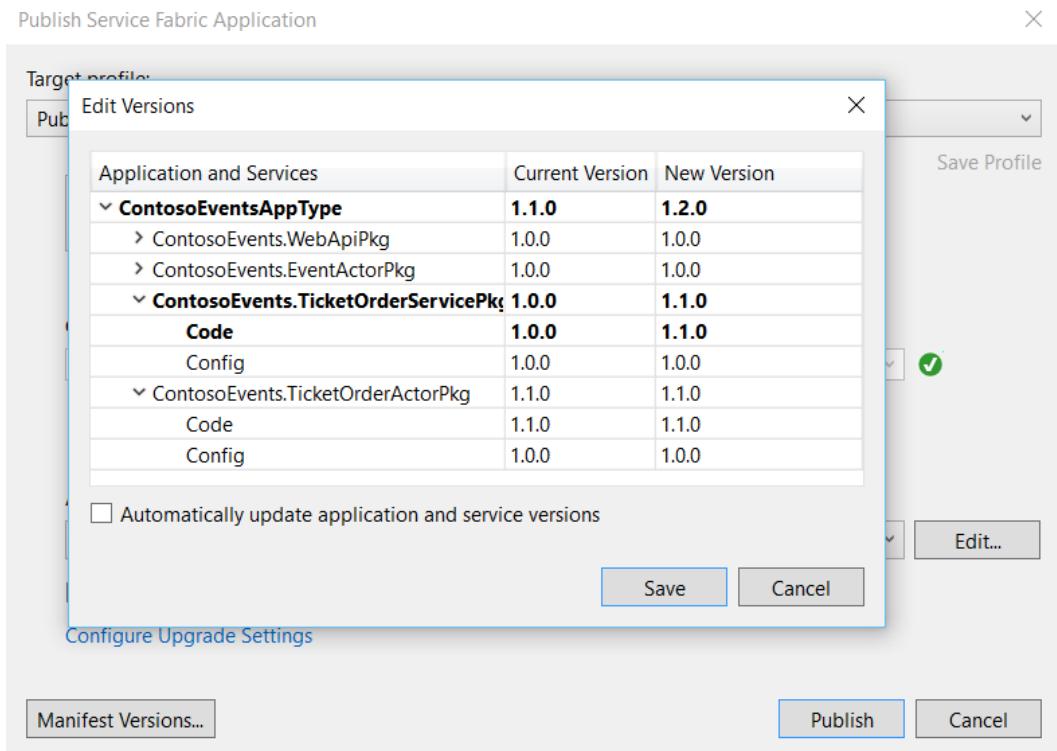
3. Compile the code to ensure there are no issues with the changes.

Task 2: Perform a troubled upgrade

In this task, you will perform an upgrade, and watch a rollback of the upgrade.

1. From Solution Explorer in Visual Studio the ContosoEventsApp project, under the Service Fabric folder, and select Publish.
2. From the Publish Service Fabric Application dialog, check Upgrade the Application, and click Manifest Versions.
3. Change the TicketOrderServicePkg\code New Version to 1.1.0. This will also require that the service package be changed to 1.1.0 and the application to 1.2.0.

4. Select Save.



- Select Publish, and return to Service Fabric Explorer.
- Observe the time it is taking to upgrade in Visual Studio and the errors that are now occurring in Service Fabric Explorer.
- Notice that the simulated error message 'Simulated error' appears in the event as shown in the following screen shot:

Kind	Health State	Description
Services	Error	Unhealthy services: 100% (1/1), ServiceType='TicketOrderServiceType', MaxPercentUnhealthyServices=0%.
Service	Error	Unhealthy service: ServiceName='fabric:/ContosoEventsApp/TicketOrderService', AggregatedHealthState='Error'.
Partitions	Error	Unhealthy partitions: 100% (1/1), MaxPercentUnhealthyPartitionsPerService=0%.
Partition	Error	Unhealthy partition: PartitionId='fac4f0d1-cc28-4da9-8291-a8e50f6df1ff', AggregatedHealthState='Error'.
Event	Error	Error event: Sourced='fabric:/ContosoEventsApp/TicketOrderService', Property='Simulated Error'.

Name	Service Type	Version	Service Kind	Health State
fabric:/ContosoEventsApp/EventActorService	EventActorServiceType	1.0.0	Stateful	OK
fabric:/ContosoEventsApp/TicketOrderActorService	TicketOrderActorServiceType	1.1.0	Stateful	OK
fabric:/ContosoEventsApp/TicketOrderService	TicketOrderServiceType	1.0.0	Stateful	Error
fabric:/ContosoEventsApp/WebApi	WebApiType	1.0.0	Stateless	OK

Note: Service Fabric is attempting to upgrade, and it is waiting for the service to report healthy status. Because this will not happen, eventually Service Fabric rolls back the upgrade.

8. In the meantime, observe that the application is still responsive. In the Swagger UI for your Service Fabric Web API, access the Events API and select GET /api/events to see a list of events as shown in the following screen shot:

The screenshot shows the Swagger UI interface for a Service Fabric Web API. At the top, there's a navigation bar with 'Events' selected, followed by 'Show/Hide', 'List Operations', and 'Expand Operations'. Below this, a 'GET /api/events' button is highlighted. The main area displays the 'Response Class (Status 200)' which is 'Ticket Events'. There are two tabs: 'Model' (selected) and 'Model Schema'. The 'Model Schema' tab shows a JSON schema for an event object, including properties like id, name, summary, description, imageUrl, latitude, longitude, and startDate. Below the schema, the 'Response Content Type' is set to 'application/json'. Under 'Response Messages', there are two entries: '400 An exception occurred' and '404 NotFound'. A 'Try it out!' button is at the bottom left.

9. So that you do not forget, return to the code and re-comment the failure report so that you can do the next exercise without rollback!

```
//TODO: Exercise 7 - Task 1
//this.HealthReporterService.SendReportForService(HealthState.Error, "Simulated Error");
```

10. Look at the Visual Studio output window and note that after some time (600 seconds by default, and this is configurable) Service Fabric gives up on the app and rolls back the upgrade as shown in the following screen shot:

The screenshot shows the Visual Studio Output window. The title bar says 'Output'. Under 'Show output from:', 'Build' is selected. The window displays several lines of text starting with '3>Waiting for upgrade...', followed by a red rectangular box around the line '3>Upgrade was Rolled back.'.

11. Return to Service Fabric Explorer note that version 1.2.0 (that contained the simulated error) was rolled back to version 1.1.0 the version that was running in your cluster before the attempted upgrade.
12. Also, note that the error remains until it is time to live expires or it is replaced by another health state. Until then, the unhealthy state will remain in the cluster, but the cluster will be running normally. You can proceed without clearing this for now.

13. The subsequent exercises (i.e. 8 and beyond) will be executed using version 1.1.0 and while the cluster shows unhealthy state. You can ignore this for now.

The screenshot shows the Microsoft Application Fabric interface for the 'ContosoEventsApp' application. The 'ESSENTIALS' tab is selected. In the top right, the 'Application Type' is listed as 'ContosoEventsAppType' and the 'Version' is '1.1.0'. Below this, the 'Health State' is shown as 'Error' with a red error icon. The status is 'Ready'. Under 'UNHEALTHY EVALUATIONS', there are five entries: Services (Error), Service (Error), Partitions (Error), Partition (Error), and Event (Error). Each entry provides a detailed description of the unhealthy state. In the 'SERVICES' section, there is a table listing four services: EventActorService, TicketOrderActorService, TicketOrderService, and WebApi. The 'TicketOrderService' row is highlighted with a red box around its 'Version' column, which shows '1.0.0'. The other columns in this row are 'Service Type' (TicketOrderServiceType), 'Service Kind' (Stateful), and 'Health State' (OK, indicated by a green checkmark).

