



# Microsoft Cloud Workshop

Microservices architecture

Developer 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

<b>Microservices architecture – developer edition hands-on lab step-by-step.....</b>	<b>1</b>
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) .....	11
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: Web App .....	32
Task 4: Function App .....	33
Task 5: Storage account.....	35
Task 6: Cosmos DB .....	37
Exercise 2: Implementing the Service Fabric solution.....	42
Task 1: Interacting with an Actor's State .....	42
Task 2: Interacting with a Stateful Service .....	44
Task 3: Interacting with an Actor from a Web API Controller .....	45
Task 4: Inspecting Service Partitions.....	46
Exercise 3: Placing ticket orders.....	47
Task 1: Run the solution .....	47
Task 2: Test the application .....	50
Task 3: Service Fabric Explorer.....	54
Task 4: Set up the Ticket Order Sync queue.....	55
Task 5: Set up the functions .....	61
Task 6: Test order data sync .....	73
Exercise 4: Publish the Service Fabric Application.....	77
Task 1: Publish the application .....	77
Task 2: Test an order from the cluster.....	79
Exercise 5: API Management.....	82
Task 1: Import API.....	82
Task 2: Retrieve the user subscription key.....	84
Task 3: Configure the Function App with the API Management key .....	85

Exercise 6: Configure and publish the web application.....	87
Task 1: Configure the web app settings .....	87
Task 2: Running the web app and creating an order.....	88
Task 3: Publish the web app.....	92
Exercise 7: Upgrading .....	95
Task 1: How upgrade works .....	95
Task 2: Update an actor state .....	96
Task 3: Perform a smooth upgrade.....	97
Task 4: Submit a new order .....	101
After the hands-on lab .....	104
Task 1: Delete the resource group .....	104

# Microservices architecture – developer 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 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.

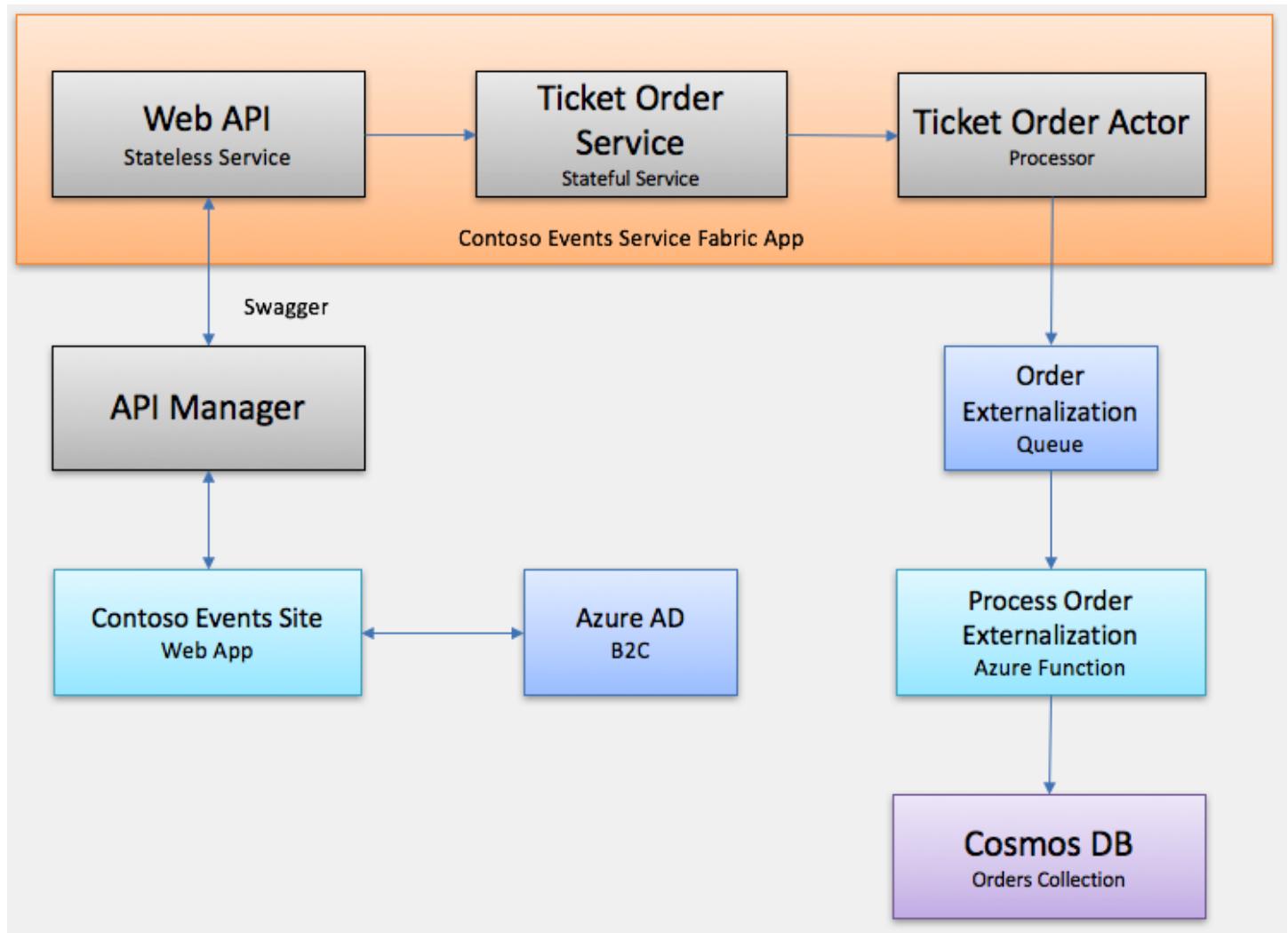
Attendees will learn how to:

- 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

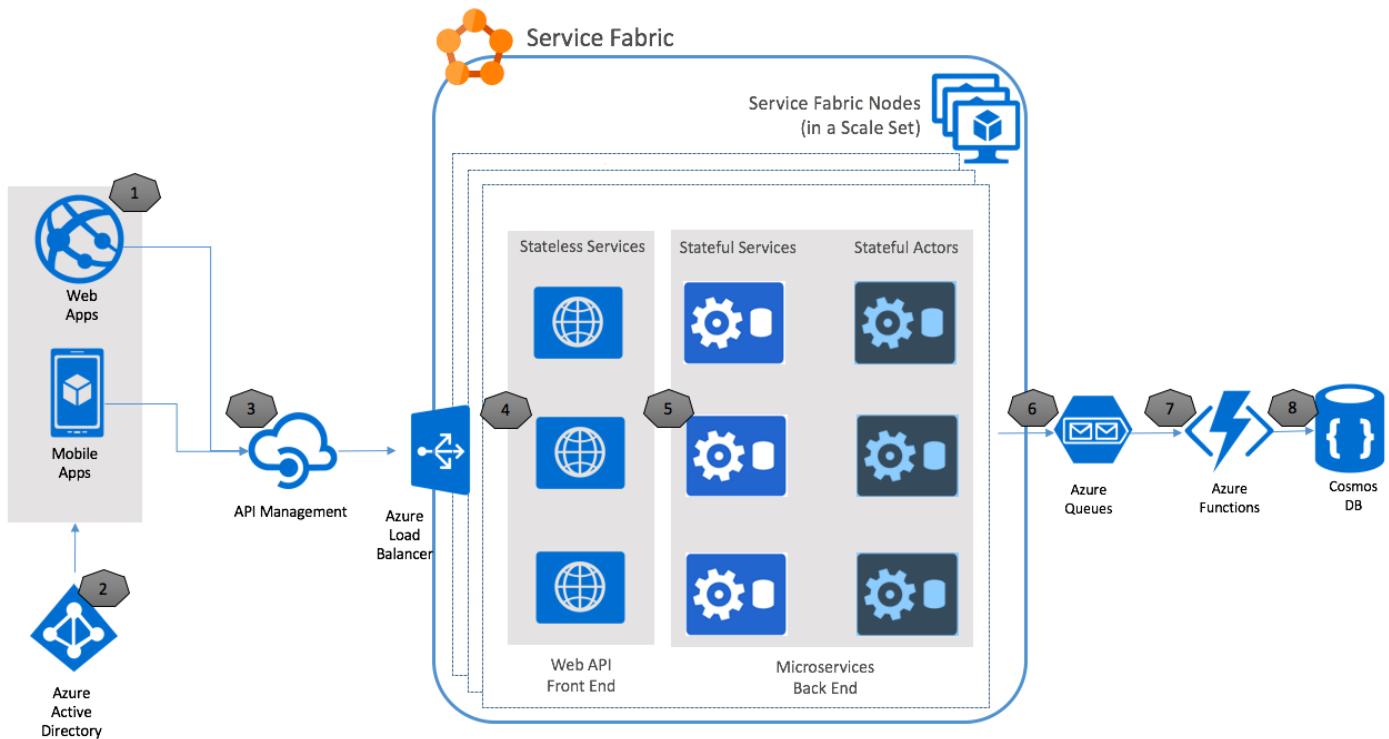
Contoso Events is an online service for concerts, sporting and other event ticket sales. They are redesigning their solution for scale with a microservices strategy and want to implement a POC for the path that receives the most traffic: ticket ordering.

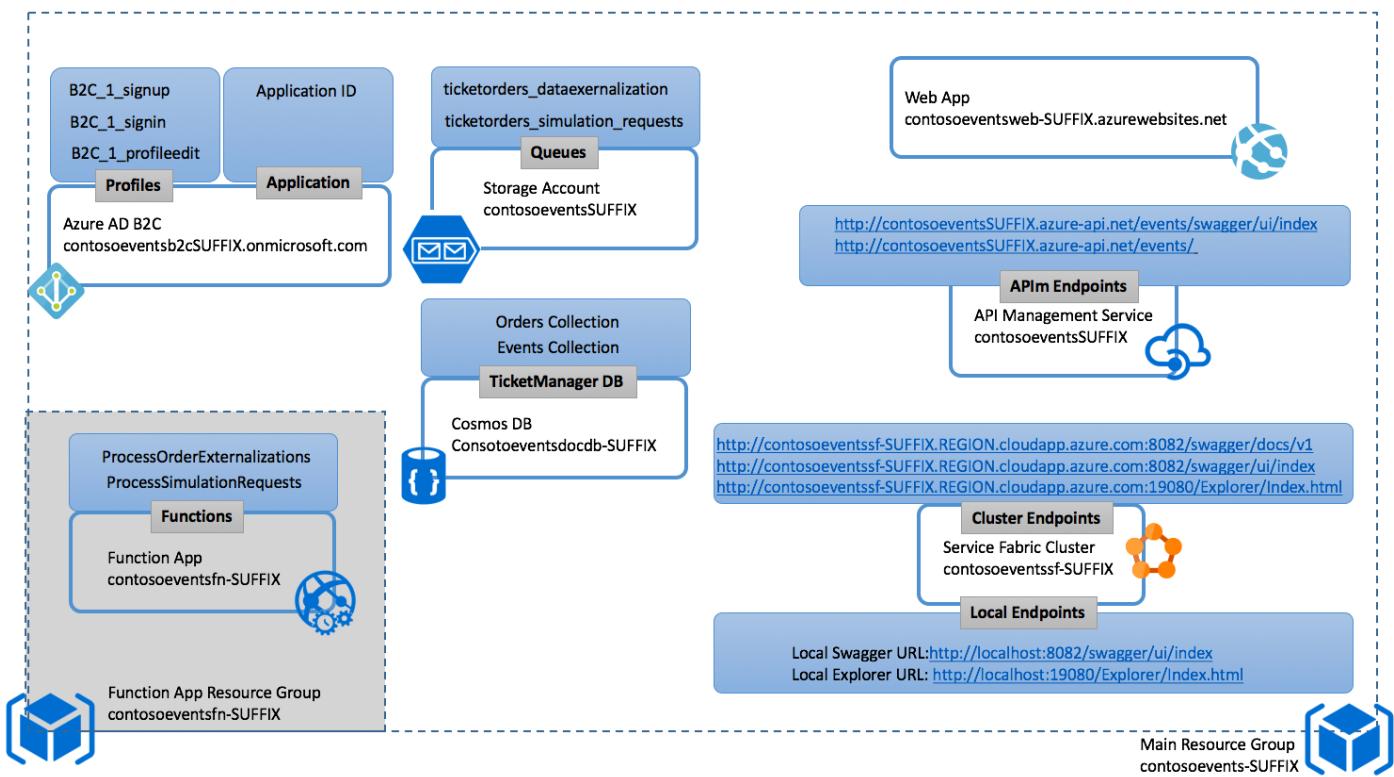
In this hands-on lab, you will construct an end-to-end POC for ticket ordering. You will leverage Service Fabric, API Management, Function Apps, Web Apps, and Cosmos DB.



## 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.

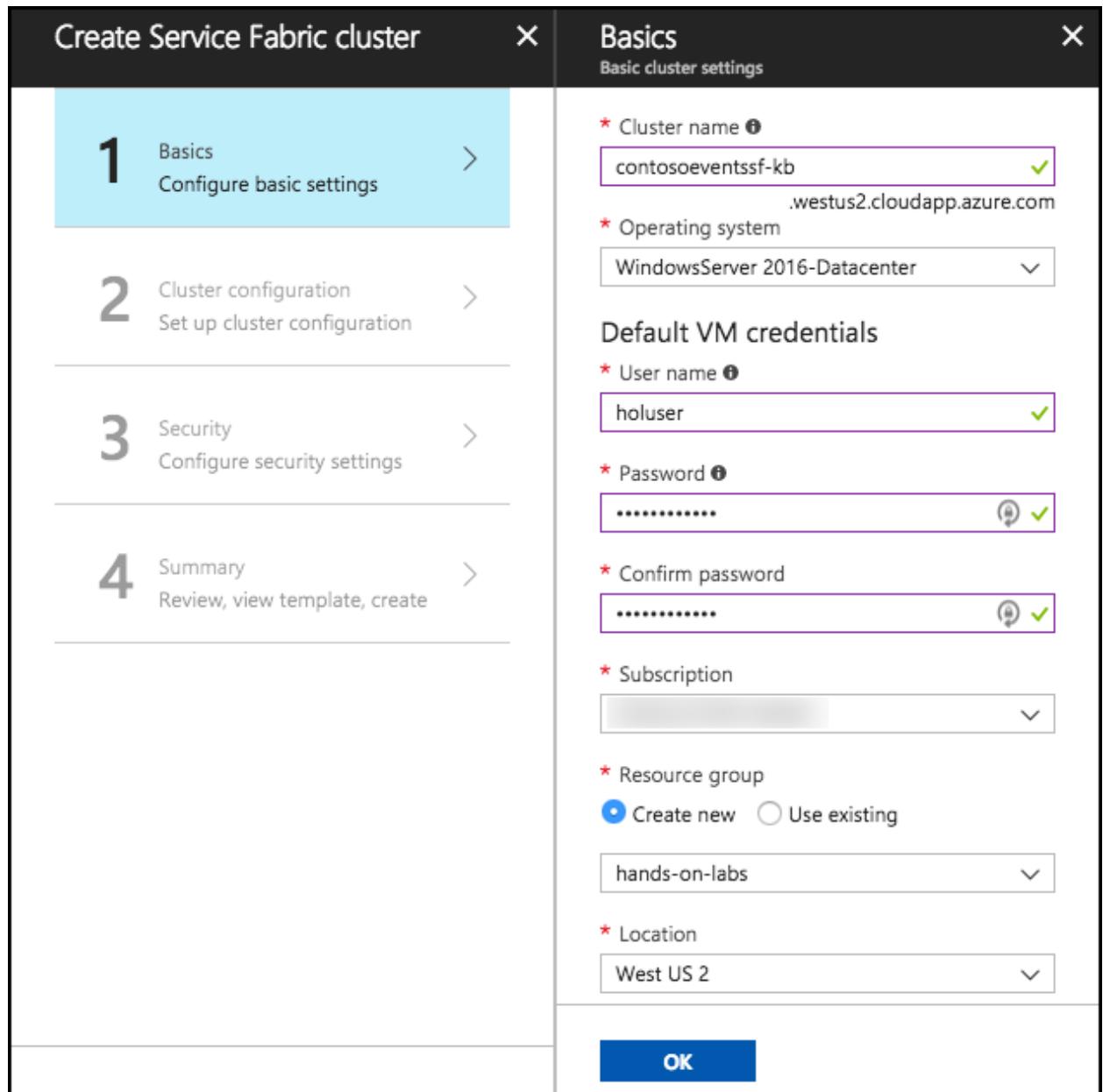
1. In the Azure Portal, select +New, then type "Service Fabric" into the Search the Marketplace box. Select Service Fabric Cluster from the results.

The screenshot shows the Azure Marketplace search interface. On the left is a sidebar with a 'New' button highlighted by a red box. The main area has a search bar containing 'Service Fabric' with a magnifying glass icon. Below the search bar is a 'Results' section. A table lists two items: 'Service Fabric Cluster' (selected and highlighted with a red box) and 'Service Fabric Analytics'. The columns in the table are NAME, PUBLISHER, and CATEGORY.

NAME	PUBLISHER	CATEGORY
Service Fabric Cluster	Microsoft	Compute
Service Fabric Analytics	Microsoft	Monitoring + Manage...

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.