Name	Service Type	Version	Service Kind	Health State
fabric/ContosoEventsApp/EventActorService	EventActorServiceType	1.0.0	Stateful	OK
fabric/ContosoEventsApp/TicketOrderActorService	TicketOrderActorServiceType	1.1.0	Stateful	OK
fabric/ContosoEventsApp/TicketOrderService	TicketOrderServiceType	1.0.0	Stateful	Error
fabric/ContosoEventsApp/WebApi	WebApiType	1.0.0	Stateless	OK

14. Optionally, from the Swagger endpoint for the deployed application, you can request health statistics at the following relative URL: /api/admin/applicationhealth.

Exercise 8: Load testing

Duration: 15 minutes

In this exercise, you will perform a load test against the Service Fabric Cluster and observe how messages are distributed across partitions.

Task 1: Simulate a 50-order request

In this task, you will simulate a load test of 50 orders against the cluster using the Web application to submit the load test and monitor partitions.

1. Navigate to the published Web application at a URL like <https://contosoeventsweb-SUFFIX.azurewebsites.net>.
2. Click the Load Test menu. Optionally give a new name to the tag for tracking. Set load to 50 requests. Click Start Load Test.

Load Test

Event :

Seattle Rock and Rollers

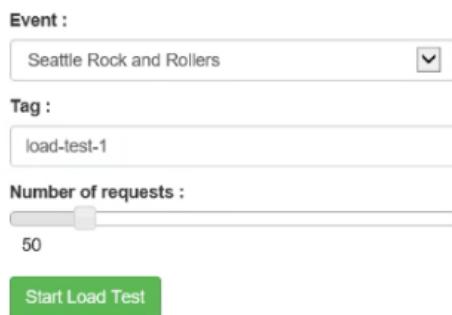
Tag :

load-test-1

Number of requests :

50

Start Load Test



3. Navigate to the Load Test Status menu. It shows you the partitions that were created for the ticket order service (reliable queue).

Simulation Status

Partition Id	Partition Status	Node Name	Health State	Items in Queue
075e7e0f-3f3b-47a9-af64-0aeb30d01bf7	Ready	_Web_4	Ok	0
fb057210-b8cf-4234-8295-1d0b6caa9e8c	Ready	_Web_3	Ok	0
ec94a533-3165-4514-b4d4-07935048eb25	Ready	_Web_1	Ok	0
f8a54270-0542-4990-8efc-bb61d1843108	Ready	_Web_2	Ok	0
5591d8d0-dd09-49b0-9acc-7098beb7ea63	Ready	_Web_0	Ok	0

4. While the load test is running, refresh this page and watch the changes to the Items in queue across partitions. It will fill while processing completes and then eventually drain.
5. After a few minutes you will see that the queues are drained, and the orders were processed.

Note: If you still have more time, run this again with additional iterations progressively such as 100, 150, 200, 250. Or, take a look at Exercise 9 which takes you through a much more comprehensive load test and partition analysis process using the API endpoint.

BONUS Exercise 9: Load testing w/ partitions

Note: If you completed the other work in good time and still have another 50 minutes free, have some load testing and partitioning fun! Otherwise, skip to [After the hands-on lab](#).

Duration: 50 minutes

In this task, you will perform several load tests against the Cluster using different partitions. This experimentation is crucial to get an idea of how many partitions you will really need. It is important to note that the number of partitions in your service cannot be changed after you have created the service. The main measure you are going to rely on is the average number of seconds it takes an order to be processed.

Task 1: Setting up for load testing

In this task, you will clean the existing orders from the Cosmos DB, so you can start with clean slate. This step is optional but because we are going to rely on reporting about orders in the database, it is much better to keep in the database only the orders that were simulated for load testing.

1. In the Swagger UI for your Service Fabric Web API, please access the admin APIs `DELETE /api/admin/orders` and issue the API. When you get back a 200-response code, the orders will have been deleted from the Cosmos DB.

ContosoEvents.WebApi

Admin

Show/Hide | List Operations | Expand Operations

`GET /api/admin/partitions`

`GET /api/admin/applicationhealth`

`GET /api/admin/servicehealth/{servicename}`

`GET /api/admin/actorhealth/{actorname}`

`POST /api/admin/simulate/orders`

`PUT /api/admin/health/service`

`DELETE /api/admin/events`

`DELETE /api/admin/orders`

Response Class (Status 200)

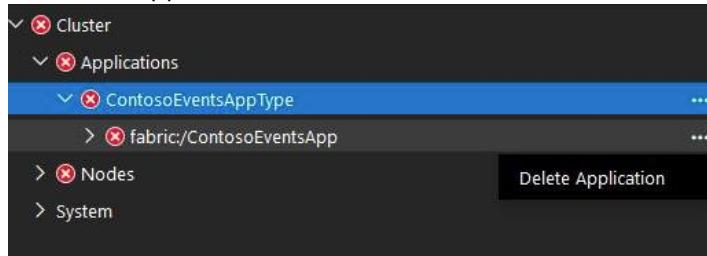
Response Content Type `application/json`

Response Messages

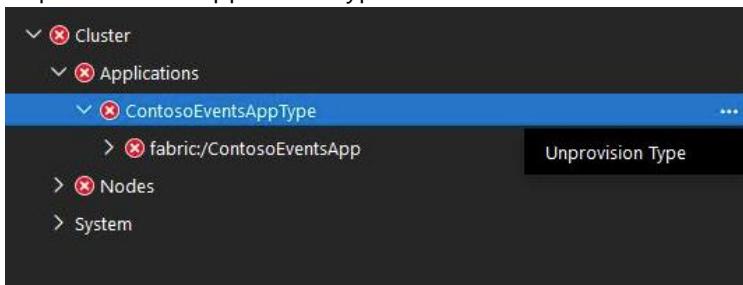
HTTP Status Code	Reason	Response Model	Headers
<code>400</code>	An exception occurred		
<code>404</code>	NotFound		

`Try it out!`

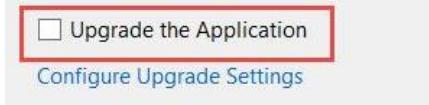
2. Because you have previously performed an upgrade to the Service Fabric application, and we want to play with partition sizes in this exercise, you will unprovision the application type using Service Fabric Explorer. Navigate to the explorer.
3. Delete the application.



4. Unprovision the application type.

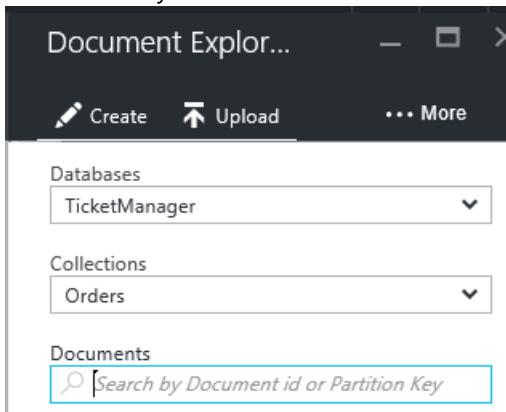


5. In the tasks that follow in this exercise you will not be upgrading when you publish the application. You will be re-publishing. Make sure Upgrade the application checkbox is unchecked each time you publish.



6. Service Fabric Explorer shows no application types or applications deployed.

7. You can verify from Cosmos DB Orders Document Explorer that there are no more orders in the database:



Task 2: Using the current partition count, simulate 100 orders

In this task, you will simulate a load test against the Cluster using the current partition count (that is, 5) of the Ticket Order Service.

1. Using the Service Fabric Web API Swagger UI, please issue a POST request against /api/admin/simulate/orders

The screenshot shows the Swagger UI for the /api/admin/simulate/orders endpoint. At the top, there's a green button labeled 'POST' and the URL '/api/admin/simulate/orders'. Below this, a section titled 'Response Class (Status 200)' is shown. Under 'Parameters', there's a table with one row. The 'Value' column contains a large text input field with '(required)' placeholder text. A red callout box with the text 'Place the simulation request here' points to this input field. To the right of the table, there's a 'Model Schema' box containing a JSON object:

```
{  
  "baseUrl": "string",  
  "eventId": "string",  
  "userName": "string",  
  "email": "string",  
  "tag": "string",  
  "iterations": 0  
}
```

Below the schema, a link says 'Click to set as parameter value'. Further down, under 'Response Messages', there are two entries: '400 An exception occurred' and '404 NotFound'. At the bottom left of this section is a button labeled 'Try it out!'

2. A sample simulation order request can be formed as follows. The tag identifies the simulation run so we can report on it later. Be sure to replace the values in the baseUrl property:

```
{  
  "baseUrl": "http://<your-cluster-name>.<your-region>.cloudapp.azure.com:8082",  
  "eventId": "EVENT1-ID-00001",  
  "userName": "tester",  
  "email": "load.tester@gmail.com",  
  "tag": "partitions-5-100",  
  "iterations": 100  
}
```

3. As the simulation request is processed, please issue a GET request against /api/admin/partitions to see how the different partitions are functioning. This returns partition info, health, and the number of items in each partition queue. You should notice that the queues fill up with order requests and are drained.

Admin

GET /api/admin/partitions

Response Class (Status 200)
Ticket Order Partitions

Model Model Schema

```
[  
  {  
    "partitionId": "string",  
    "partitionKind": "string",  
    "partitionStatus": "string",  
    "nodeName": "string",  
    "healthState": "string",  
    "serviceKind": "string",  
    "itemsInQueue": 0  
  }]
```

Response Content Type application/json ▾

Response Messages

HTTP Status Code	Reason	Response Model	Headers
400	An exception occurred		
404	NotFound		

[Try it out!](#) [Hide Response](#)

4. The response might look like this.

```
[  
  {  
    "partitionId": "856b5048-bb35-45fb-b804-f91f4fb48ccf",  
    "partitionKind": "Int64Range",  
    "partitionStatus": "Ready",  
    "nodeName": "_Node_4",  
    "healthState": "Ok",  
    "serviceKind": "Stateful",  
    "itemsInQueue": 10  
  },  
  {  
    "partitionId": "f912eb50-bf89-4be9-8d39-61dfc3b69a01",  
    "partitionKind": "Int64Range",  
    "partitionStatus": "Ready",  
    "nodeName": "_Node_3",  
    "healthState": "Ok",  
    "serviceKind": "Stateful",  
    "itemsInQueue": 10  
  }]
```

```
        "healthState": "Ok",
        "serviceKind": "Stateful",
        "itemsInQueue": 8
    },
    {
        "partitionId": "acefa86e-c57c-4a84-b6cb-230d6c8a9e23",
        "partitionKind": "Int64Range",
        "partitionStatus": "Ready",
        "nodeName": "_Node_2",
        "healthState": "Ok",
        "serviceKind": "Stateful",
        "itemsInQueue": 6
    },
    {
        "partitionId": "2c6c7dd9-8ff4-4303-bd1f-fb48ed51952d",
        "partitionKind": "Int64Range",
        "partitionStatus": "Ready",
        "nodeName": "_Node_1",
        "healthState": "Ok",
        "serviceKind": "Stateful",
        "itemsInQueue": 6
    },
    {
        "partitionId": "1da411e7-577f-4440-a3ab-257962051b0c",
        "partitionKind": "Int64Range",
        "partitionStatus": "Ready",
        "nodeName": "_Node_0",
        "healthState": "Ok",
        "serviceKind": "Stateful",
        "itemsInQueue": 0
    }
]
```

5. Wait for 2-3 minutes to make sure that queues are drained and that orders are processed before completing this task.

Task 3: Using a partition count of 10, simulate 100 orders

In this task, you will repeat the same load test you performed in Task 2 but using a partition count of 10.

1. Change the service and actor partition count to 10 in the Cloud.xml Application Parameters file.
2. Double-click the document in Visual Studio and make the following changes:

```
<Parameter Name="TicketOrderService_PartitionCount" Value="10" />
<Parameter Name="TicketOrderService_MinReplicaSetSize" Value="3" />
<Parameter Name="TicketOrderService_TargetReplicaSetSize" Value="3" />
<Parameter Name="WebApi_InstanceCount" Value="5" />
<Parameter Name="TicketOrderActorService_PartitionCount" Value="10" />
<Parameter Name="EventActorService_PartitionCount" Value="1" />
```

3. Using Visual Studio Publish command, redeploy the app to the cluster (as shown in a previous task).
4. Repeat the same test load you did in Task 2 but with a different tag:

```
{
    "baseUrl": "http://<your-cluster-name>.<your-region>.cloudapp.azure.com:8082",
    "eventId": "EVENT1-ID-00001",
    "userName": "tester",
    "email": "load.tester@gmail.com",
    "tag": "partitions-10-100",
    "iterations": 100
}
```

5. Wait for 2-3 minutes until to make sure that queues are drained and that orders are processed before completing this task.

Task 4: Using a partition count of 15, simulate 100 orders

In this task, you will repeat the same load test you performed in Task 2 but using a partition count of 15.

1. Change the service and actor partition count to 15 in the Cloud.xml Application Parameters file.

2. Double-click the document in Visual Studio and make the following changes:

```
<Parameter Name="TicketOrderService_PartitionCount" Value="15" />
<Parameter Name="TicketOrderService_MinReplicaSetSize" Value="3" />
<Parameter Name="TicketOrderService_TargetReplicaSetSize" Value="3" />
<Parameter Name="WebApi_InstanceCount" Value="5" />
<Parameter Name="TicketOrderActorService_PartitionCount" Value="15" />
<Parameter Name="EventActorService_PartitionCount" Value="1" />
```

3. Using the Visual Studio Publish command, redeploy the app to the cluster (as shown in a previous task).

4. Repeat the same test load you did in Task 2 but with a different tag:

```
{
    "baseUrl": "http://<your-cluster-name>.<your-region>.cloudapp.azure.com:8082",
    "eventId": "EVENT1-ID-00001",
    "userName": "tester",
    "email": "load.tester@gmail.com",
    "tag": "partitions-15-100",
    "iterations": 100
}
```

5. Wait for 2-3 minutes until to make sure that queues are drained and that orders are processed before completing this task.

Task 5: Using a partition count of 20, simulate 100 orders

In this task, you will repeat the same load test you performed in Task 2 but using a partition count of 20.