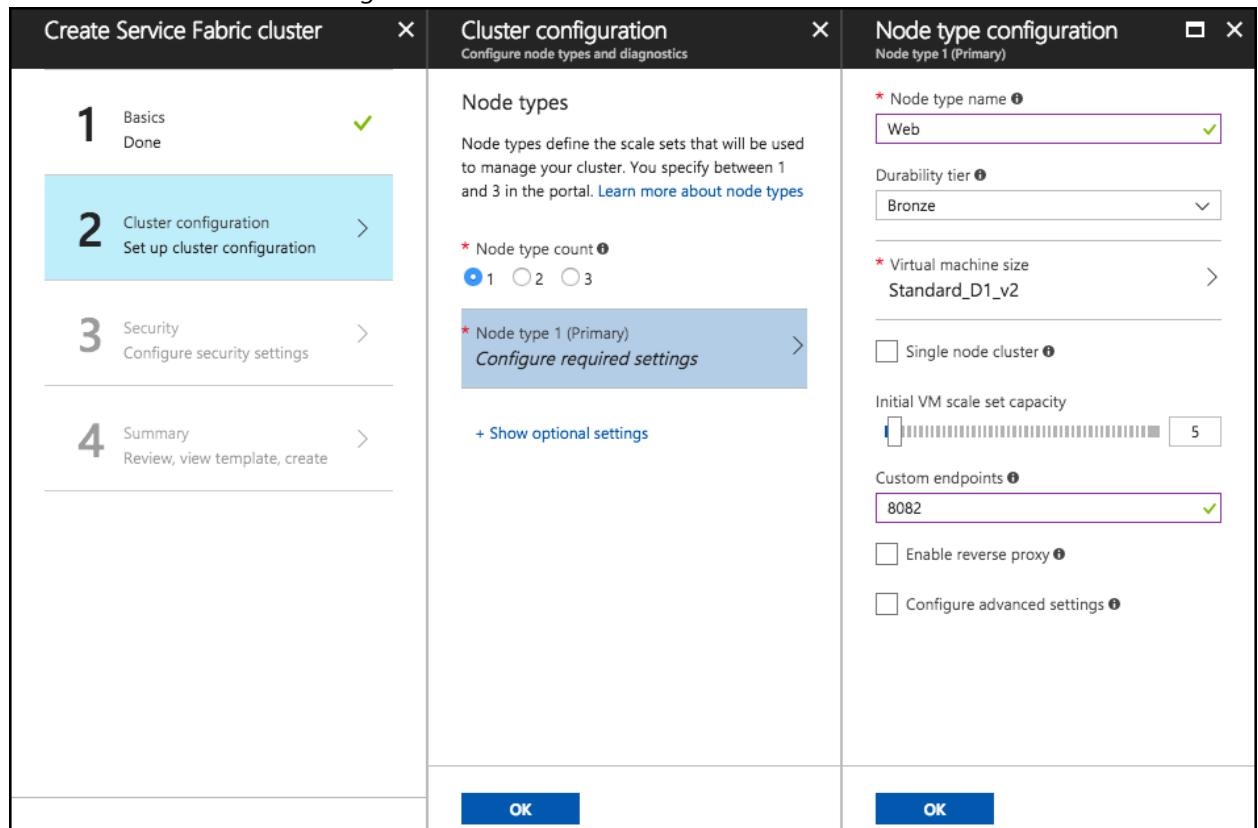
4. On the Cluster configuration blade, set the following:
  - a. Node type count: Select 1
  - b. Node type 1 (Primary): Select to configure required settings. On the Node type configuration blade enter:
    - i. Node type name: Enter Web
    - ii. Durability tier: Leave Bronze selected

- 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
<b>91.51</b> USD/MONTH (ESTIMATED)	<b>183.02</b> USD/MONTH (ESTIMATED)	<b>365.30</b> USD/MONTH (ESTIMATED)

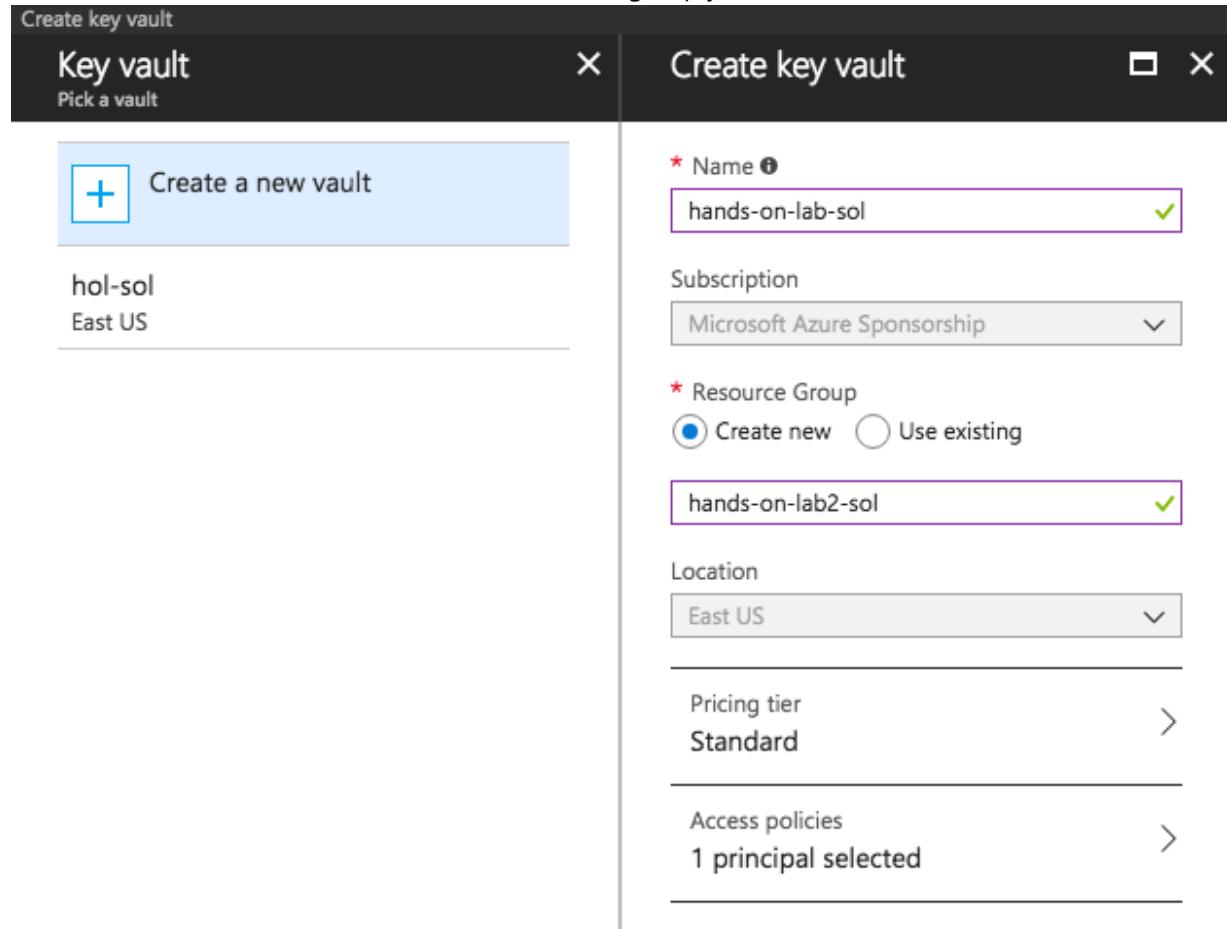
- iv. Single node cluster: Leave unchecked  
v. Initial VM scale set capacity: Leave set to 5  
vi. Custom endpoints: Enter 8082. This will allow the Web API to be accessible through the cluster.  
vii. Enable reverse proxy: Leave unchecked  
viii. Configure advanced settings: Leave unchecked  
ix. Select 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".
  - On the "Create key vault" configuration blade enter:
    - Name: hands-on-lab-SUFFIX
    - 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:
- Click "Edit access policies for hands-on-lab-SUFFIX"
  - In the Access policies configuration blade, click the link "Click to show advanced access policies"
  - Check the "Enable access to Azure Virtual Machines for deployment" checkbox.

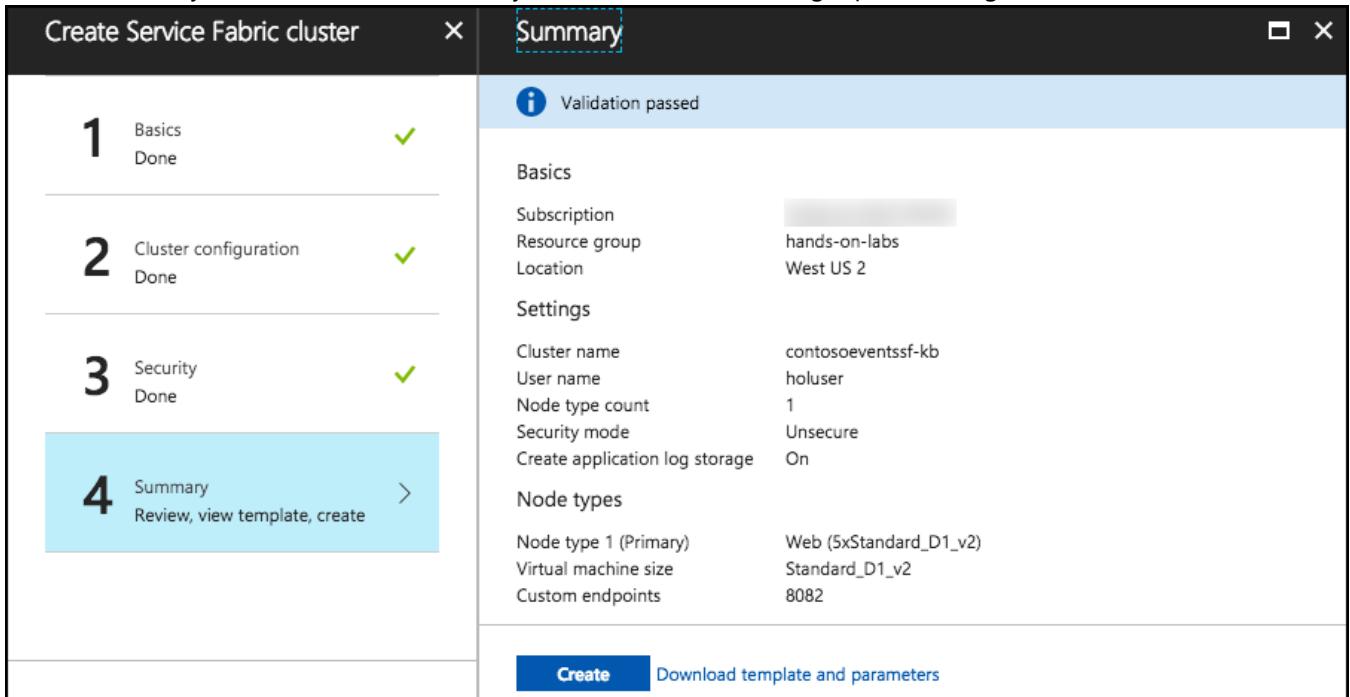
- iv. Click "Save". When the key vault update completes, close the Access policies blade.

The screenshot shows two overlapping windows. The main window is titled 'Security' with the sub-header 'Configure cluster security settings'. It has a 'Configuration Type' section with a radio button selected for 'Basic'. Below it is a note about certificate creation for basic mode. A 'Key vault' section shows 'hands-on-lab-sol' with an edit icon. A warning message box is overlaid on the right, stating that the selected key vault is not enabled for deployment and providing instructions to enable it via access policies. At the bottom is a blue button labeled 'Edit access policies for hands-on-lab-sol'. The second window, titled 'Access policies', has tabs for 'Save', 'Discard', and 'Refresh'. It lists several policy options with checkboxes: 'Enable access to Azure Virtual Machines for deployment' (checked), 'Enable access to Azure Resource Manager for template deployment' (unchecked), and 'Enable access to Azure Disk Encryption for volume encryption' (unchecked). There are also 'Add new' and '...' buttons.

- f. Enter "hands-on-lab-sol" as the certificate name. Then click Ok on the Security configuration blade.

The screenshot shows the 'Security' configuration blade. It has a 'Configuration Type' section with 'Basic' selected. A note about certificate creation is present. Below is a 'Key vault' section with 'hands-on-lab-sol'. Underneath is a 'Certificate name' section with 'hands-on-lab-sol' entered, indicated by a green checkmark. The blade has standard 'Save', 'Discard', and 'X' close buttons at the top right.

6. On the Summary blade, review the summary, and select Create to begin provisioning the new cluster.



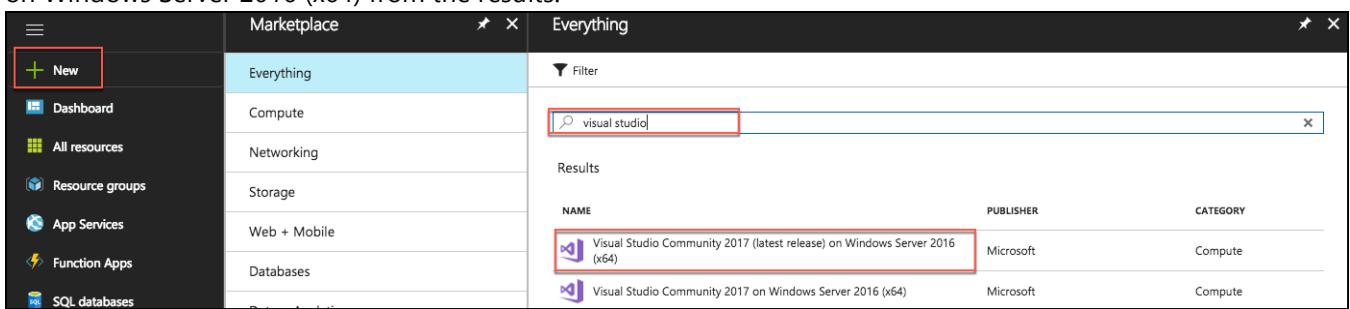
7. It can take up to 30 minutes or more to provision your Service Fabric Cluster. You can move on to the next task while you wait.

**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.

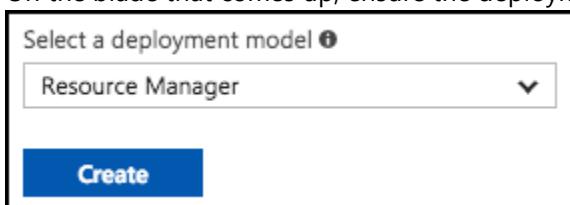
## Task 2: Provision a lab virtual machine (VM)

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

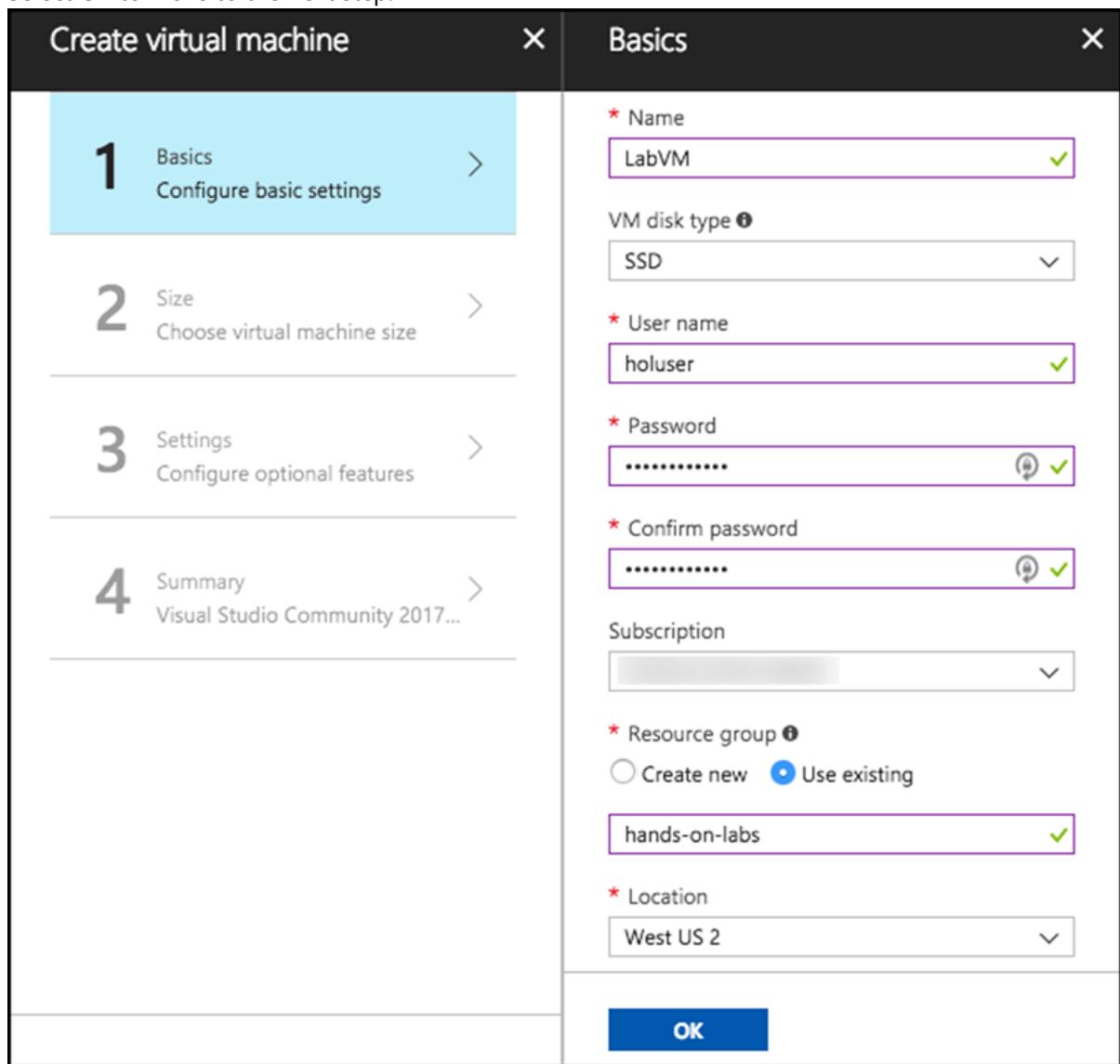
1. Launch a web browser, and navigate to the [Azure Portal](#).
2. Select +New, then type "Visual Studio" into the search bar. Select Visual Studio Community 2017 (latest release) on Windows Server 2016 (x64) from the results.



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



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
<span style="color: blue;">★ Recommended</span>   <span style="border: 2px solid red; padding: 2px;">View all</span>		
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
Premium disk support	Premium disk support	Premium disk support
Load balancing	Load balancing	Load balancing
<b>142.85</b> USD/MONTH (ESTIMATED)	<b>285.70</b> USD/MONTH (ESTIMATED)	<b>571.39</b> USD/MONTH (ESTIMATED)
<span style="background-color: #0072BC; color: white; padding: 5px 20px; border-radius: 5px;">Select</span>		

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 links to 'Pricing for other VM sizes', 'Terms of use', and 'privacy policy'. A note about Azure resources states that users can use monetary commitment funds or subscription credits. 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 box. 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 'Resource groups' blade in the Azure portal. On the left sidebar, 'Resource groups' is selected and highlighted with a red box. In the main area, there is a search bar with 'hands-on' typed into it, also highlighted with a red box. Below the search bar, it says '1 items'. A table lists one item: 'NAME' (hands-on-labs), 'TYPE' (Virtual machine), and 'LOCATION' (West US 2). The entire row for 'hands-on-labs' is highlighted with a red box.

- 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

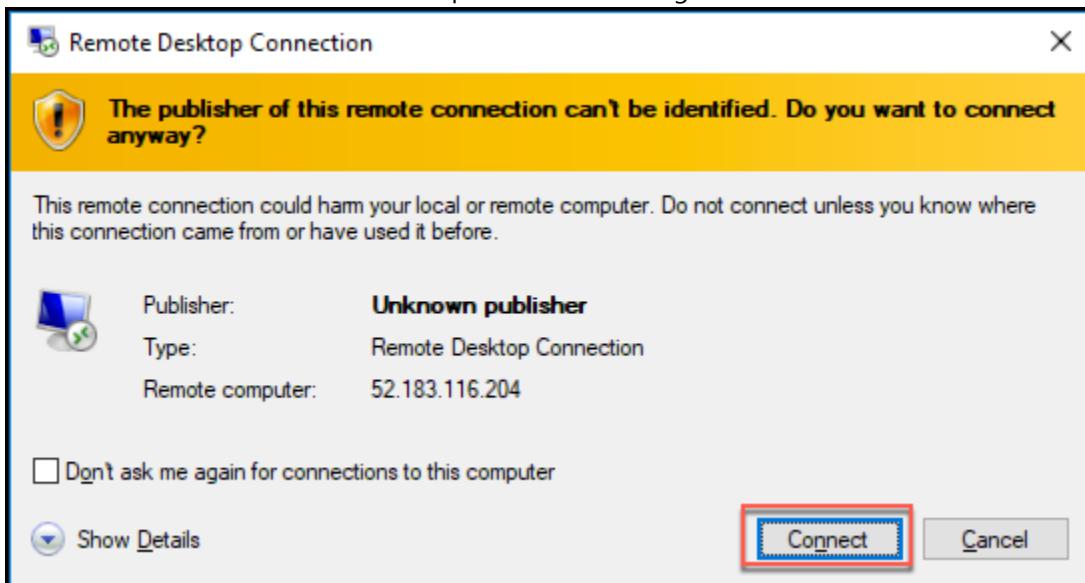
- On your Lab VM blade, select Connect from the top menu.