1. Change the service and actor partition count to 20 in the Cloud.xml Application Parameters file.
2. Double-click the document in Visual Studio and make the following changes:

```
<Parameter Name="TicketOrderService_PartitionCount" Value="20" />
<Parameter Name="TicketOrderService_MinReplicaSetSize" Value="3" />
<Parameter Name="TicketOrderService_TargetReplicaSetSize" Value="3" />
<Parameter Name="WebApi_InstanceCount" Value="5" />
<Parameter Name="TicketOrderActorService_PartitionCount" Value="20" />
<Parameter Name="EventActorService_PartitionCount" Value="1" />
```

3. Using Visual Studio Publish command, redeploy the app to the cluster (as shown in a previous task).

4. Repeat the same test load you did in Task 2 but with a different tag:

```
{  
  "baseUrl": "http://<your-cluster-name>.<your-region>.cloudapp.azure.com:8082",  
  "eventId": "EVENT1-ID-00001",  
  "userName": "tester",  
  "email": "load.tester@gmail.com",  
  "tag": "partitions-20-100",  
  "iterations": 100  
}
```

5. Wait for 2-3 minutes until to make sure that queues are drained and that orders are processed before completing this task.

Task 6: Determine the performance variations among the different load tests

In this task, you will find out how each load test performed and plot the result.

1. Using the Service Fabric Web API Swagger UI, please issue a GET request against /api/orders/stats.

The screenshot shows the Swagger UI interface for the /api/orders/stats endpoint. At the top, there is a 'GET' button and the URL '/api/orders/stats'. Below this, the 'Response Class (Status 200)' section is titled 'Get Order Stats'. Underneath, there are two tabs: 'Model' (selected) and 'Model Schema'. The 'Model Schema' tab displays the following JSON schema:

```
[  
  {  
    "tag": "string",  
    "count": 0,  
    "sumSeconds": 0,  
    "averageSeconds": 0  
  }  
]
```

Below the schema, the 'Response Content Type' is set to 'application/json'. The 'Response Messages' section contains two entries:

HTTP Status Code	Reason	Response Model	Headers
400	An exception occurred		
404	NotFound		

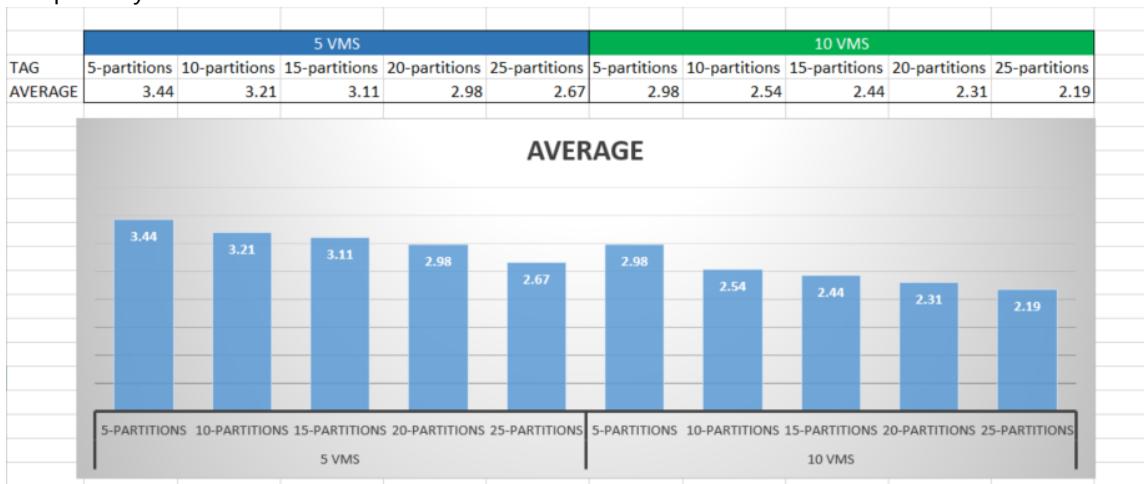
At the bottom left, there is a 'Try it out!' button.

2. The command returns stats about each load test you performed in JSON that looks like this:

```
[  
  {  
    "tag": "partitions-5-100",  
    "count": 100,  
    "sumSeconds": 879.2576443999999,  
    "averageSeconds": 8.792576443999998  
  },  
  {  
    "tag": "partitions-10-100",  
    "count": 100,  
    "sumSeconds": 340.0996231000001,  
    "averageSeconds": 3.400996231000001  
  },  
  {  
    "tag": "partitions-15-100",  
    "count": 100,  
    "sumSeconds": 162.2106096,  
    "averageSeconds": 1.622106096  
  },  
  {  
    "tag": "partitions-20-100",  
    "count": 100,  
    "sumSeconds": 165.1475782000003,  
    "averageSeconds": 1.651475782000003  
  }]  
]
```

3. Scale out the VMs count to 10, for example, in your cluster and repeat tasks 2, 3, 4, and 5. This step is out of scope and is shown here only to complete the picture.

4. Given the load test results for each Cluster configuration, produce an Excel sheet from the data you collected. A sample may look like this:



5. Armed with an average number of seconds it takes to process orders for different partition and VMs count, you are in a better situation to recommend a partition count for your application.

BONUS Exercise 10: Secure the web application

If you completed the other work in good time and still have another 30 minutes, do this exercise and integrate Azure Active Directory B2C with the application. Otherwise, skip to [After the hands-on lab](#).

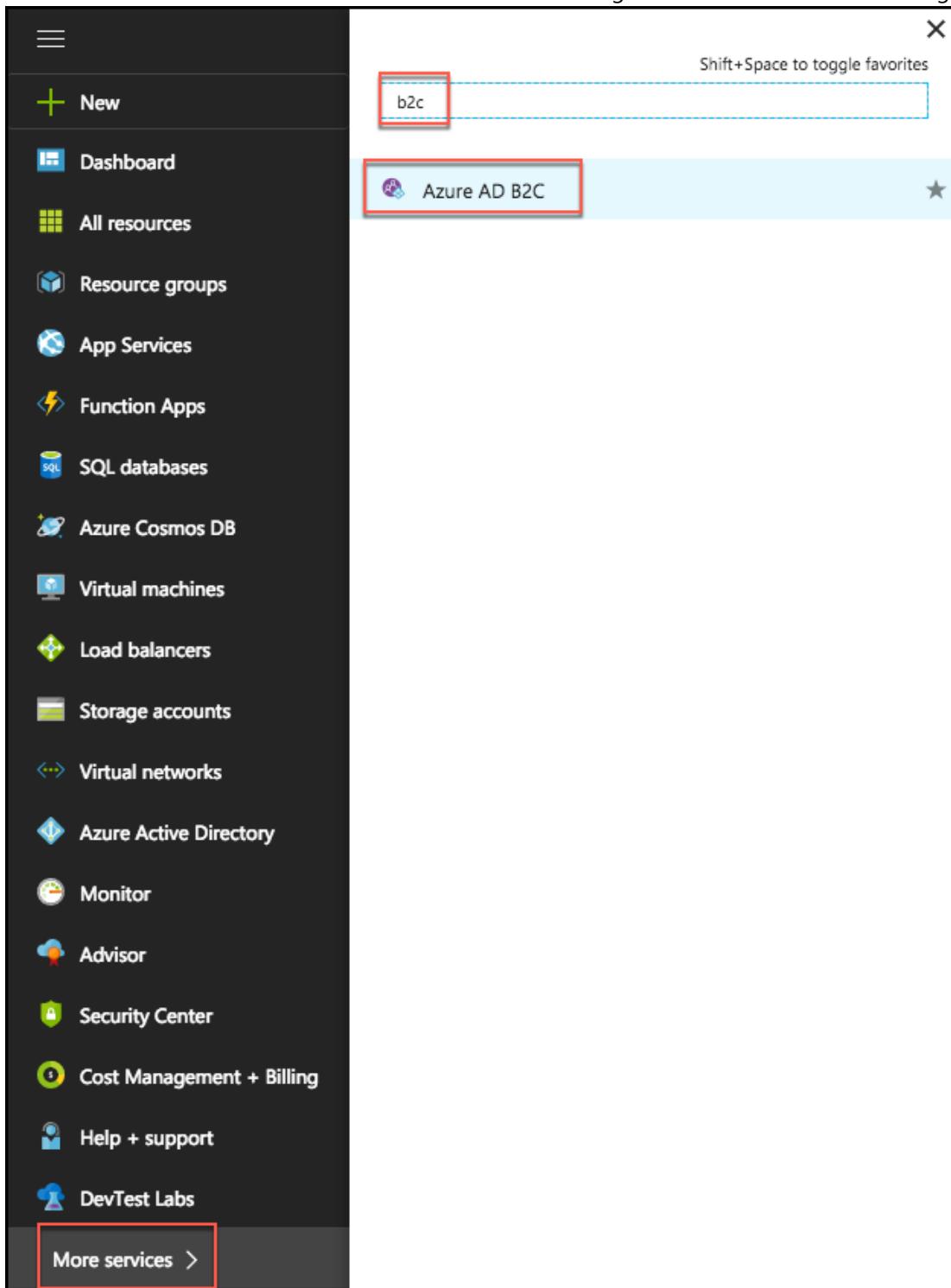
Duration: 30 minutes

In this exercise, you will configure the Azure Active Directory B2C tenant and configure the Web front end to integrate with it for user login. Once all the settings are in place, you will be able to run the website, register as a user, and login.

Task 1: Configure the Azure Active Directory B2C

In this task, you will set up the Azure Active Directory B2C directory for your application to integrate with it.

1. From the Azure Portal browse to Azure B2C. Select it to navigate to the Azure AD B2C Settings blade.



2. You should see the domain name you created earlier for your B2C directory.

Note: If you see an authorization error, you may have to select the directory from your profile menu and then repeat steps 1 and 2. Further, if your directory does not yet appear in the profile list, you should refresh the portal page to see it in the list.

Azure AD B2C SETTINGS
contosoeventsb2csoll3.onmicrosoft.com

Settings

Essentials ^

Domain name: contosoeventsb2csoll3.onmicrosoft.com Tenant type: Production-scale tenant

All settings →

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

Quick Start

Submit support request

A red arrow points to the 'Domain name' field.

CONTOSOEVENTSB2CSOLL3

Sign out

Change password

My permissions

Submit an idea

View my bill

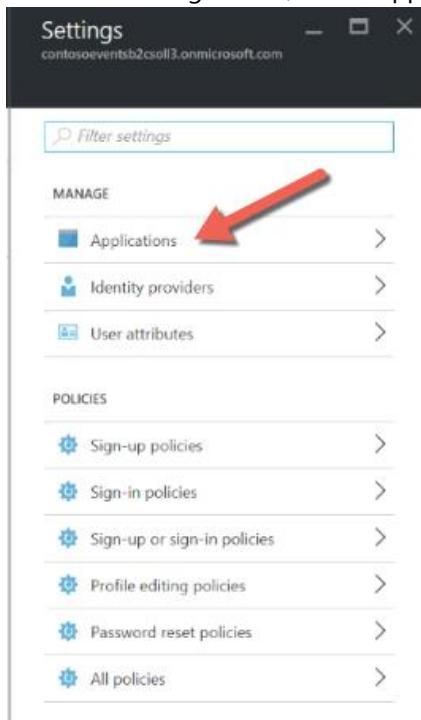
DIRECTORY

contosoeventsb2csoll3

contosoeventsb2csoll3.onmicrosoft.com

A red arrow points to the 'DIRECTORY' option in the profile menu.

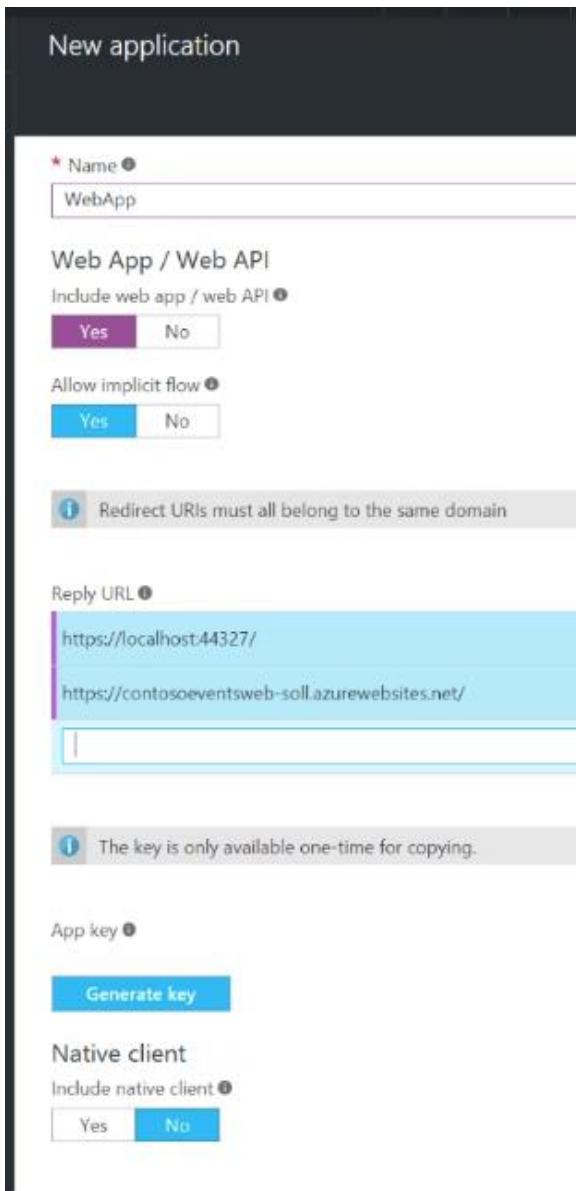
3. From the Settings blade, select Applications.



4. Click +Add.
5. Set the application name to WebApp.
6. Select Yes for include Web App / Web API.
7. Select Yes for Allow implicit flow.
8. Add a reply URL for local testing: <https://localhost:44327/>
9. Add a reply URL for the hosted Web App as you named it, for example: <https://contosoeventsweb-SUFFIX.azurewebsites.net/>

Note: Make sure to include the closing / or the configuration will not work.

10. Click Create.



11. In the Settings blade, select Identity providers.

12. Select Username for Local accounts.

13. Click Save.

The screenshot shows two adjacent browser tabs. The left tab is titled 'Settings' and has a URL of '.onmicrosoft.com - PREVIEW'. It contains a sidebar with sections for 'MANAGE' (Applications, Identity providers, User attributes) and 'POLICIES' (Sign-up policies, Sign-in policies, Sign-up or sign-in policies, Profile editing policies, Password reset policies, All policies). The 'Identity providers' item in the Manage section is highlighted. The right tab is titled 'Identity provider' and also has a URL of '.onmicrosoft.com - PREVIEW'. It shows a 'Local accounts' section with a dropdown menu set to 'Username'. Below it is a 'Social identity providers' section with the message 'No identity providers are defined in the tenant'. At the top of the right tab, there are buttons for '+ Add', 'Save' (which is highlighted in blue), and 'Discard'.