The screenshot shows the 'Lab VM' blade for the 'LabVM' virtual machine. At the top, there is a toolbar with several buttons: 'Connect' (highlighted with a red box), 'Start', 'Restart', 'Stop', 'Capture', 'Move', 'Delete', and 'Refresh'. Below the toolbar, there is a summary section with the following details:

- Resource group ([change](#)): hands-on-labs
- Status: Running
- Location: West US 2
- Computer name: LabVM
- Operating system: Windows
- Size: Standard D2s v3 (2 vcpus, 8 GB memory)

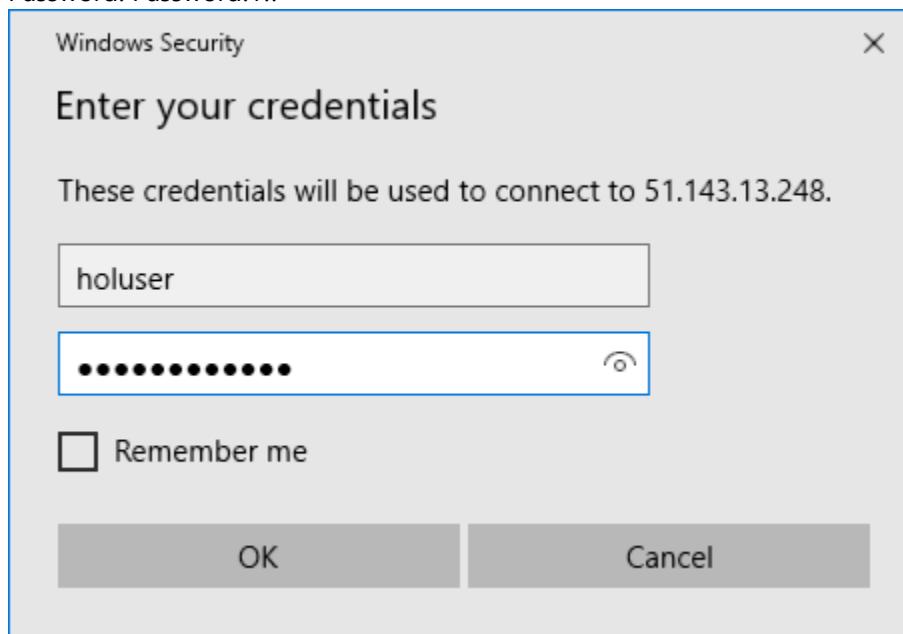
- Download and open the RDP file.

6. Select Connect on the Remote Desktop Connection dialog.

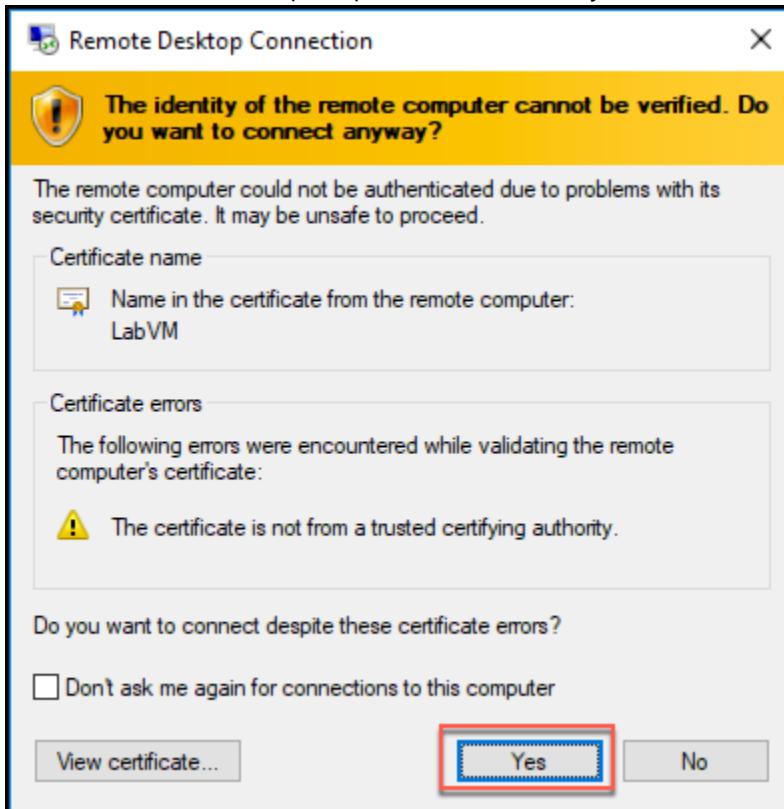


7. Enter the following credentials (or the non-default credentials if you changed them):

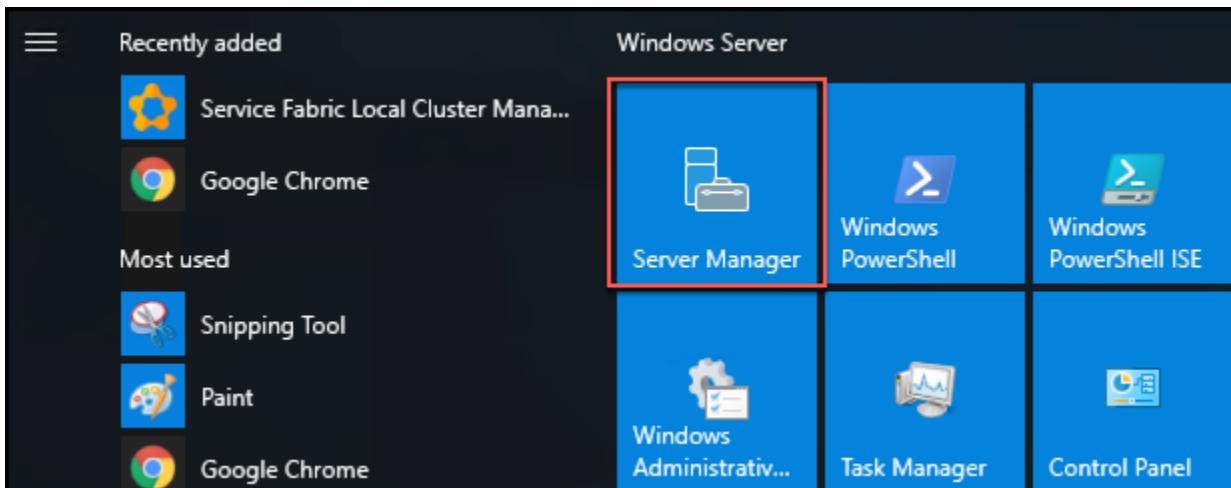
- User name: holuser
- Password: Password.1!!



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



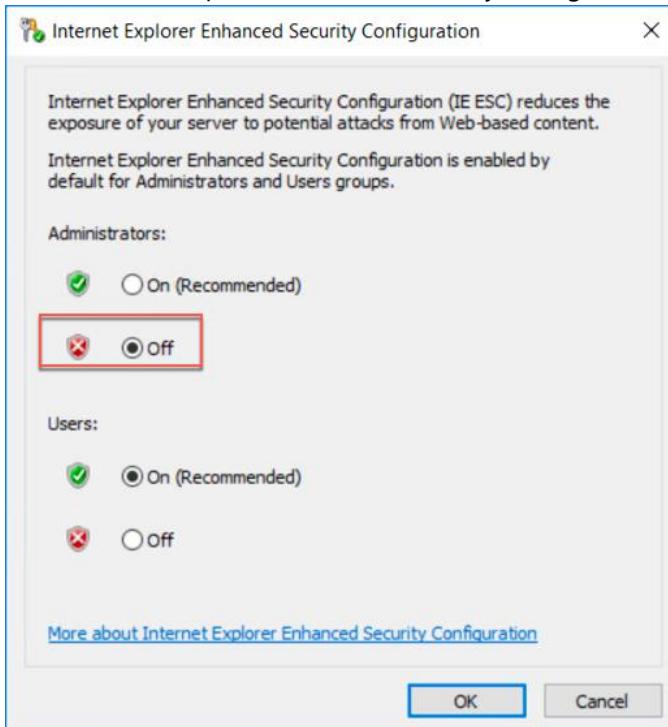
9. Once logged in, launch the Server Manager. This should start automatically, but you can access it via the Start menu if it does not start.



10. Select Local Server, then 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

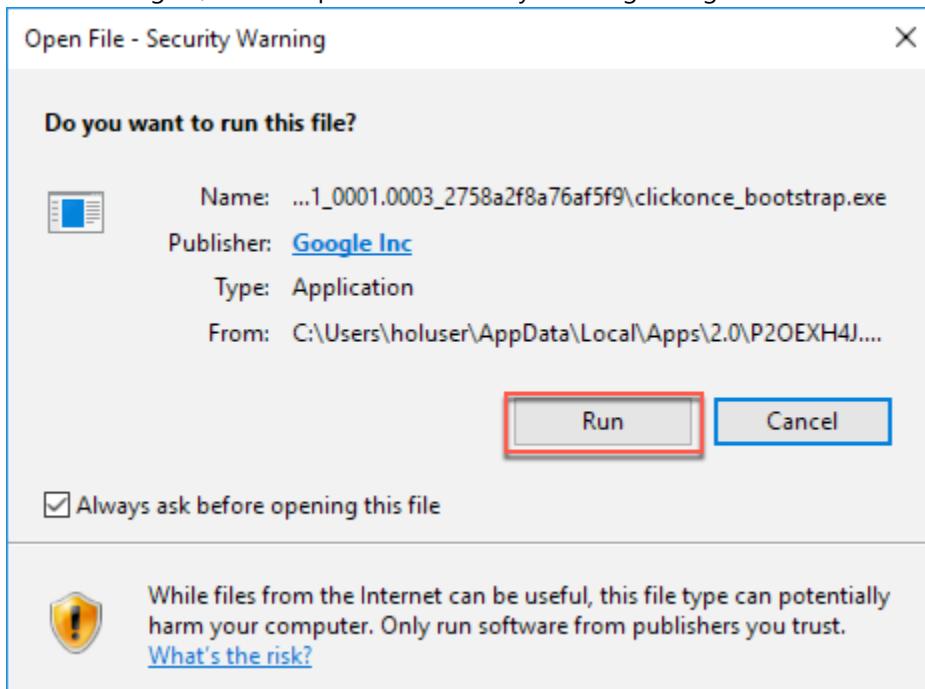
2. Select Accept and Install on the terms of service screen.

The screenshot shows a window titled "Download Chrome for Windows" for Windows 10/8.1/8/7 64-bit. It displays the "Google Chrome Terms of Service". The text states: "These Terms of Service apply to the executable code version of Google Chrome. Source code for Google Chrome is available free of charge under open source software license agreements at <http://code.google.com/chromium/terms.html>". Below this, section 1.1 is expanded, showing "1. Your relationship with Google" and "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". A "Printer-friendly version" link is also present. At the bottom left is a checked checkbox for "Help make Google Chrome better by automatically sending usage statistics and crash reports to Google." with a "Learn more" link. A large blue button labeled "ACCEPT AND INSTALL" is at the bottom.

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

The screenshot shows a "Application Run - Security Warning" dialog. It asks "Do you want to run this application?". The application name is "Google Installer" from "dl.google.com" published by "Google Inc". There are two buttons at the bottom: "Run" (highlighted with a red box) and "Don't Run". A warning message at the bottom states: "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...](#)" with a shield icon.

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.

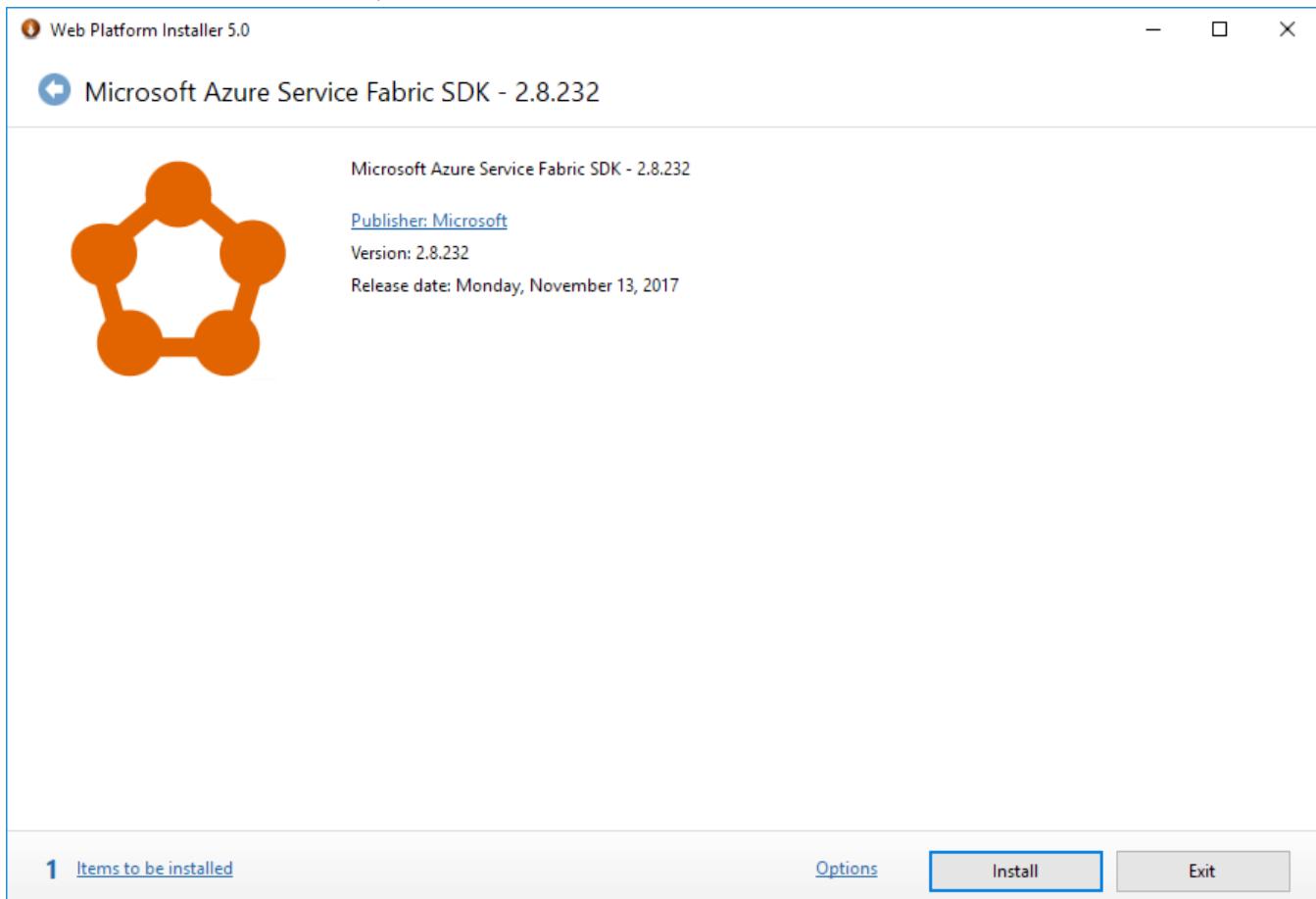
### Install the SDK and tools

#### To use Visual Studio 2017

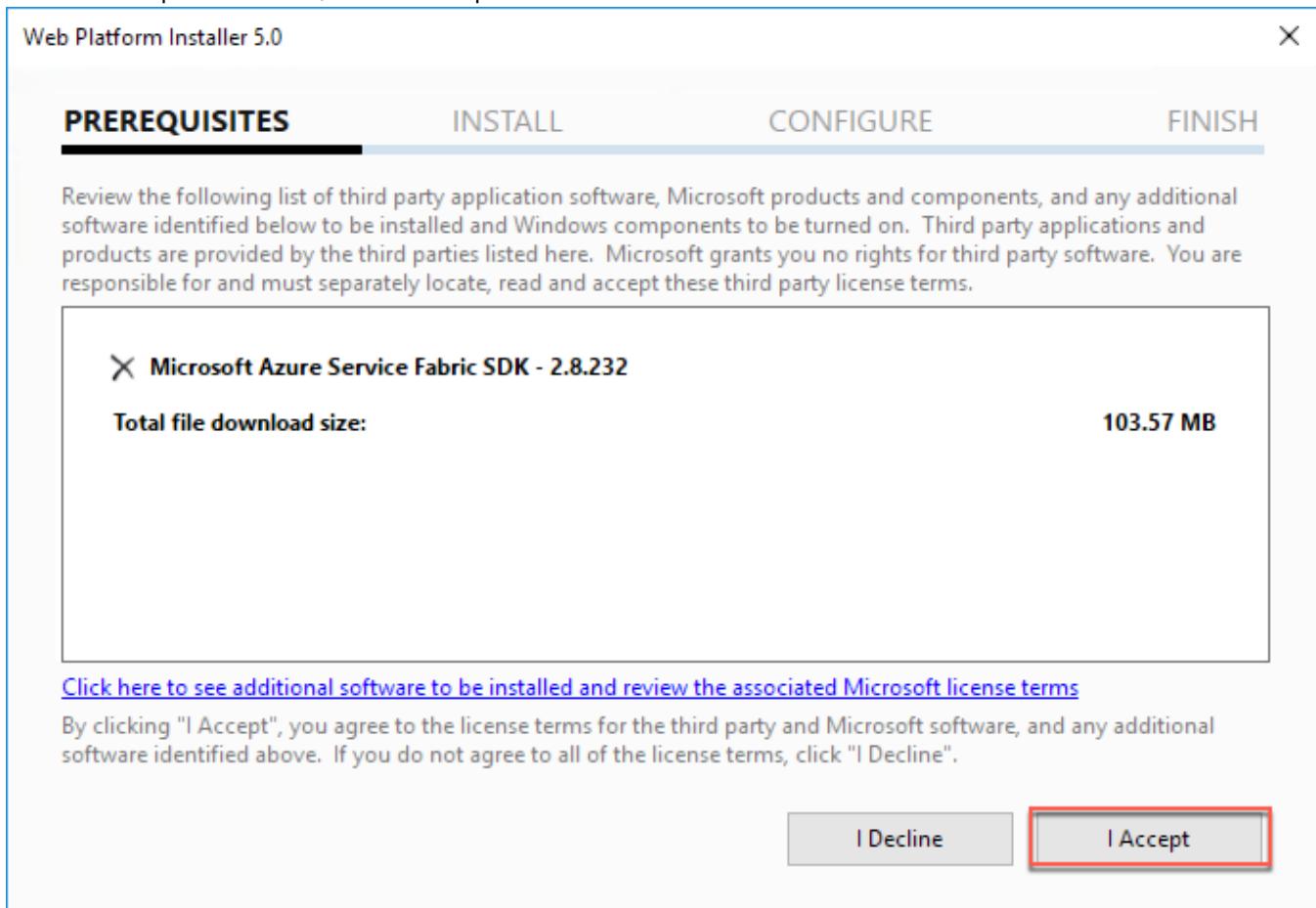
Service Fabric Tools are part of the Azure Development workload in Visual Studio 2017. Enable this workload as part of your Visual Studio installation. In addition, you need to install the Microsoft Azure Service Fabric SDK, using Web Platform Installer.