14. In the Settings blade, select Sign-up policies.

15. Click + Add.

16. Set the policy name to 'signup'.

The screenshot shows three windows from the Microsoft Azure portal:

- Settings .onmicrosoft.com - PREVIEW**: Shows the 'MANAGE' section with 'Sign-up policies' selected. A blue box highlights the 'Sign-up policies' link under 'POLICIES'.
- Sign-up policies PREVIEW**: Shows a search bar and a message 'No policies found'.
- Add sign-up policy PREVIEW**: A configuration page for creating a new policy.
 - Name**: The input field contains 'signup'.
 - * Identity providers**: Sub-section with '0 Selected'.
 - Sign-up attributes**: Sub-section with '0 Selected'.
 - Application claims**: Sub-section with '0 Selected'.
 - Multifactor authentication**: Sub-section with 'Off'.
 - Page UI customization**: Sub-section with 'Default'.

A large blue box highlights the 'Create' button at the bottom right of the third window.

17. Select Identity providers.

18. Select User ID signup.

19. Click OK.

The screenshot shows two overlapping windows. The left window is titled 'Add sign-up policy' with a 'PREVIEW' link at the top right. It has a 'Name' field containing 'signup' with a green checkmark. Below it is a section titled 'Identity providers' with a note '0 Selected'. Other sections include 'Sign-up attributes' (0 Selected), 'Application claims' (0 Selected), 'Multifactor authentication' (Off), and 'Page UI customization' (Default). At the bottom are 'Create' and 'OK' buttons. The right window is titled 'Select identity providers' with a 'PREVIEW' link at the top right. It lists a single provider: 'User ID signup' under 'Local Account' with a checked checkbox.

NAME	IDENTITY PROVIDER
User ID signup	Local Account

20. Select Sign-up attributes.

21. Select Email Address, Given Name, and Surname.

22. Click OK.

The screenshot shows two adjacent windows side-by-side. The left window is titled 'Add sign-up policy' and has a 'PREVIEW' button at the top right. It contains the following configuration:

- Name: signup
- * Identity providers: 1 Selected
- Sign-up attributes: 0 Selected (highlighted in blue)
- Application claims: 0 Selected
- Multifactor authentication: Off
- Page UI customization: Default

The right window is titled 'Select sign-up attributes' and also has a 'PREVIEW' button at the top right. It displays a list of user attributes with checkboxes:

NAME	DATA TYPE	DESCRIPTION
City	String	The city in which the user is located.
Country/Region	String	The country/region in which the user is located.
Display Name	String	
<input checked="" type="checkbox"/> Email Address	String	
<input checked="" type="checkbox"/> Given Name	String	The user's given name (also known as first name).
Job Title	String	The user's job title.
Postal Code	String	The postal code of the user's address.
State/Province	String	The state or province in user's address.
Street Address	String	The street address where the user is located
<input checked="" type="checkbox"/> Surname	String	The user's surname (also known as family name) or

At the bottom of each window are 'Create' and 'OK' buttons.

23. Select Application Claims.

24. Select Email Addresses, Given Name, Surname, and User's Object ID.

25. Click OK.

The screenshot shows the 'Add sign-up policy' dialog with the 'PREVIEW' tab selected. On the left, under 'Application claims', there is a section titled '0 Selected'. On the right, a table titled 'Select application claims' lists various user attributes:

NAME	CLAIM TYPE	DATA TYPE	DESCRIPTION
City	city	String	The city in which the user is located.
Country/Region	country	String	The country/region in which the user is located.
Display Name	displayName	String	
Email Addresses	emails	StringCollection	Email addresses of the user.
Given Name	givenName	String	The user's given name (also known as first name).
Identity Provider	identityProvider	String	
Job Title	jobTitle	String	The user's job title.
Postal Code	postalCode	String	The postal code of the user's address.
State/Province	state	String	The state or province in user's address.
Street Address	streetAddress	String	The street address where the user is located.
Surname	surname	String	The user's surname (also known as family name).
User is new	newUser	Boolean	
User's Object ID	objectId	String	Object identifier (ID) of the user object.

At the bottom of the dialog, there are 'Create' and 'OK' buttons.

26. Select Token, session & SSO config.

27. Select acr under Claim representing policy ID.

28. Click OK.

The screenshot shows two overlapping dialog boxes. The left dialog is titled 'Edit policy' and has a sub-section title 'B2C_1_signup'. It contains fields for 'Name' (set to 'B2C_1_signup'), 'Identity providers' (1 Selected), 'Sign-up attributes' (3 Selected), 'Application claims' (5 Selected), 'Token, session & SSO config' (Default selected), 'Multifactor authentication' (Off), 'Page UI customization' (Default), and 'Language customization (Preview)' (Disabled). The right dialog is titled 'Token, session & SSO config' and has a sub-section title 'B2C_1_signup'. It contains sections for 'Token lifetimes' (Access & ID token lifetimes set to 60 minutes, Refresh token lifetime set to 14 days, Refresh token sliding window lifetime set to 90 days), 'Token compatibility settings' (Issuer (iss) claim set to https://login.microsoftonline.com/878eafcf-4710-4b0c-bd6a-2be2aa0e67ad, Subject (sub) claim set to ObjectID (Not supported), Claim representing policy ID set to tfp and acr), 'Session behavior' (Web app session lifetime set to 1440 minutes, Web app session timeout set to Rolling), and 'Single sign-on configuration' (Tenant selected). A red box highlights the 'tfp' and 'acr' buttons in the 'Claim representing policy ID' section. A blue 'OK' button is at the bottom of the right dialog.

29. Click Create.

30. In the Settings blade, select Sign-in policies.
31. Click + Add.

32. Set the policy name to 'signin'.

The screenshot shows three windows from the Microsoft Azure portal:

- Settings**: Shows the 'MANAGE' section with 'Applications', 'Identity providers', and 'User attributes'. The 'POLICIES' section is expanded, showing 'Sign-up policies', 'Sign-in policies' (which is selected), 'Sign-up or sign-in policies', 'Profile editing policies', 'Password reset policies', and 'All policies'.
- Sign-in policies**: Shows a search bar and a message 'No policies found'.
- Add sign-in policy**: A configuration page with the following fields:
 - Name**: 'signin' (highlighted with a green checkmark).
 - Identity providers**: '0 Selected'.
 - Application claims**: '0 Selected'.
 - Multifactor authentication**: 'Off'.
 - Page UI customization**: 'Default' (locked).

A blue 'Create' button is located at the bottom right of the 'Add sign-in policy' window.

33. Select Identity providers.

34. Select Local Account Siginin.

35. Click OK.

The screenshot shows two windows side-by-side. The left window is titled 'Add sign-in policy' with a 'PREVIEW' link at the bottom right. It has a 'Name' field containing 'signin' with a green checkmark. Below it is a section titled 'Identity providers' with a blue background, showing '0 Selected'. Other sections include 'Application claims' (0 Selected), 'Multifactor authentication' (Off), and 'Page UI customization' (Default). At the bottom are 'Create' and 'OK' buttons. The right window is titled 'Select identity providers' with a 'PREVIEW' link at the bottom right. It has a table with two columns: 'NAME' and 'IDENTITY PROVIDER'. A single row is shown with a checked checkbox next to 'Local Account Signin' under both columns. At the bottom of this window is an 'OK' button.