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

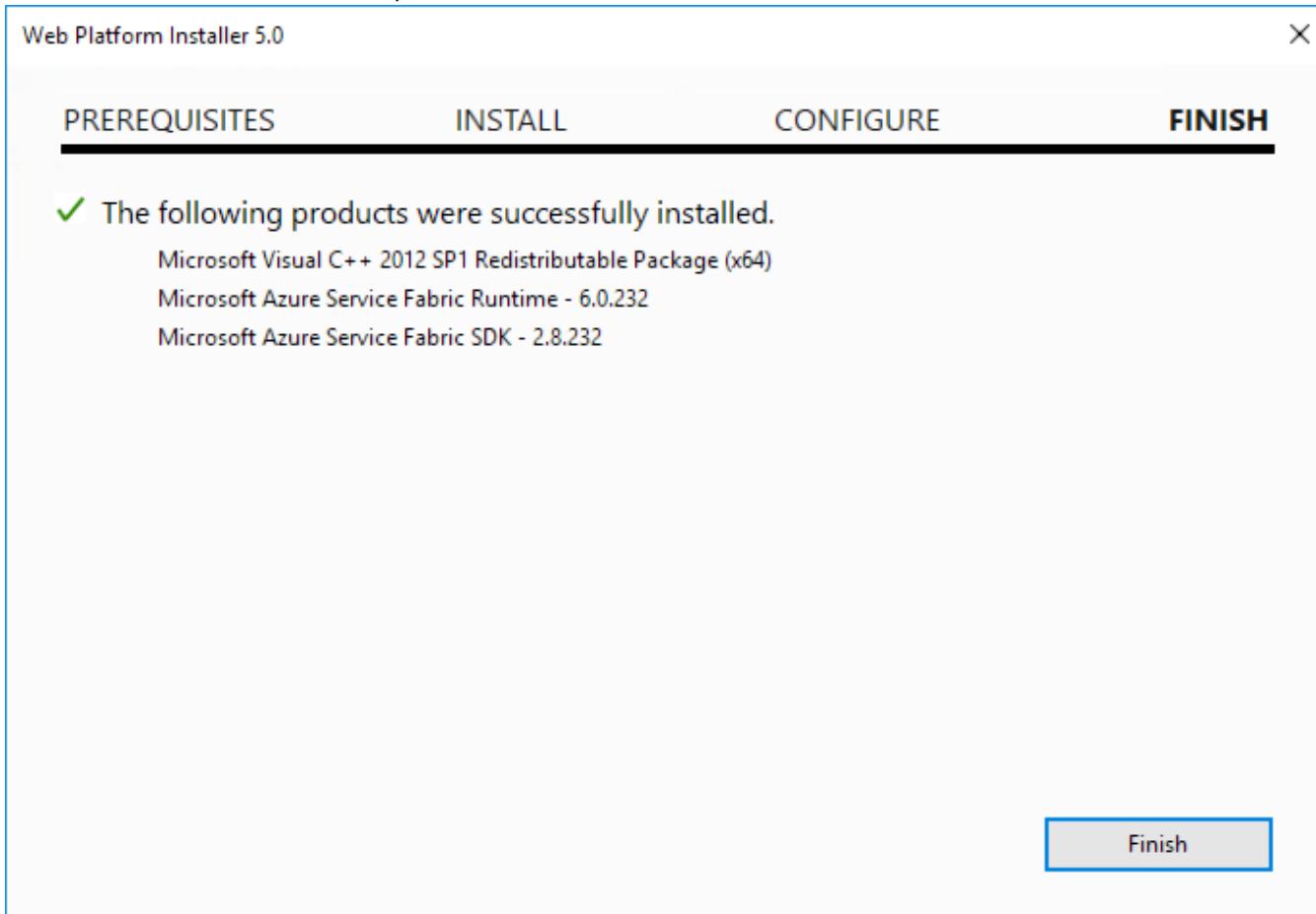
- Run the downloaded executable, and select Install in the Web Platform Installer screen.



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.

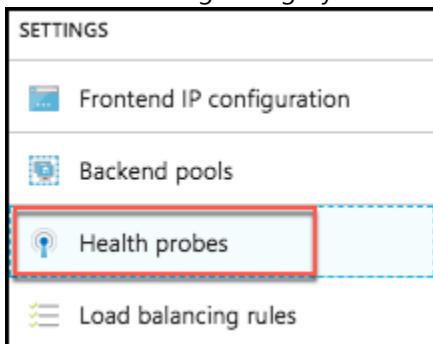
In this task, you will validate that the ports are open and if not, fix the issue.

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

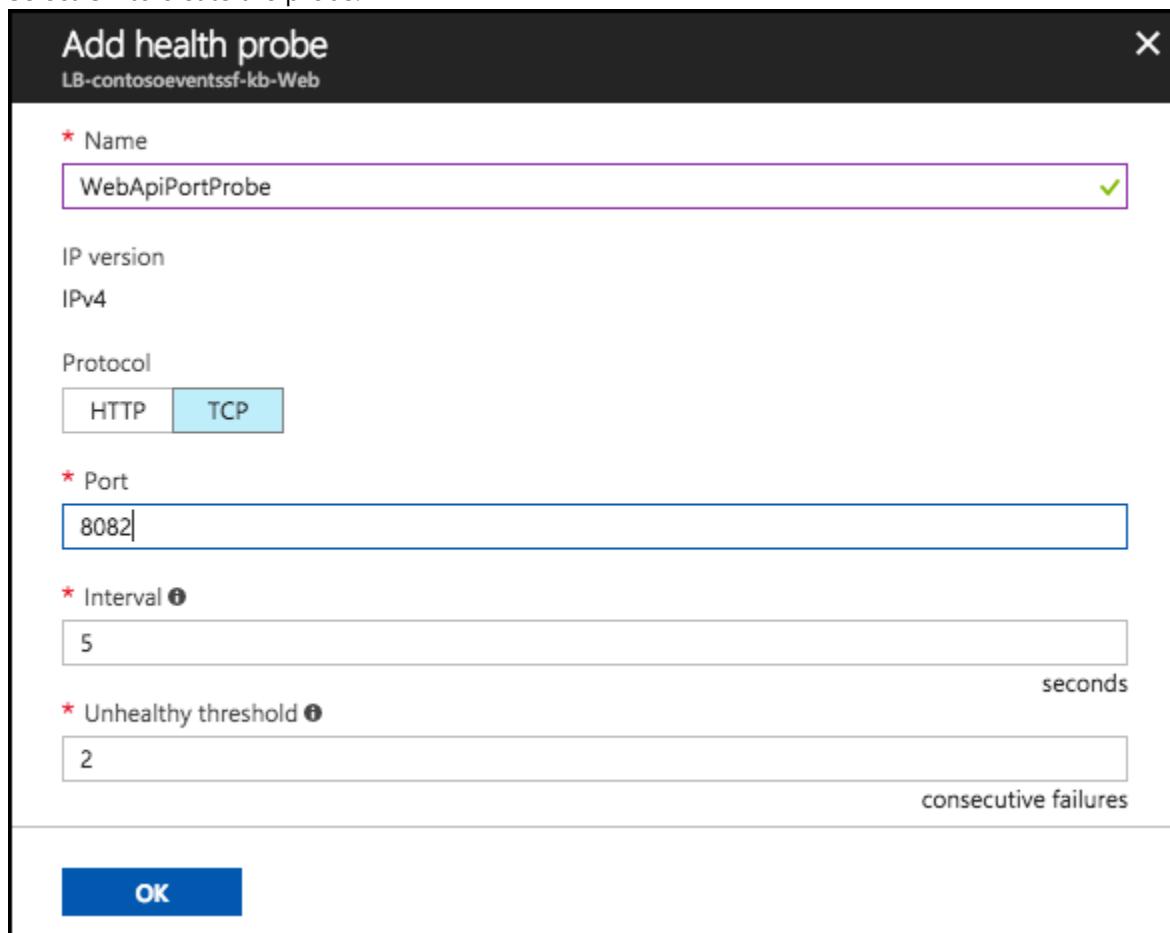
5. Select +Add on the Health Probes blade.



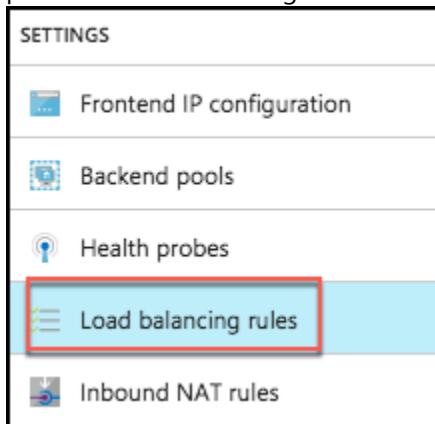
6. On the Add health probe blade, enter the following:

- 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**

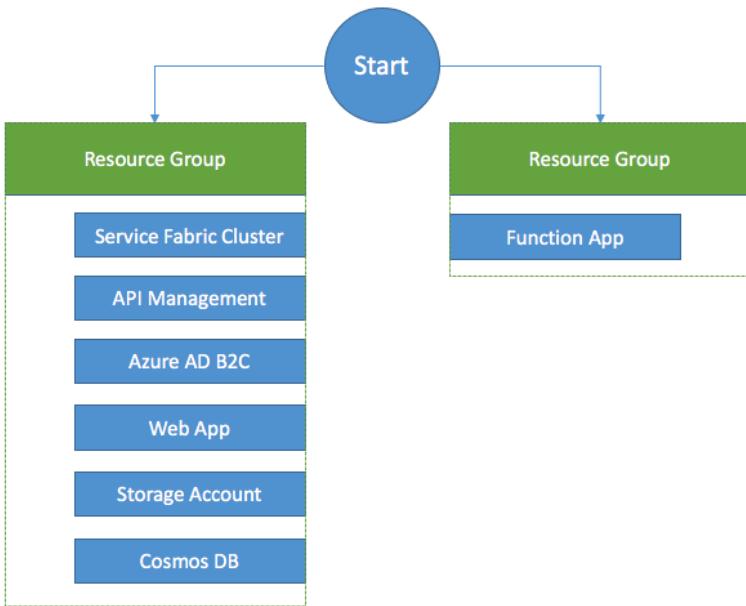
10. If you get an error notification such as "Failure to create probe", ignore this, but just go check that the probe indeed exists. It should. You now have a cluster ready to deploy to, and expose 8082 as the Web API endpoint / port.

## 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.

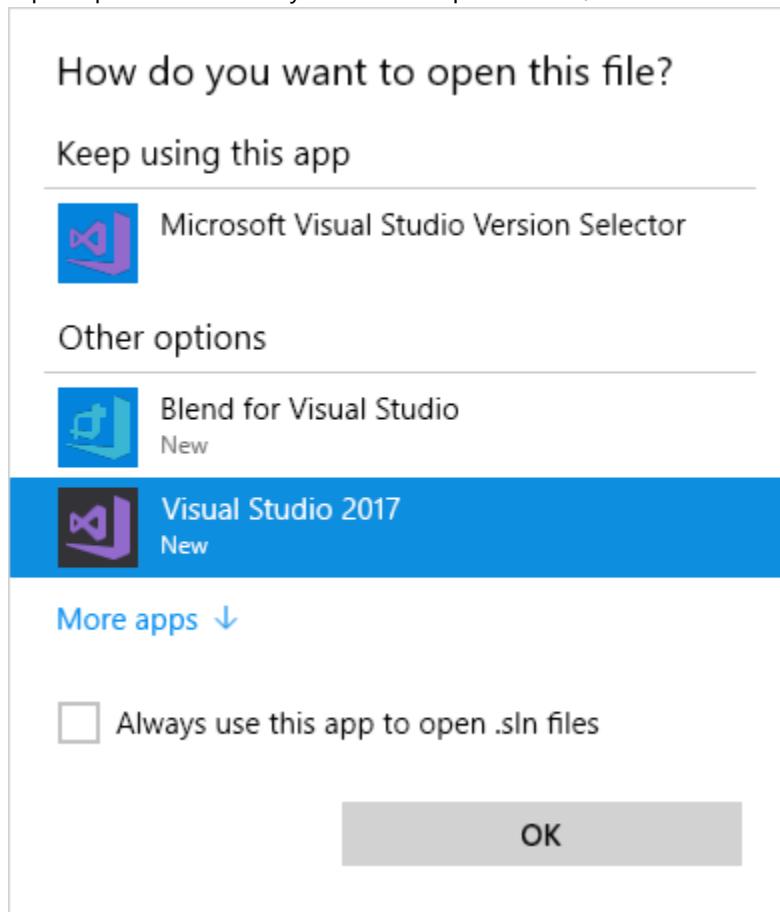
Because this is a “born in Azure” solution, it depends on many Azure resources. You will be guided through creating those resources before you work with the solution in earnest. The following figure illustrates the resource groups and resources you will create in this exercise.



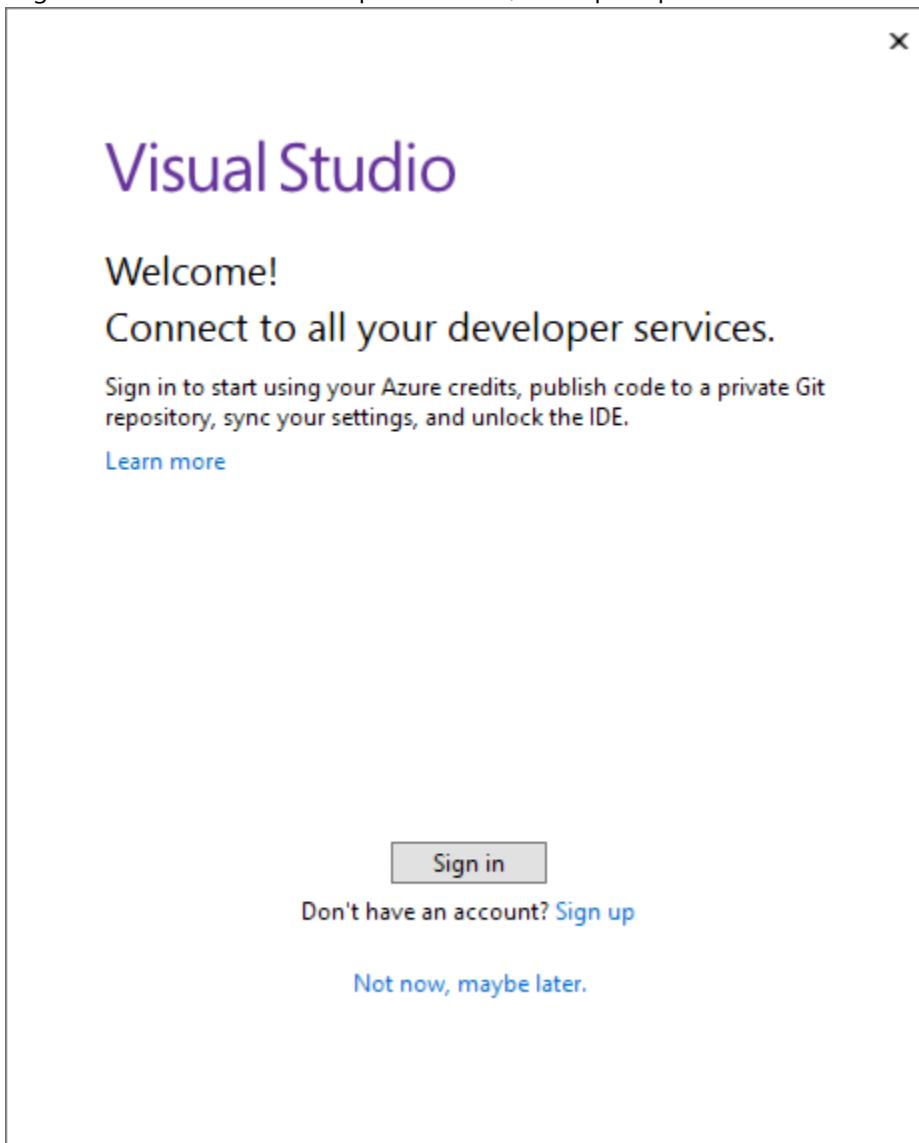
### Task 1: Download and open the ContosoEventsPoC starter solution

1. On your Lab VM, download the starter project from <http://bit.ly/2eQprpa>. (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.

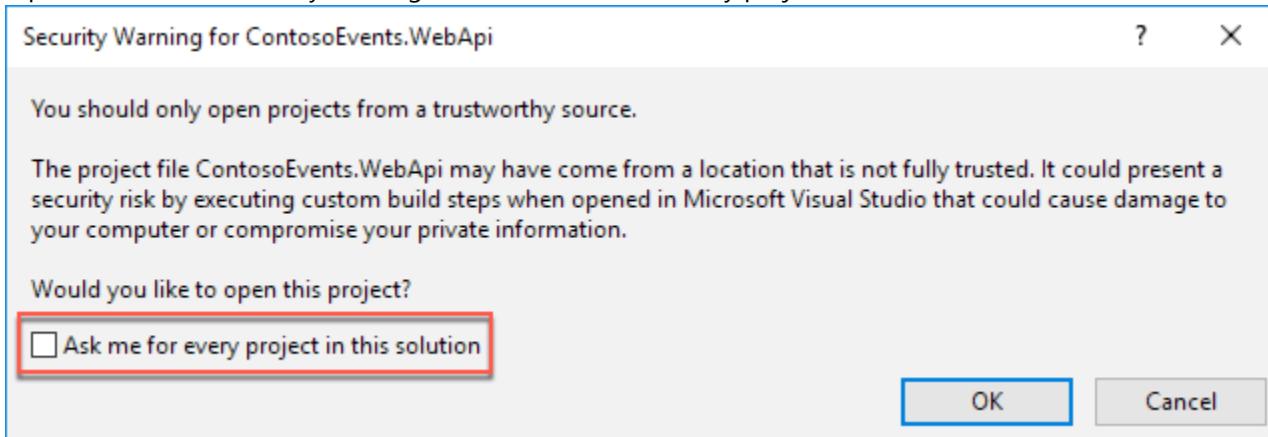
4. If prompted about how you want to open the file, select Visual Studio 2017, and select OK.



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

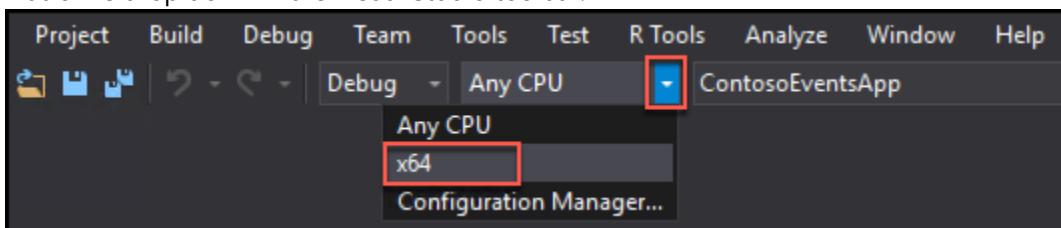


6. If presented with a security warning, uncheck Ask me for every project in this solution, and select OK.

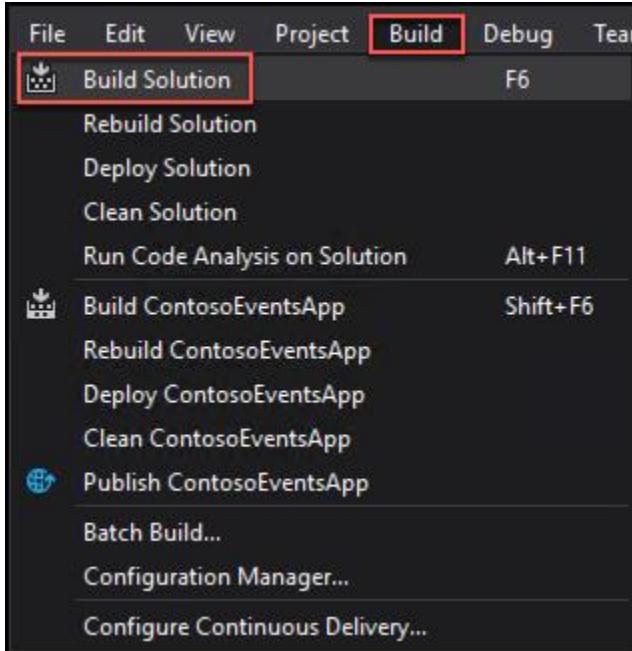


7. If you are missing any prerequisites (listed under [Requirements](#) above), you may be prompted to install these at this point.

- 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.

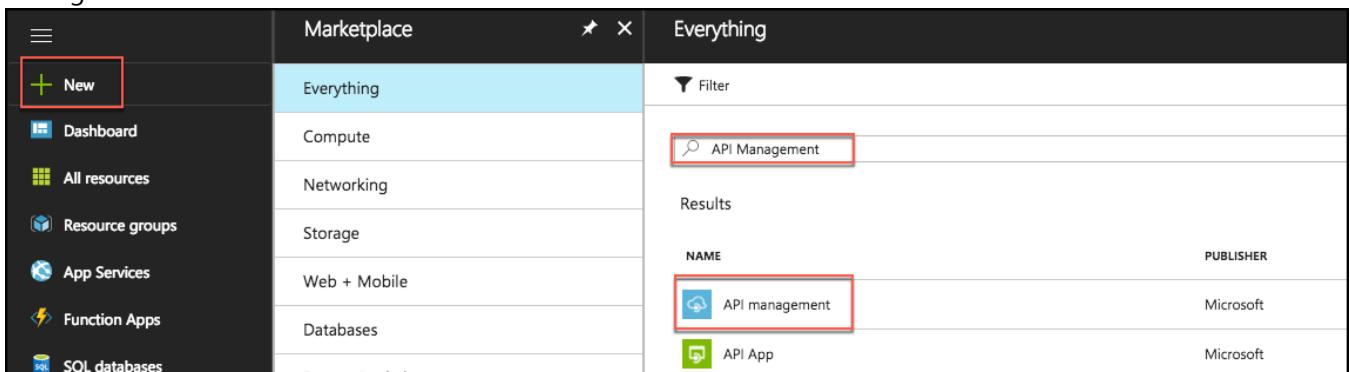


- You will have some compile-time errors at this point. These are expected, and will be fixed as you proceed with the hands-on lab.

## 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

**API Management service**

\* Name  
contosoevents-kb .azure-api.net

\* Subscription

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

\* Location  
West US 2

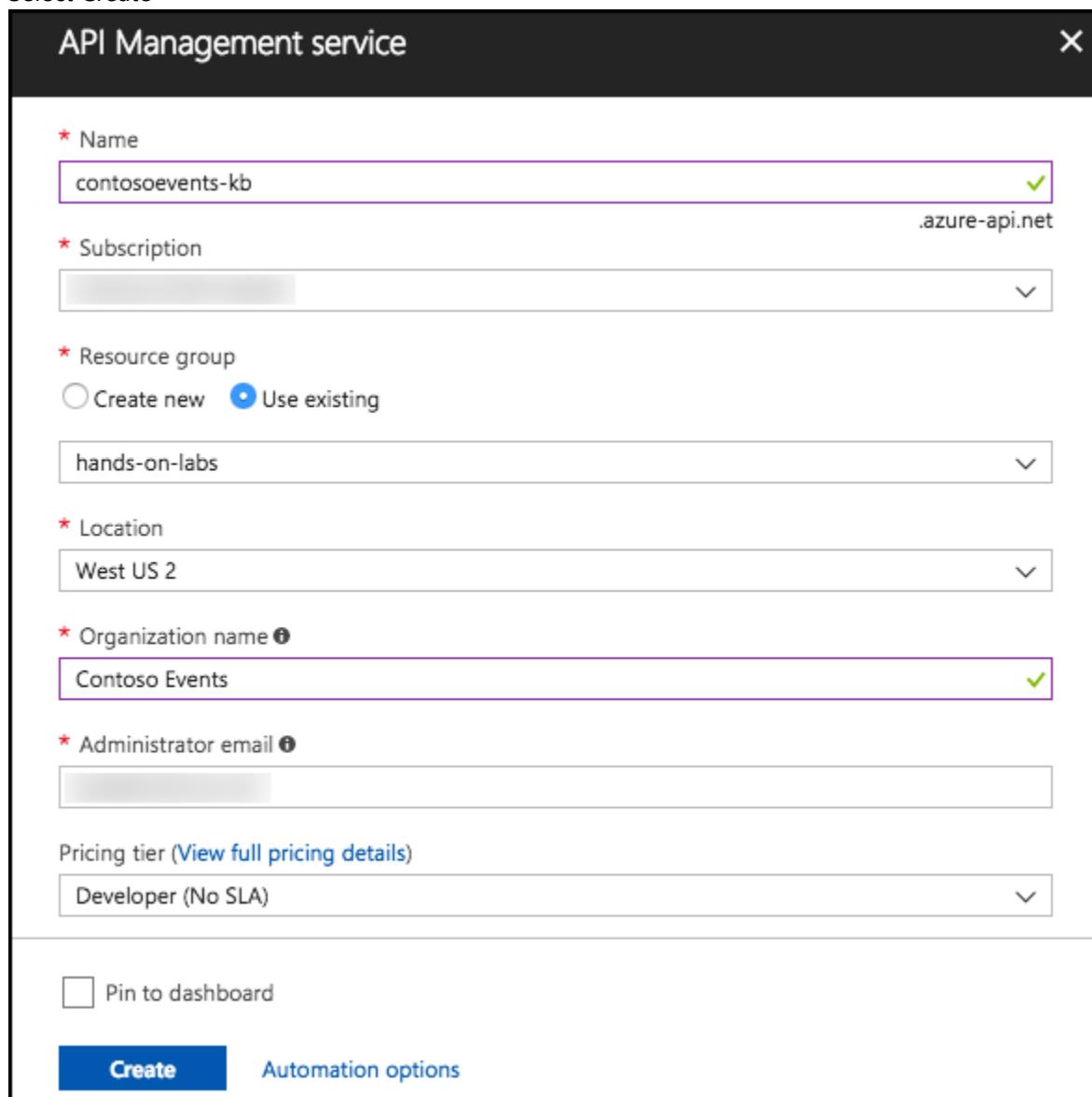
\* Organization name ⓘ  
Contoso Events

\* 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: 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.

The screenshot shows the Azure Portal's 'New' blade. On the left, there's a sidebar with various service icons like Dashboard, All resources, Resource groups, App Services, Function Apps, SQL databases, and Azure Cosmos DB. A red box highlights the '+ New' button. The main area has a search bar at the top labeled 'Search the Marketplace'. Below it, there are two tabs: 'Azure Marketplace' and 'Featured'. Under 'Featured', there are four categories: 'Get started', 'Recently created', 'Compute', and 'Networking'. To the right of these is a 'Mobile App' section with a 'Learn more' link. Further down is a 'Storage' section with a 'Logic App' section below it. At the bottom of the 'Storage' section, a 'Web + Mobile' category is highlighted with a dashed blue border and a red box. This 'Web + Mobile' category contains a 'Web App' item with a 'Quickstart tutorial' link, which is also highlighted with a red box.

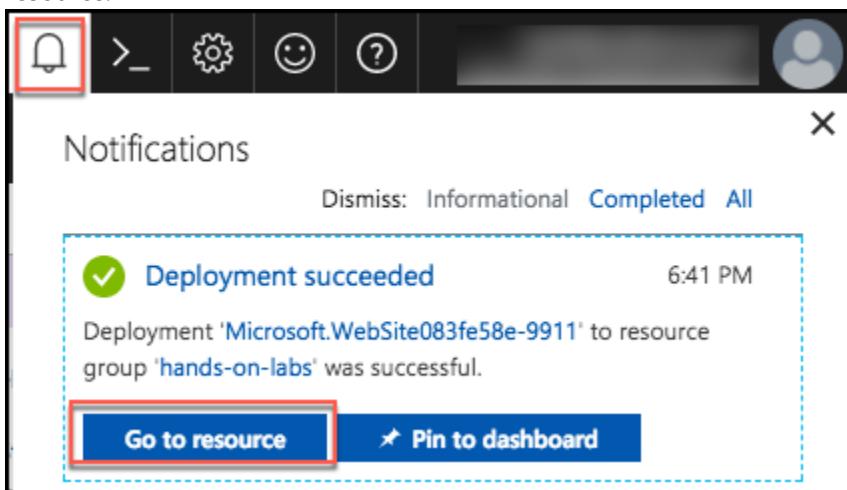
2. On the Create Web App blade, enter the following:

- 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. It consists of three tabs: 'Web App', 'App Service plan', and 'New App Service Plan'. The 'Web App' tab shows fields for App name (contosoeventsweb-kb), Subscription (Solliance MVP MSDN), Resource Group (hands-on-labs), OS (Windows), and Application Insights (On). The 'App Service plan' tab shows a list of existing plans: ServicePlan69507b4-8322(S1) in South Central US (1 instance(s)), DemoAppPlan(S1) in East US (1 instance(s)), DefaultTier1ServerFarm(F1) in West US (Free), pschatappserviceplan(S1) in West US 2 (1 instance(s)), and azure-scenario-experience(F1) in West US (Free). The 'New App Service Plan' tab shows fields for App Service plan (contosoeventsplan-kb), Location (West US 2), and Pricing tier (S1 Standard). A blue 'OK' button is at the bottom right of this tab.

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.

The screenshot shows the 'Web Apps' overview blade. It displays details for a web app named 'hands-on-labs': Status (Running), Location (West US 2), OS name (Windows Server 2016), and URL (highlighted with a red box). The URL is <https://contosoeventsweb-kb.azurewebsites.net>. Other sections include 'Resource group (change)' and 'App Service plan/pricing tier'.

## Task 4: 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 shows two items: 'Function App' (Microsoft) and 'Functions Bot' (Microsoft). The 'Function App' row is highlighted with a red box.

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 ?  
 Create new  Use existing  
hands-on-labs ▼

\* OS Windows Linux (Preview)

\* Hosting Plan ?  
Consumption Plan ▼

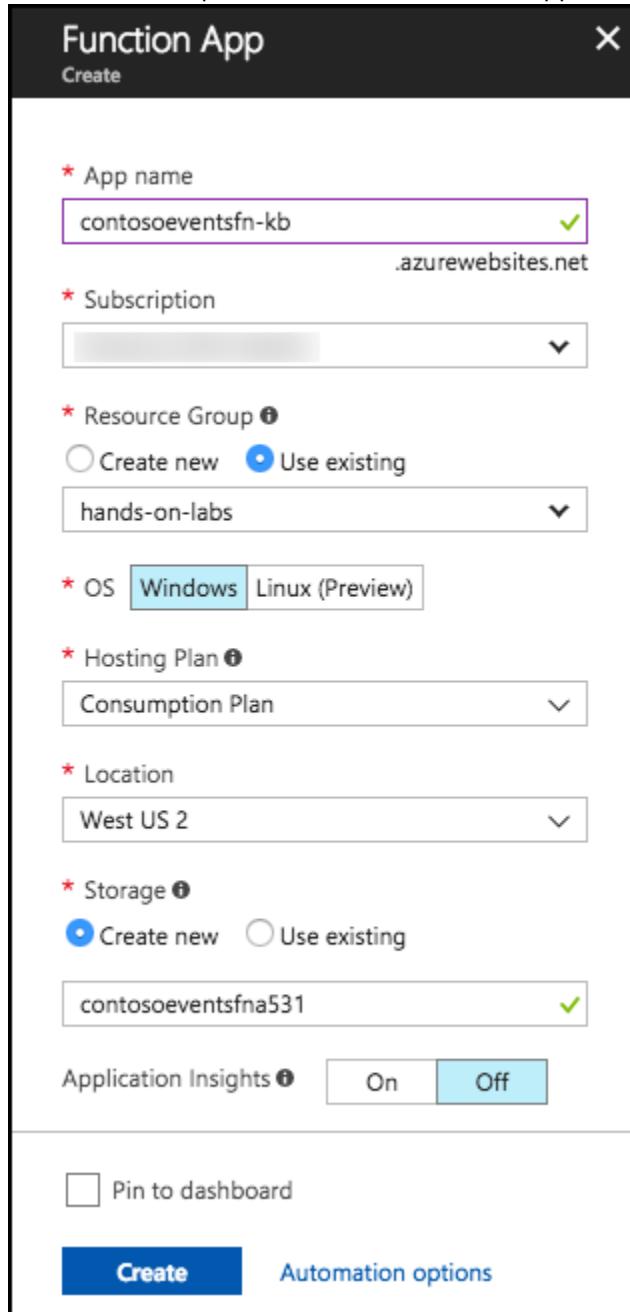
\* Location  
West US 2 ▼

\* Storage ?  
 Create new  Use existing  
contosoeventsfn531 ✓

Application Insights ? On Off

Pin to dashboard

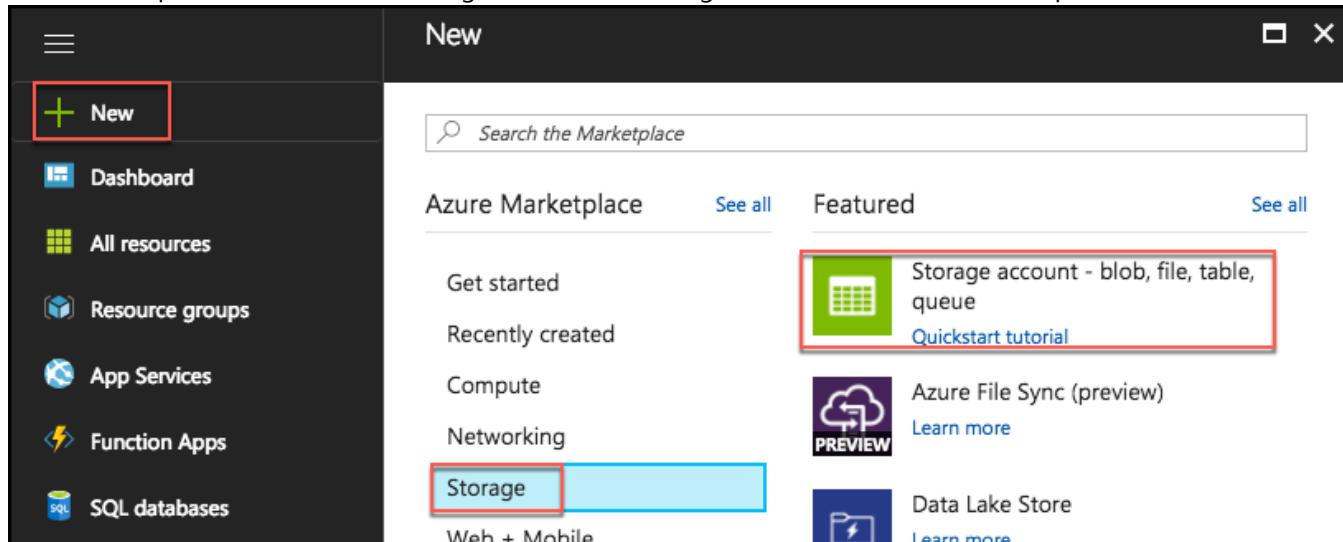
Create Automation options



## Task 5: 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 □ X

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

\* Name i  
contosoeventskb .core.windows.net

Deployment model i  
 Resource manager  Classic

Account kind i  
 Storage (general purpose v1) ▼

Performance i  
 Standard  Premium

Replication i  
 Locally-redundant storage (LRS) ▼

\* Secure transfer required i  
 Disabled  Enabled

\* Subscription  
▼

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

\* Location  
 West US 2 ▼

Virtual networks (Preview)  
Configure virtual networks i  
 Disabled  Enabled

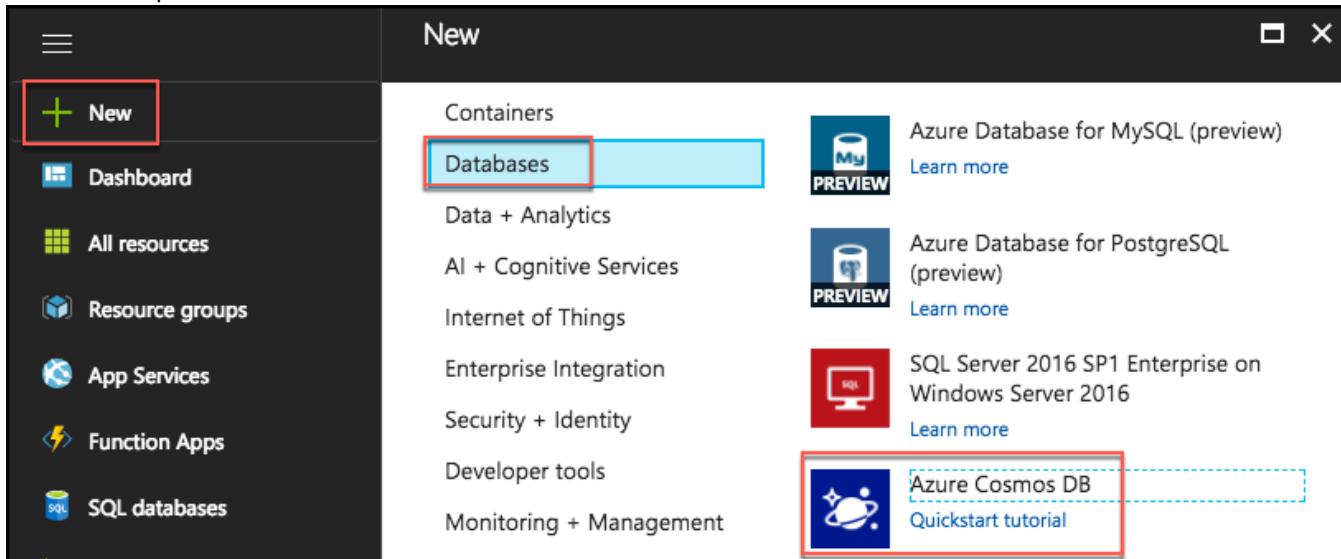
Pin to dashboard

Create [Automation options](#)