36. Select Application Claims.

37. Select Email Addresses, Given Name, Surname, and User's Object ID.

38. Click OK.

The screenshot shows two overlapping windows. The top window is titled 'Select application claims' and has a 'PREVIEW' button at the bottom right. It contains a table with columns: NAME, CLAIM TYPE, DATA TYPE, DESCRIPTION, and ATTRIBUTE TYPE. Several rows are listed, including 'City', 'Country/Region', 'Display Name', 'Email Addresses' (which is selected), 'Given Name' (which is selected), 'Identity Provider', 'Job Title', 'Postal Code', 'State/Province', 'Street Address', 'Surname' (which is selected), and 'User's Object ID'. The bottom window is titled 'Add sign-in policy' and has a 'PREVIEW' button at the bottom right. It shows a configuration page with sections for 'Name' (set to 'signin'), 'Identity providers' (1 Selected), 'Application claims' (0 Selected, highlighted in blue), 'Multifactor authentication' (Off), and 'Page UI customization' (Default). At the bottom are 'Create' and 'OK' buttons.

39. Select Token, session & SSO config.

40. Select acr under Claim representing policy ID.

41. Click OK.

The screenshot shows two overlapping windows from the Azure portal:

- Edit policy (B2C_1_signin)**:
 - Name: B2C_1_signin
 - * Identity providers: 1 Selected
 - Application claims: 5 Selected
 - Token, session & SSO config (Default): This item is highlighted with a blue background.
 - Multifactor authentication: Off
 - Page UI customization: Default
 - Language customization (Preview): Disabled
- Token, session & SSO config (B2C_1_signin)**:
 - Token lifetimes**: Access & ID token lifetimes (minutes): 60
 - Refresh token lifetime (days): 14
 - Refresh token sliding window lifetime (days): Bounded (selected), No expiry (disabled), 90 (checked)
 - Token compatibility settings**:
 - Issuer (iss) claim: https://login.microsoftonline.com/878eafc...
 - Subject (sub) claim: ObjectId (selected), Not supported (disabled)
 - Claim representing policy ID: tfp (disabled), acr (selected)
 - Session behavior**:
 - Web app session lifetime (minutes): 1440
 - Web app session timeout: Absolute (selected), Rolling (disabled)
 - Single sign-on configuration: Tenant (selected), Application, Policy, Disabled

42. Click Create.

43. In the Settings blade, select Profile editing policies.

44. Click + Add.

45. Set the policy name to 'profileedit'.

The screenshot shows two side-by-side browser windows. The left window is titled 'Profile editing poli...' and has a 'PREVIEW' button at the top. It features a search bar and a large message 'No policies found'. Below that are three sections: 'Identity providers', 'Profile attributes', and 'Application claims', each with a count of '0 Selected'. The right window is titled 'Add profile editin...' and also has a 'PREVIEW' button. It has a 'Name' field containing 'profileedit'. Below it are the same three sections as the left window, with a lock icon next to the 'Page UI customization' section. A large blue 'Create' button is at the bottom of this window.

46. Select Identity providers.
47. Select Local Account Signin.

48. Click OK.

The screenshot shows two overlapping windows. The top window is titled 'Select identity providers' and has a URL 'onmicrosoft.com - PREVIEW'. It contains a table with one row:

NAME	IDENTITY PROVIDER
<input checked="" type="checkbox"/> Local Account Signin	Local Account Signin

The bottom window is titled 'Add profile editin...' and also has a URL 'onmicrosoft.com - PREVIEW'. It contains several sections: 'Name' (profileedit), 'Identity providers' (0 Selected), 'Profile attributes' (0 Selected), 'Application claims' (0 Selected), and 'Page UI customization' (Default). At the bottom are 'Create' and 'OK' buttons.

49. Select Profile attributes.

50. Select Given Name and Surname.

51. Click OK.

The screenshot shows two overlapping windows. The top window is titled 'Select profile attributes' and has a 'PREVIEW' tab selected. It contains a table with columns: NAME, DATA TYPE, DESCRIPTION, and ATTRIBUTE TYPE. Two rows are highlighted with blue borders: 'Given Name' (String, Description: 'The user's given name (also known as first name)', Attribute Type: 'Built-in') and 'Surname' (String, Description: 'The user's surname (also known as family name or last name)', Attribute Type: 'Built-in'). The bottom window is titled 'Add profile editin...' and has a 'PREVIEW' tab selected. It contains a table with columns: NAME, DATA TYPE, DESCRIPTION, and ATTRIBUTE TYPE. The 'Profile attributes' section is expanded, showing 'Given Name' and 'Surname' listed under it. The 'User attributes' section is collapsed. At the bottom of both windows are 'Create' and 'OK' buttons.

NAME	DATA TYPE	DESCRIPTION	ATTRIBUTE TYPE
Given Name	String	The user's given name (also known as first name).	Built-in
Surname	String	The user's surname (also known as family name or last name).	Built-in

52. Select Application Claims.

53. Select Email Addresses, Given Name, Surname, and User's Object ID.

54. Click OK.

The screenshot shows two windows side-by-side. The left window is titled 'Add profile editin...' and has sections for 'Name' (set to 'profileedit'), 'Identity providers' (1 Selected), 'Profile attributes' (2 Selected), 'Application claims' (0 Selected), and 'Page UI customization' (Default). The right window is titled 'Select application claims' and lists various attributes with checkboxes. The checked attributes are: Email Addresses (emails), Given Name (givenName), Surname (surname), and User's Object ID (objectId). The 'OK' button at the bottom of the right window is highlighted with a red arrow.

55. Click Create.

56. In the Settings blade, select Applications.

57. Select the created app.

58. Take note of the Application ID as you will need this for the next Task.

The screenshot shows the 'WebApp' settings blade. It has fields for 'Name' (set to 'WebApp') and 'Application ID' (set to '20ba8701-38ac-4a64-9eb1-f042e8b119f2'). A red arrow points to the 'Application ID' field.

59. In the Settings blade, select All Policies. You should see three policies and the B2C instance. Take note of the names for these policies with the prefix 'B2C' as these names will be used in the next task.

The screenshot shows two side-by-side browser windows. The left window is titled 'Settings' and has a sidebar with 'MANAGE' and 'POLICIES' sections. The 'All policies' link in the 'POLICIES' section is highlighted with a blue selection bar. The right window is titled 'All policies' and shows a list of three policies: 'B2C_1_profileedit' (Default template), 'B2C_1_signin' (Default template), and 'B2C_1_signup' (Default template). A search bar is at the top of the right window.

60. Your B2C directory is ready for use and after Task 2 is complete, you will be able to exercise it.

Task 2: Configure the web app Settings

In this task, you will update configuration settings in preparation for Azure AD B2C integration.

You will be guided through the instructions to find the information necessary to populate the configuration settings.

1. Within Visual Studio Solution Explorer, expand the Web folder, then expand the ContosoEvents.Web project, and open Web.config. You will update these appSettings in this file:

```
<add key="ida:Tenant" value="" />
<add key="ida:ClientId" value="" />
<add key="ida:SignUpPolicyId" value="" />
<add key="ida:SignInPolicyId" value="" />
<add key="ida:UserProfilePolicyId" value="" />
```

Azure Active Directory B2C:

1. For the ida:Tenant enter the domain of the Azure Active Directory B2C you created such as 'contosoeventsb2cSUFFIX.onmicrosoft.com'.
2. For the ida:ClientId enter the Application ID generated for your B2C application created in the previous task.
3. For the ida:SignUpPolicyId enter the name of the sign-up policy you created in the tenant (e.g., B2C_1_signup).

4. For the ida:SignInPolicyId enter the name of the sign in policy you created in the tenant (e.g., B2C_1_signin).
5. For the ida:UserProfilePolicyId enter the name of the profile editing policy you created in the tenant (e.g., B2C_1_profileedit).