## Task 6: 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

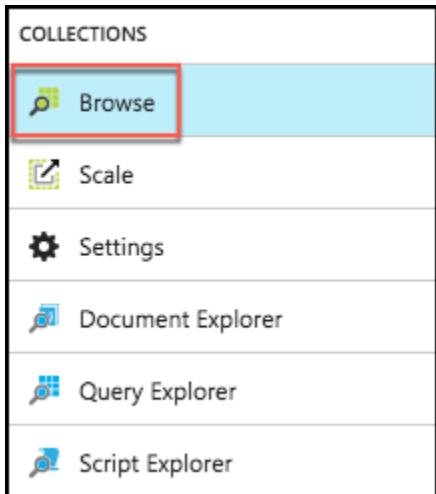
- g. Select Create to provision the Cosmos DB

The screenshot shows the 'Azure Cosmos DB' creation dialog. The 'ID' field is set to 'contosoeventsdb-kb'. The 'API' dropdown is set to 'SQL'. The 'Subscription' dropdown is empty. The 'Resource Group' section shows 'Create new' is unselected and 'Use existing' is selected with 'hands-on-labs' chosen. The 'Location' is set to 'West US'. The 'Enable geo-redundancy' checkbox is checked. At the bottom, there is a 'Pin to dashboard' checkbox and two buttons: 'Create' (highlighted in 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
contosoeventsdb-kb	Azure Cosmos DB account	West US
contosoeventskb	Storage account	West US 2
contosoevents-kb	API Management service	West US 2
contosoeventsplan-kb	App Service plan	West US 2

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

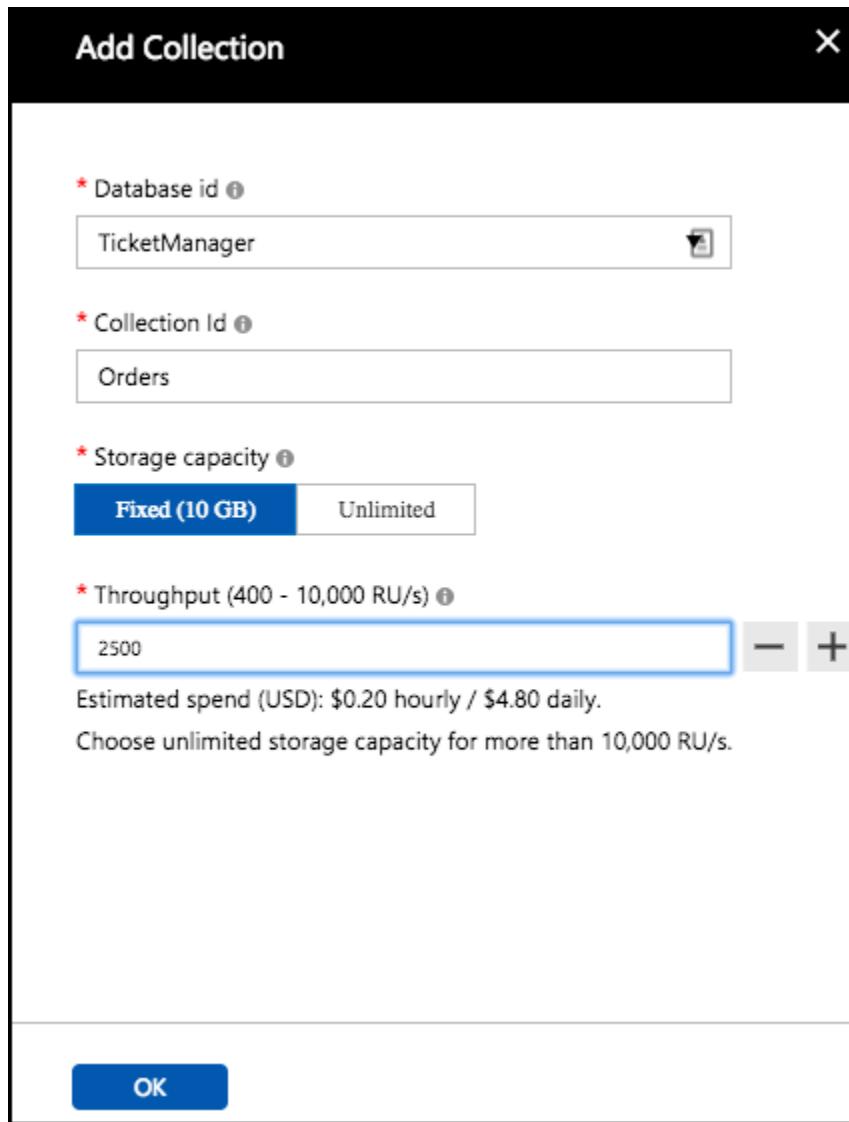


5. On the Browse blade, select +Add Collection.

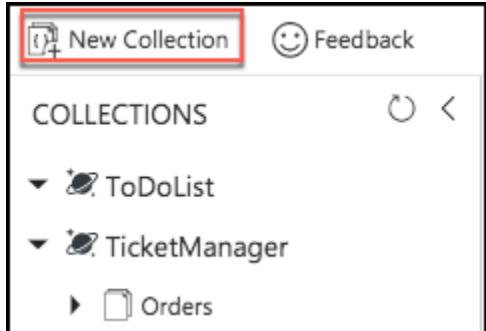


6. On the Add Collection dialog, enter the following:
- Database id: Enter TicketManager
  - Collection id: Enter Orders
  - Storage capacity: Select Fixed (10 GB)
  - Throughput: Enter 2500

- e. Select OK to create the new collection



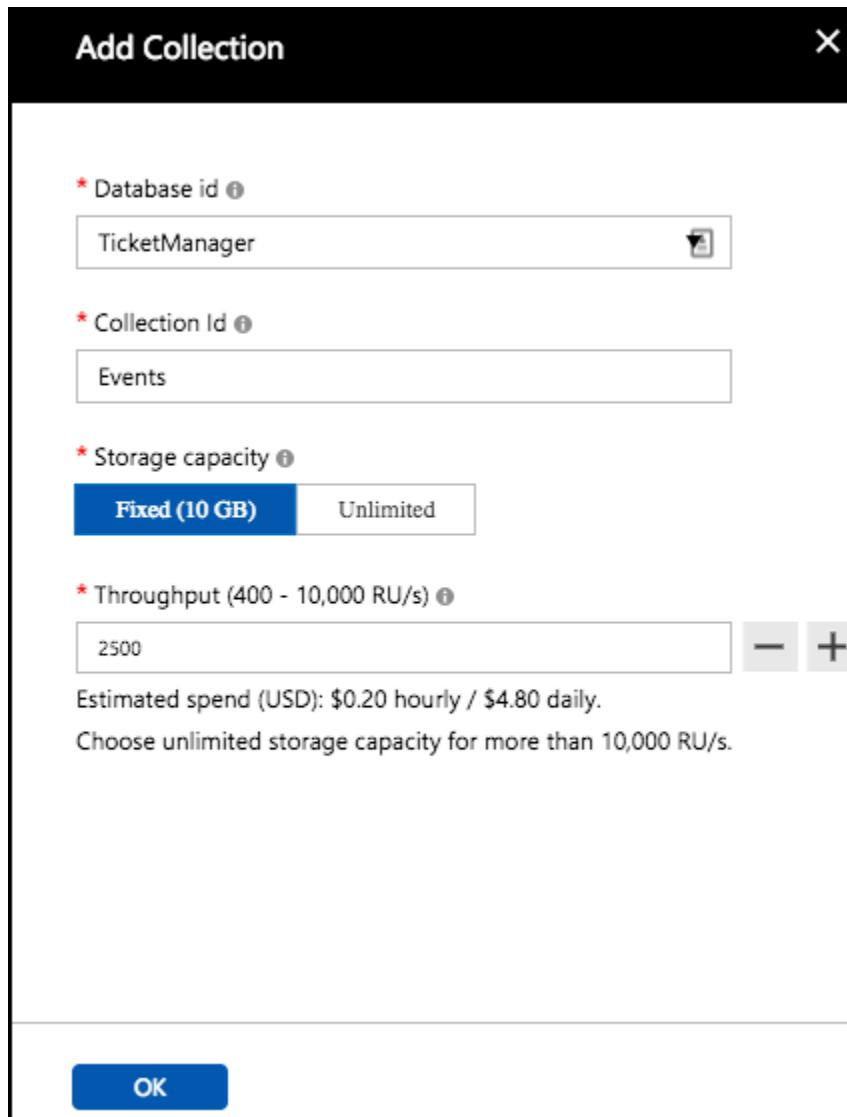
7. Select New collection from the new screen that appears.



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. Under 'COLLECTIONS', it lists 'ToDoList' and 'TicketManager'. The 'TicketManager' section is expanded, showing 'Orders' and 'Events', which are highlighted with a red border.

New Collection   Feedback

COLLECTIONS

ToDoList

TicketManager

Orders

Events

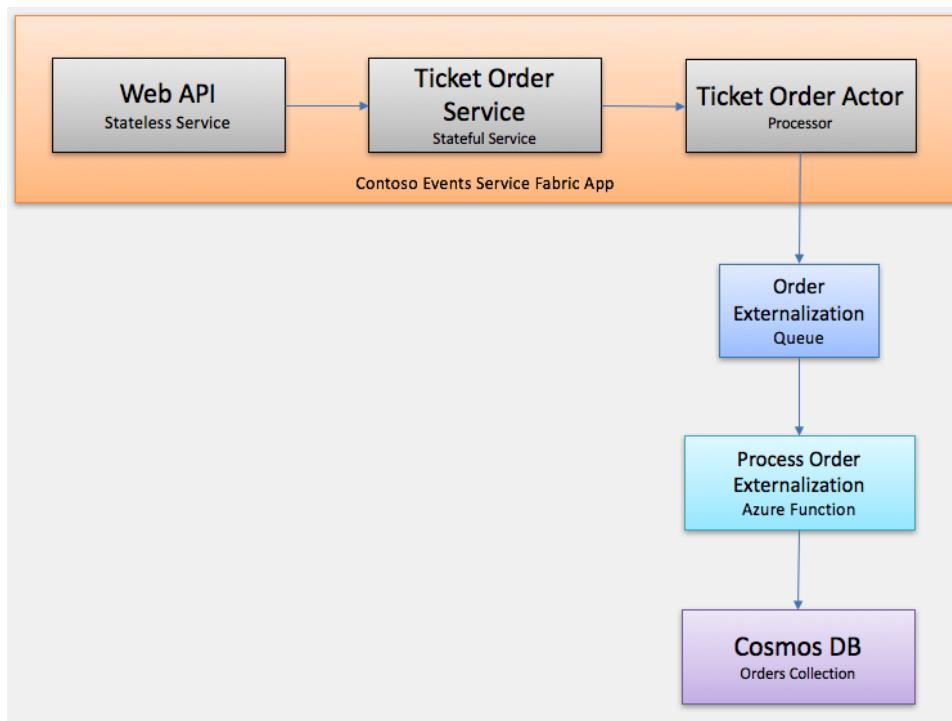
## Exercise 2: Implementing the Service Fabric solution

Duration: 60 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.

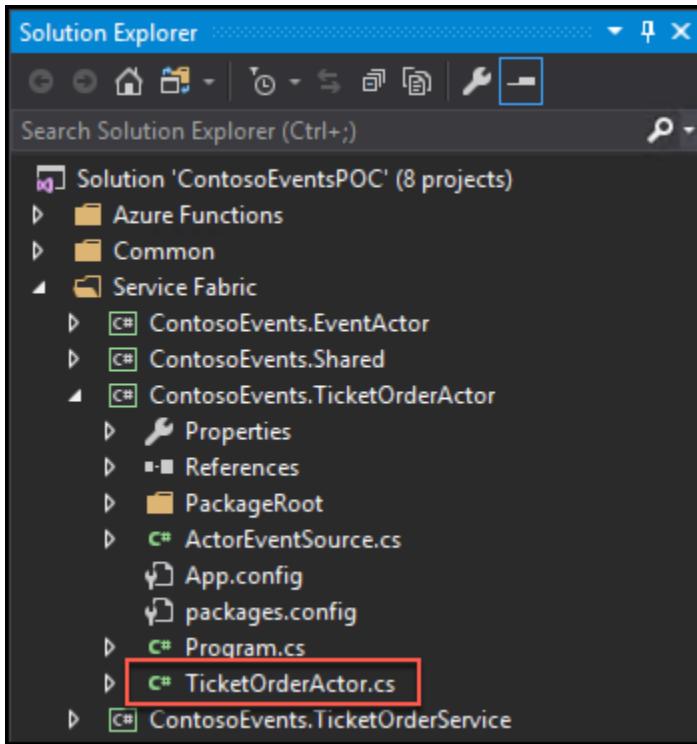
Note: The code to write to the storage queue is already in place within the actor, so setting up configuration keys is the only requirement to lighting up that feature.



### Task 1: Interacting with an Actor's State

In this task, you will write code to process the cancellation of an order by interacting with the Ticket Order Actor. To cancel an order, a Ticket Order Actor instance must be retrieved, and the `CancelTicket` operation it provides is invoked that changes the actor's state to reflect a "canceled" status.

1. On your Lab VM, with the ContosoEventsPoC solution open in Visual Studio, use the Solution Explorer to open the TicketOrderActor.cs file, in the Service Fabric folder, under the ContosoEvents.TicketOrderActor project.



2. Locate the CancelOrder method (line 239).
3. Locate the following TODO, and complete the commented line by replacing it with the following:

```
//TODO: Task 1.1 - Acquire an instance of the Actor  
IEventActor eventActor = this.ActorLocationService.Create<IEventActor>(new  
ActorId(state.EventId), Constants.ContosoEventsApplicationName);
```

4. Locate the following TODO and complete the commented line by replacing it with the following:

```
//TODO: Task 1.2 - Invoke the Cancel Ticket Operation with the supplied state  
await eventActor.CancelTickets(state);
```

5. Locate the following TODO and complete the commented lines by replacing them with the following:

```
//TODO: Task 1.3 - Update the state object to reflect that the order is cancelled  
state.CancellationDate = DateTime.Now;  
state.IsFulfilled = false;  
state.IsCancelled = true;
```

6. Locate the following TODO and complete the commented line by replacing it with the following:

```
//TODO: Task 1.4 - Save the updated state  
await SetEntityStateAsync(state);
```

7. Navigate to the SetEntityStateAsync method implementation, locate the following TODO, and complete the commented line by replacing it with the following:

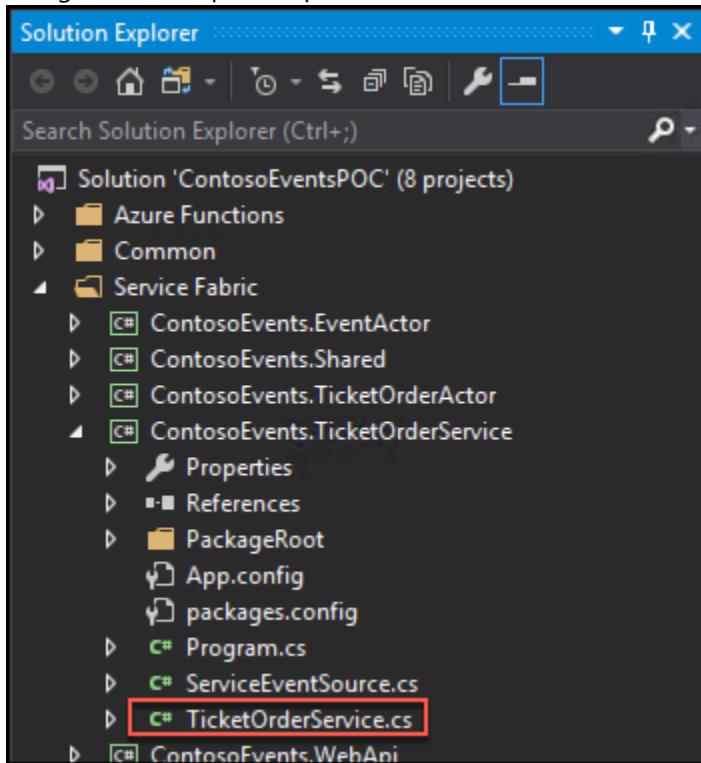
```
//TODO: Task 1.5 - update the actor state with the new state  
await this.StateManager.SetStateAsync<TicketOrder>(ActorStatePropertyName, state);
```

8. Save the file. At this point you have completed the code to cancel an order using the Ticket Order Actor. However, the solution is still incomplete (and will not compile if you try to). Continue to the next task.

## Task 2: Interacting with a Stateful Service

In this task, you will write code to enqueue an order to the Ticket Order Stateful Service whenever an order is being processed. This Order is enqueued by the Web API and then dequeued by the Ticket Order Stateful Service for processing by the Ticket Order Actor. To accomplish this, the order will be enqueued into a reliable queue.

1. Using Solution Explorer open `TicketOrderService.cs` under the `ContosoEvents.TicketOrderService` project.



2. Locate the `EnqueueOrder` method. (line 67)
3. Locate the following TODO and complete the commented line by replacing it with the following:

```
//TODO: Task 2.1 - Get (or create) a reliable queue called "OrderQueue" in this partition.
var requests = await
this.StateManager.GetOrAddAsync<IReliableQueue<TicketOrder>>(OrderQueueName);
```

4. Locate the following TODO and complete the commented line by replacing it with the following:

```
//TODO: Task 2.2 - Create a new transaction scope
using (var tx = this.StateManager.CreateTransaction())
{
```

5. Locate the following TODO and complete the commented line by replacing it with the following:

```
//TODO: Task 2.3 - Enqueue the order to the reliable queue within the transaction
await requests.EnqueueAsync(tx, order);
```

6. Locate the following TODO and complete the commented line by replacing it with the following:

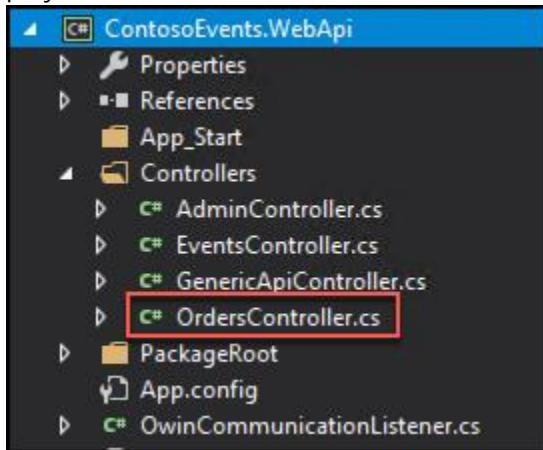
```
//TODO: Task 2.4 - Commit the transaction if enqueue was successful  
await tx.CommitAsync();  
}
```

7. Save the file. At this point you have completed the code to enqueue a message to the reliable queue. However, the solution is still incomplete (and will not compile if you try to). Continue to the next task.

## Task 3: Interacting with an Actor from a Web API Controller

In this task, you will write code that runs within the Orders Web API controller that delegates the cancellation request to the actor.

1. Using Solution Explorer, open the OrdersController.cs file, in the Controllers folder of the ContosoEvents.WebApi project.



2. Locate the CancelOrder method. (line 179)
3. Locate the following TODO and complete the commented line by replacing it with the following:

```
//TODO: Task 3.1 - Get the actor location service  
IActorLocationService locator = ServiceFactory.GetInstance().GetActorLocationService();
```

4. Locate the following TODO and complete the commented line by replacing it with the following:

```
//TODO: Task 3.2 - Acquire the Order Actor instance  
ITicketOrderActor orderActor = locator.Create<ITicketOrderActor>(new ActorId(id),  
Constants.ContosoEventsApplicationName);
```

5. Locate the following TODO and complete the commented line by replacing it with the following:

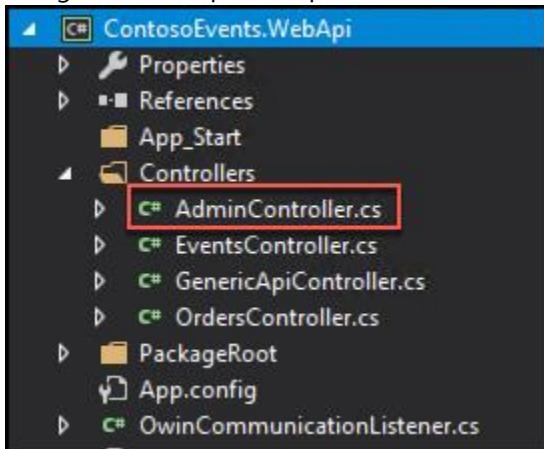
```
//TODO: Task 3.3 - Cancel the order  
await orderActor.CancelOrder();
```

6. Save the file. At this point you have completed the code to cancel the order in response to a request against the OrdersController CancelOrder operation. However, the solution is still incomplete (and will not compile if you try to). Continue to the next task.

## Task 4: Inspecting Service Partitions

In this task, you will write code that runs within the Admin Web API controller that uses the Fabric Client to get information about all the Ticket Order Service partitions.

1. Using Solution Explorer, open AdminControllers.cs in the Controllers folder in the ContosoEvents.WebApi project.



2. Locate the GetTicketOrderPartitions method. (line 37)
3. Locate the following TODO and complete the commented line by replacing it with the following:

```
//TODO: Task 4.1 - Use the FabricClient to get the list of partitions for the Ticket Order Service
ServicePartitionList partitions = await
    _fabricClient.QueryManager.GetPartitionListAsync(builder.ToUri());
```

4. Locate the following TODO and complete the commented line by replacing it with the following:

```
//TODO: Task 4.2 - Collect the partition info
infos.Add(new TicketOrderServiceInfo()
{
    PartitionId = p.PartitionInformation.Id.ToString(),
    PartitionKind = p.PartitionInformation.Kind.ToString(),
    PartitionStatus = p.PartitionStatus.ToString(),
    NodeName = await dispenderService.GetNodeName(),
    HealthState = p.HealthState.ToString(),
    ServiceKind = p.ServiceKind.ToString(),
    ItemsInQueue = await dispenderService.GetOrdersCounter(CancellationToken.None)
});
```

5. Save the file. At this point you have completed the code to inspect the partition information. Continue to the next exercise to run the solution and verify functionality.

## Exercise 3: Placing ticket orders

Duration: 30 minutes

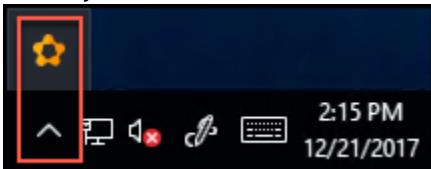
In this exercise, you will test that your completed solution works by running locally on your Lab VM.

### Task 1: Run the solution

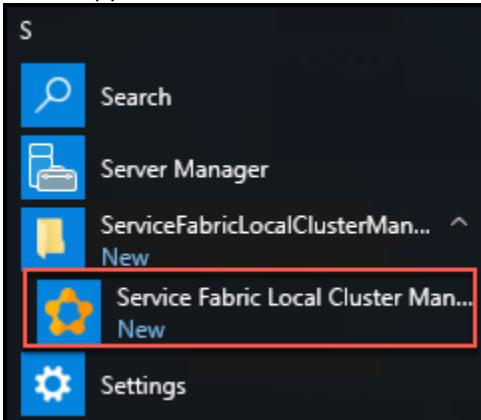
The purpose of this task is to make sure that the source code compiles, and that you can publish to a local cluster. To make sure that the app is running flawlessly, you will run some API tests and access the Service Fabric explorer.

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

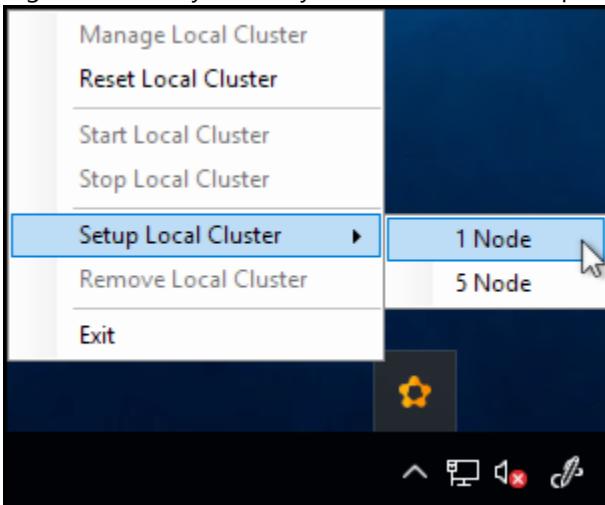
1. On the Lab VM, make sure the local Service Fabric environment is running by selecting the arrow in the system tray, and checking for the appearance of the Service Fabric icon. Note: You may need to restart your VM if you've recently installed the Service Fabric Local Cluster Manager, for the icon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



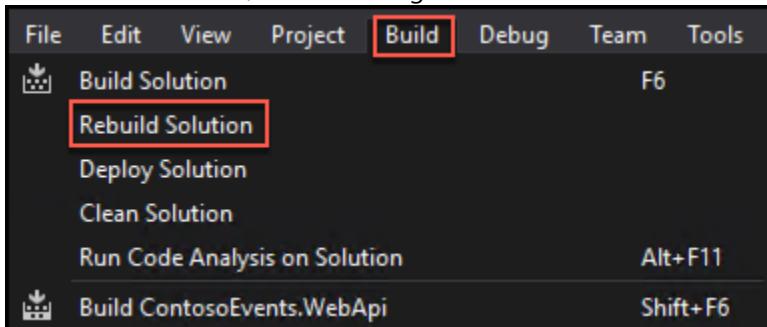
3. Right-click the system tray icon, and select Setup Local Cluster, 1 Node.



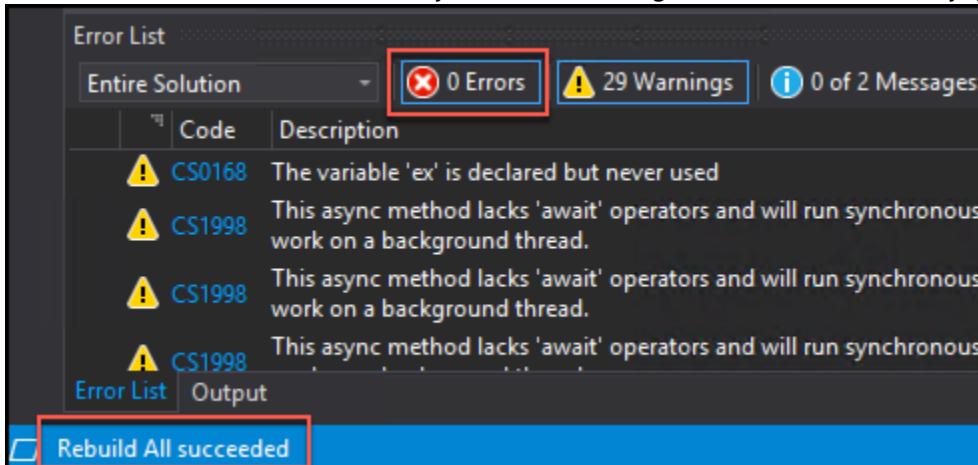
4. You should see a message that the local cluster is setting up.



5. Note: Sometimes you will see an error message appear indicating the cluster setup is retrying.
6. Open the ContosoEventsPoc solution in Visual Studio, if it is not still open from the previous exercise.
7. 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.

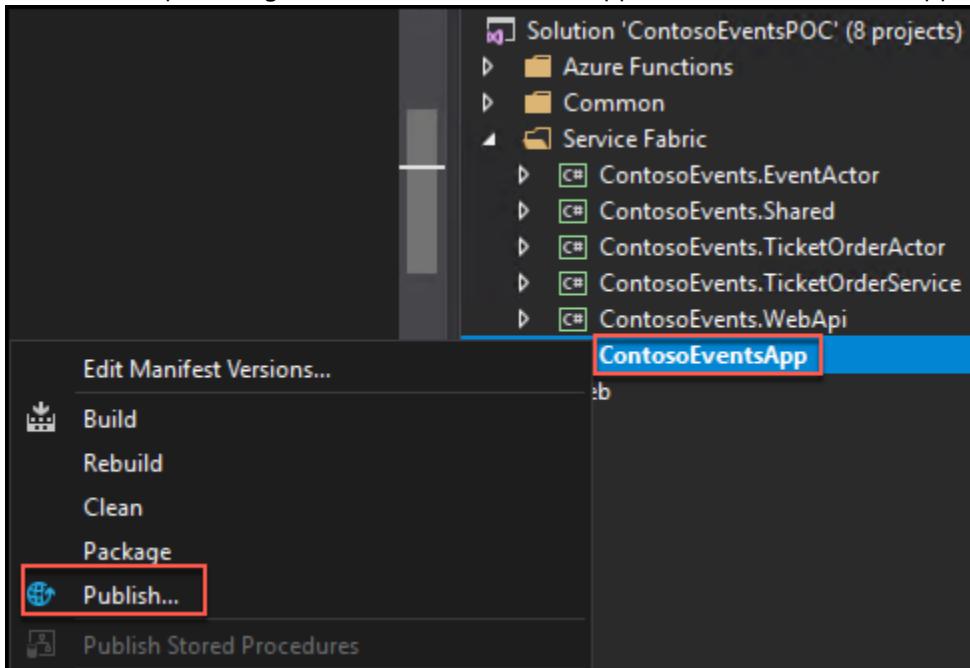


8. After the rebuild, you should see a Rebuild All succeeded message in the bottom left corner of Visual Studio, and that there are 0 Errors. Note: You may see some warnings, but these can be safely ignored.

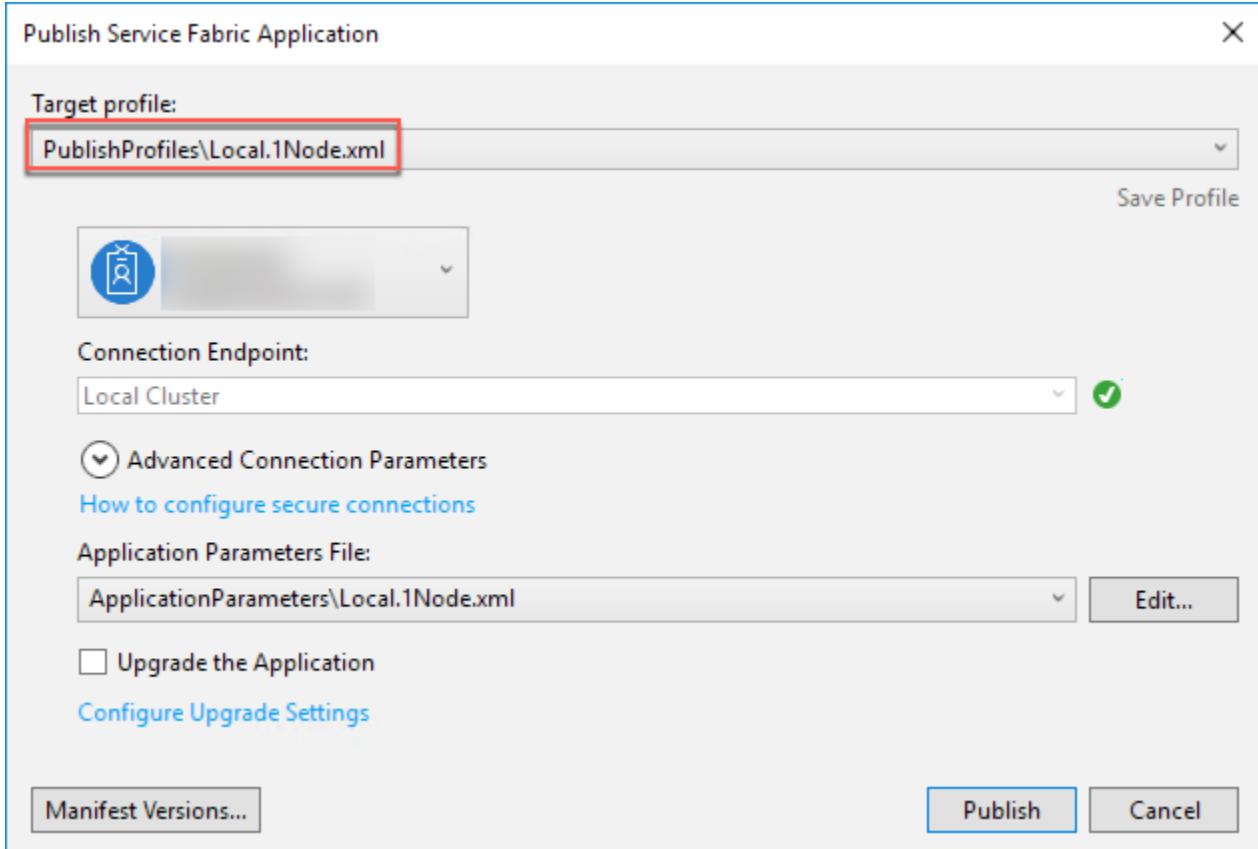


9. Now, you will Publish the Service Fabric app to the local cluster.

10. In Solution Explorer, right-click the Service Fabric Application, ContosoEventsApp, and select Publish.

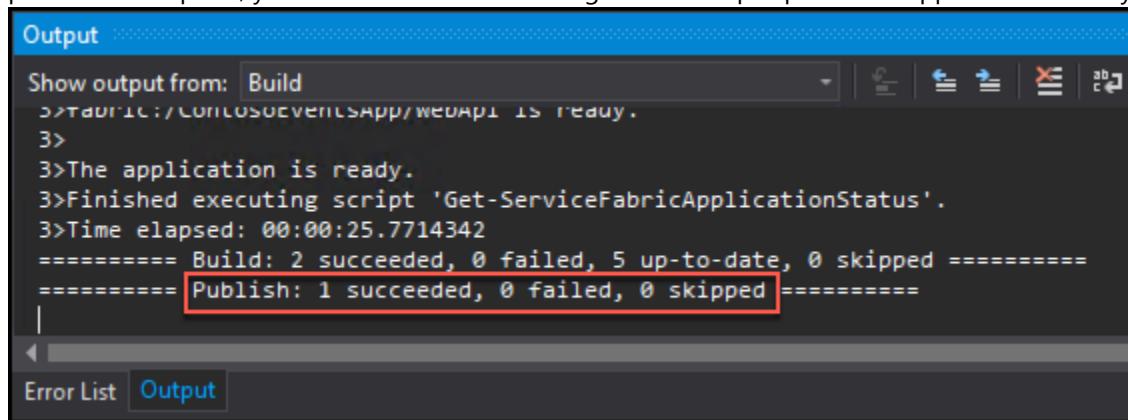


11. In the Public Service Fabric Application dialog, select PublishProfiles\Local.1Node.xml for the Target profile, ensure the correct account is selected, and select Publish.



**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.

12. 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.



The screenshot shows the Visual Studio Output pane with the following log entries:

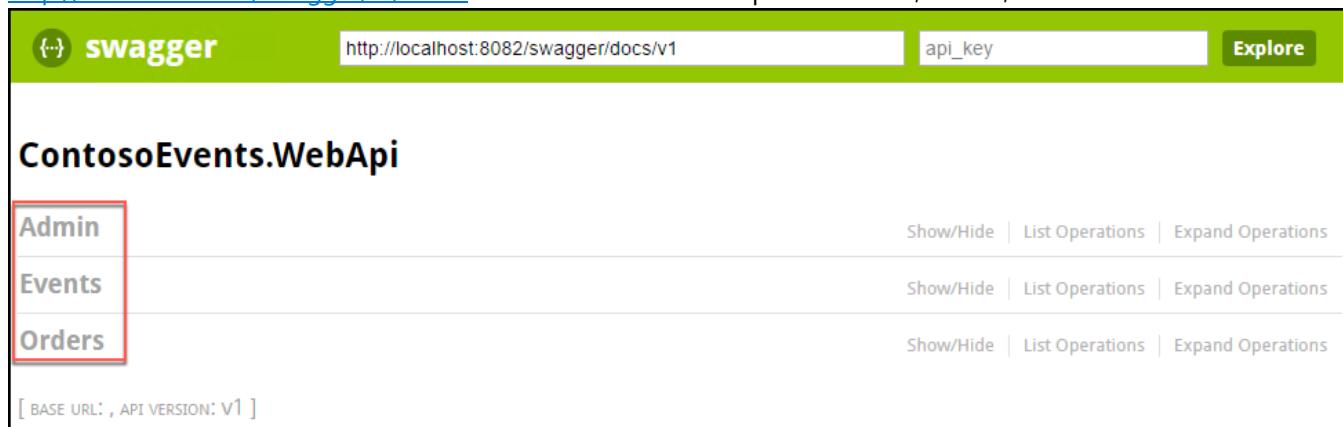
```
Output
Show output from: Build
D:\TdURL\ContosoEventsApp\WebAPI\ is ready.
3>
3>The application is ready.
3>Finished executing script 'Get-ServiceFabricApplicationStatus'.
3>Time elapsed: 00:00:25.7714342
===== Build: 2 succeeded, 0 failed, 5 up-to-date, 0 skipped =====
===== Publish: 1 succeeded, 0 failed, 0 skipped =====
|
```

The "Publish: 1 succeeded, 0 failed, 0 skipped" line is highlighted with a red box.