Task 3: Add security features to the web application

In this task, you will enable security features that will leverage the configuration you just applied.

1. From Solution Explorer, expand the website folder ContosoEvents.Web.
2. Open Startup.cs in the root folder and uncomment the line of code below the TODO item. This will set up the authentication middleware so that users are redirected to authenticate to Azure AD B2C when they access protected resources.

```
//TODO Exercise 10 - Task 3  
ConfigureAuth(app);
```

3. Open FilterConfig.cs from the App_Start folder, and uncomment the line below the TODO item. This will provide default protection for all routes so that authentication is required unless the route allows anonymous callers.

```
//TODO Exercise 10 - Task 3  
filters.Add(new Policies.PolicyAuthorize { Policy = Startup.SignInPolicyId });
```

4. Open _Layout.cshtml under Views\Shared and remove the 3 lines following the TODO item, and uncomment area below it. This will prevent the menu items to be hidden unless the user is authenticated. It will look like this when you complete the change.

```
@*TODO Exercise 10 - Task 3 *@  
  
@if (Request.IsAuthenticated)  
{  
  
    <li><a href="@Url.Action("Index", "Orders")"><span class="glyphicon glyphicon-shopping-cart"></span> Orders</a></li>  
  
    <li><a href="@Url.Action("Index", "LoadTest")"><span class="glyphicon glyphicon-flash"></span> LoadTest</a></li>  
  
    <li><a href="@Url.Action("Status", "LoadTest")"><span class="glyphicon glyphicon-stats"></span> LoadTest Status</a></li>  
  
}
```

5. Open _LoginPartial.cshtml under Views\Shared and uncomment the area below the TODO item. This will enable the menus for user sign in and sign out.

```
@*TODO Exercise 10 - Task 3 *@  
  
@if (  
    Request.IsAuthenticated)  
{  
  
    <text>  
        <ul class="nav navbar-nav navbar-right">
```

```
<li>

    <a id="profile-link"><span class="glyphicon glyphicon-user"></span>
    @User.Identity.Name</a>

    <div id="profile-options" class="nav navbar-nav navbar-right">

        <ul class="profile-links">

            <li class="profile-link">

                <a href="@Url.Action("Profile", "Account")"><span
class="glyphicon glyphicon-pencil"></span> Edit Profile</a>

            </li>

        </ul>

    </div>

</li>

<li>

    <a href="@Url.Action("SignOut", "Account")"><span class="glyphicon
glyphicon-log-out"></span> Sign out</a>

</li>

</ul>

</text>

}

else

{

    <ul class="nav navbar-nav navbar-right">

        <li><a id="signUpLink" href="@Url.Action("SignUp", "Account")"><span
class="glyphicon glyphicon-user"></span> Sign up</a></li>

        <li><a id="loginLink" href="@Url.Action("SignIn", "Account")"><span
class="glyphicon glyphicon-log-in"></span> Sign in</a></li>

    </ul>

}
```

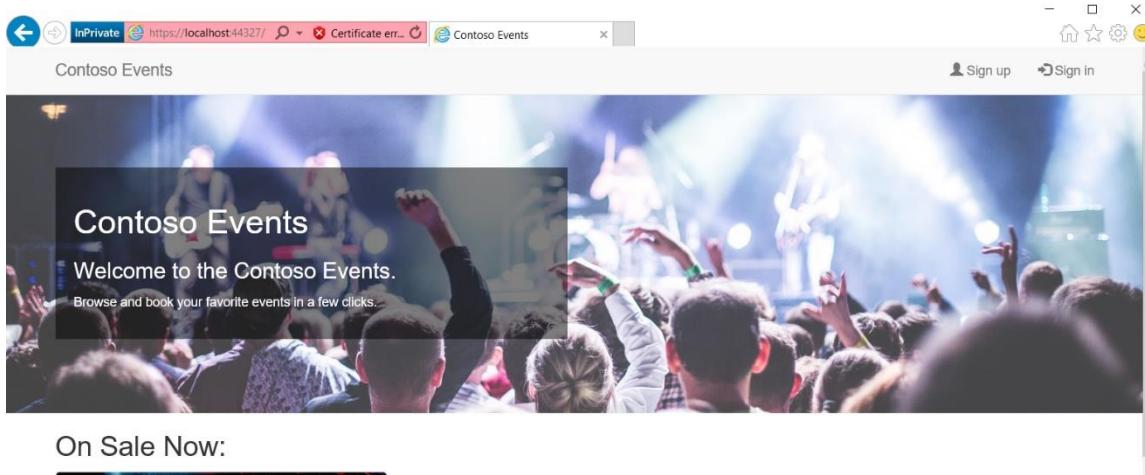
6. Compile the application and ensure no errors before the next step.

Task 4: Running the web app and signing in

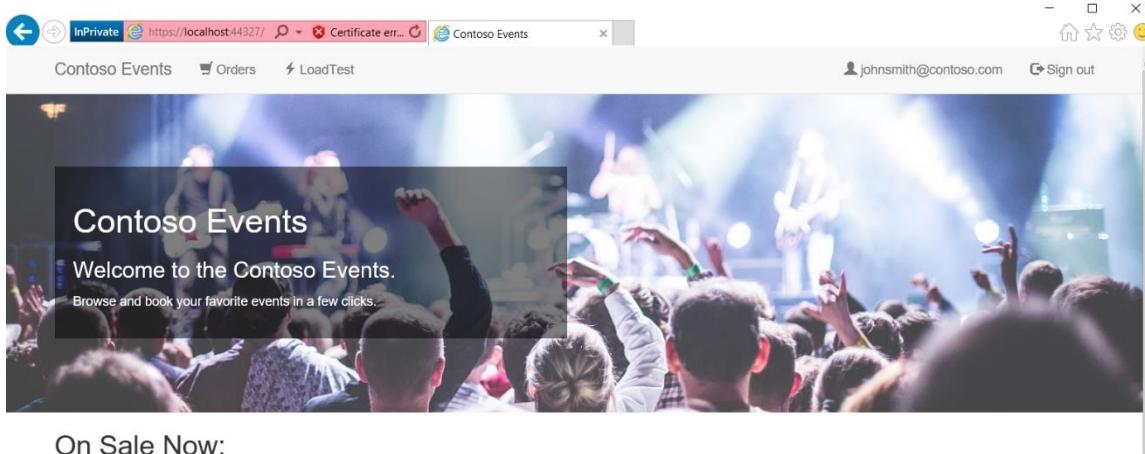
In this task, you will test the web application and register yourself as a user to log in to the Azure AD B2C tenant.

1. Using Solution Explorer in Visual Studio, open the Web folder.
2. Right click the ContosoEvents.Web project, select Debug, and then Start new instance.

3. When the application launches you will see the website home page



4. Click Sign up and complete a new user registration. After that, you will be returned to the website. Note that your username is shown in the menu bar.



5. You will be redirected back to the web app
6. You can now sign in, sign out, and register users to the website.

Note: You can optionally publish this application to Azure and test sign in, sign out continues to work from the deployed URL.

After the hands-on lab

Duration: 10 minutes

In this exercise, attendees will deprovision any Azure resources that were created in support of the lab.

Task 1: Delete the resource group

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

1. Using the Azure portal, navigate to the Resource group you used throughout this hands-on lab by selecting Resource groups in the left menu.
2. Search for the name of your research group, and select it from the list.
3. Select Delete in the command bar, and confirm the deletion by re-typing the Resource group name, and selecting Delete.
4. In the Management portal, from the left-hand menu, select Active Directory.



5. Select the tenant to delete, and delete it.