## 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. The top navigation bar includes a logo, the URL <http://localhost:8082/swagger/docs/v1>, a dropdown for "api\_key", and a "Explore" button. Below the header, the title "ContosoEvents.WebApi" is displayed. The main content area lists three API endpoints:

- Admin** (highlighted with a red box)
- Events**
- Orders**

Each endpoint row contains "Show/Hide", "List Operations", and "Expand Operations" buttons. At the bottom left, there is 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 **ContosoEvents.WebApi**. At the top, there's a navigation bar with the Swagger logo, the URL <http://localhost:8082/swagger/docs/v1>, a search bar for 'api\_key', and a 'Explore' button. Below the header, the title 'ContosoEvents.WebApi' is displayed. The main content area is organized into sections: 'Admin', 'Events', and 'Orders'. The 'Admin' section is currently active and contains a list of API operations:

Method	Path
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 shows a method (e.g., GET, POST) and its corresponding endpoint path. The 'Events' and 'Orders' sections are also present but are collapsed. 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

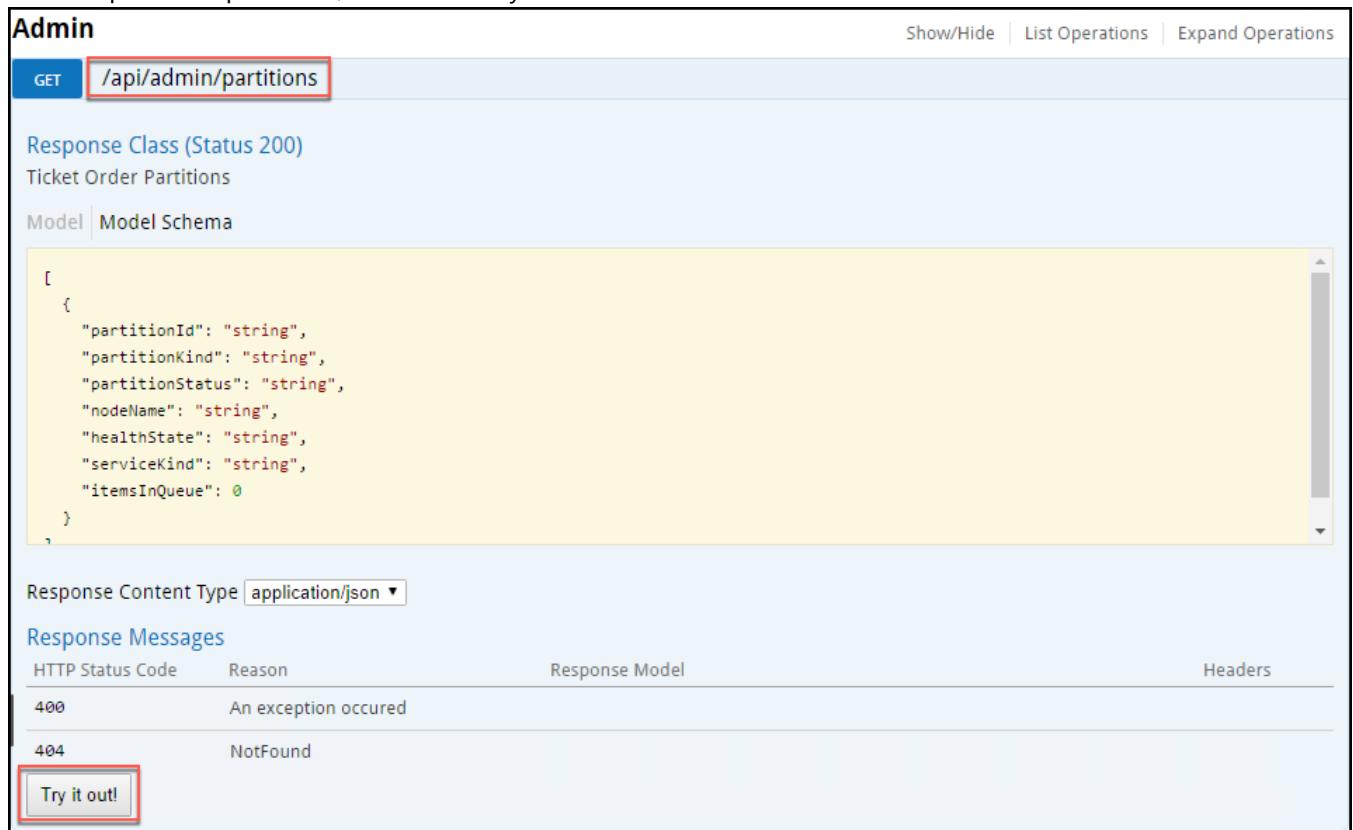
```
[  
  {  
    "partitionId": "string",  
    "partitionKind": "string",  
    "partitionStatus": "string",  
    "nodeName": "string",  
    "healthState": "string",  
    "serviceKind": "string",  
    "itemsInQueue": 0  
  }  
,
```

Response Content Type **application/json ▾**

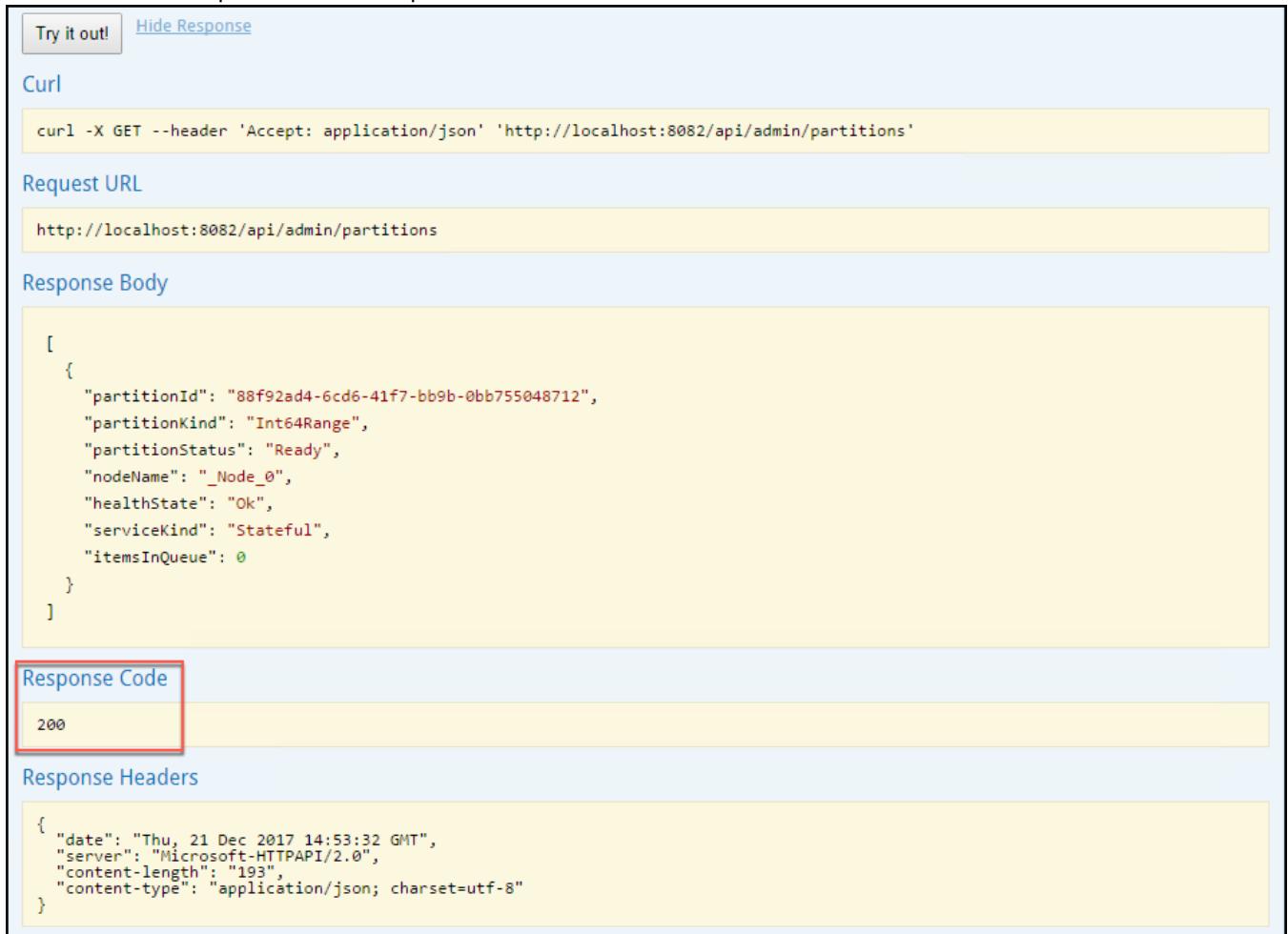
**Response Messages**

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 the Swagger UI interface for a microservices architecture. It displays the following sections:

- Curl**: A code snippet using curl to make a GET request to `http://localhost:8082/api/admin/partitions`.
- Request URL**: The URL `http://localhost:8082/api/admin/partitions`.
- Response Body**: A JSON array containing one element, representing a partition state. The JSON is:

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

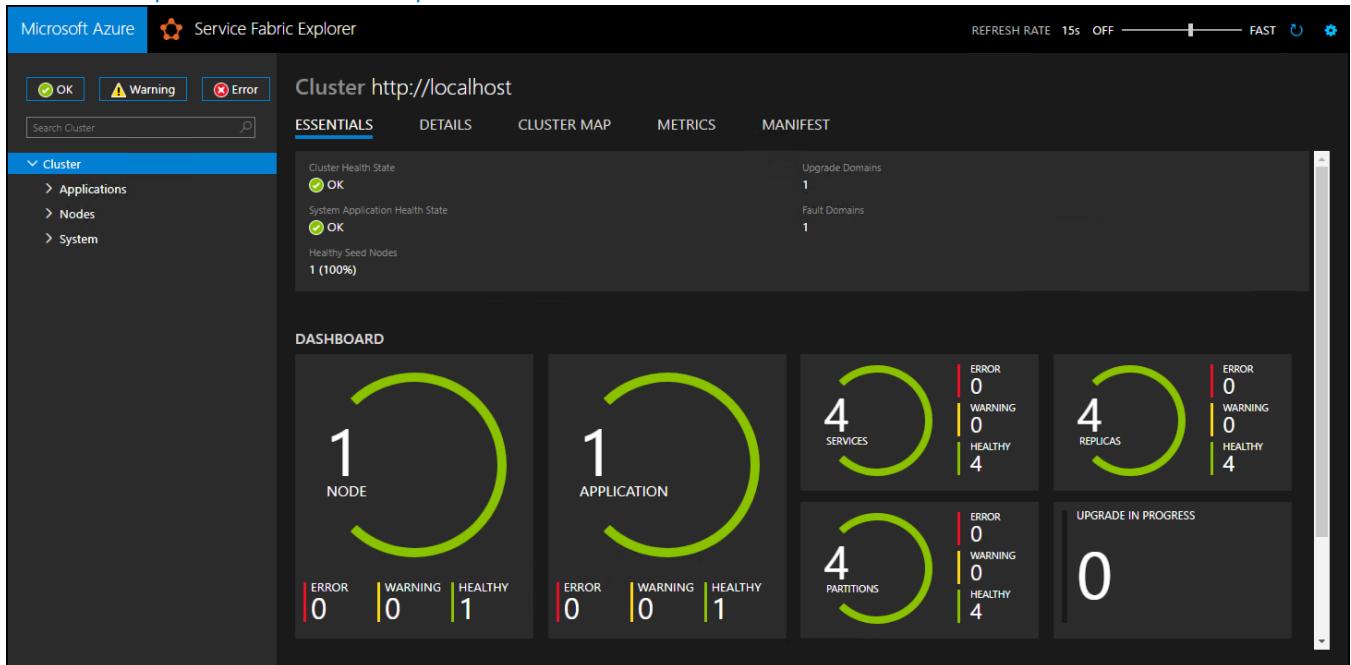
```
{  
  "date": "Thu, 21 Dec 2017 14:53:32 GMT",  
  "server": "Microsoft-HTTPAPI/2.0",  
  "content-length": "193",  
  "content-type": "application/json; charset=utf-8"  
}
```

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.

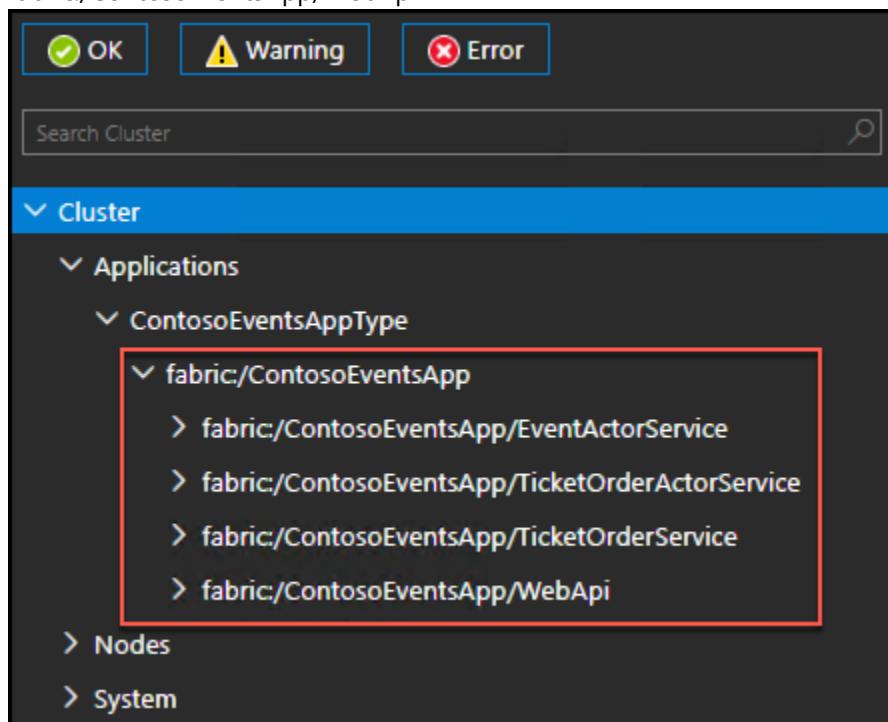
## Task 3: Service Fabric Explorer

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>.



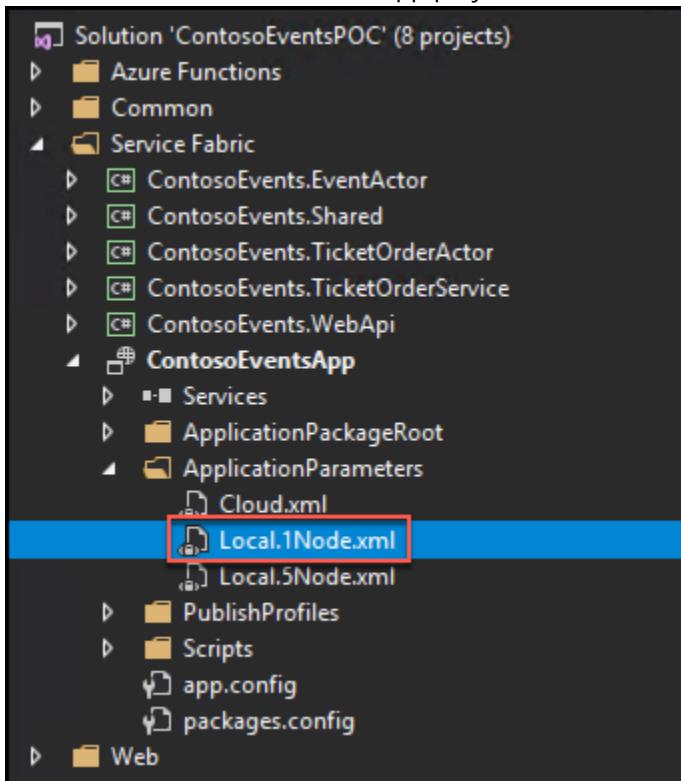
2. Observe that the ContosoEventsApp is deployed with the following services:
  - a. fabric:/ContosoEventsApp/EventActorService
  - b. fabric:/ContosoEventsApp/TicketOrderActorService
  - c. fabric:/ContosoEventsApp/TicketOrderService
  - d. fabric:/ContosoEventsApp/WebApi



## 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

- In the Azure Portal, browse to the Azure Cosmos DB account you created in [Exercise 1, Task 6](#), and select Keys, under Settings in the left-hand menu.

**contosoeventsdb-kb - Keys**  
Azure Cosmos DB account

SETTINGS

- Replicate data globally
- Default consistency
- Firewall
- Keys**
- Add Azure Search
- Add Azure Function
- Locks

Read-write Keys    Read-only Keys

URI  
https://contosoeventsdb-kb.documents.azure.com:443/

PRIMARY KEY  
V592d2J3TTCZld0aaPZVAJ05rx4bMLPh4SHWHfrFUfxDxSCHpngodqd72vBMFUO3CCB8s5l...

SECONDARY KEY  
877949Aaq6lzoSYDFDxN3bY9xheKtFx4Q4ecR8u4Wd558Zi3JWHVXDHzoWFxILzdh3EoX...

PRIMARY CONNECTION STRING  
AccountEndpoint=https://contosoeventsdb-kb.documents.azure.com:443/;AccountKey=...

SECONDARY CONNECTION STRING  
AccountEndpoint=https://contosoeventsdb-kb.documents.azure.com:443/;AccountKey=...

- 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.
- 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.
- 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.

**contosoeventsdb-kb - Browse**  
Azure Cosmos DB account

COLLECTIONS

- Browse**
- Scale
- Settings
- Document Explorer

+ Add Collection    + Add Database    Refresh    Delete Collection    ... More

Database  
All Databases

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

- If you used a different database name, modify the value of the DataStorageDatabaseName parameter to reflect the Cosmos DB database name.
- 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.
- 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="V592d2J3TTCZld0aaPZVAJ05rx4bMLPh4SHWHfrFUfxDxSCHpngodqd72vBMFUO3CCB8s5l..." />
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 5](#), 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.

contosoeventskb - Access keys

Storage account

SETTINGS

Access keys

Configuration

Shared access signature

Firewalls and virtual networks (P...)

Metrics (preview)

Properties

Locks

Automation script

Use access keys to authenticate your applications when making requests to this Azure storage account. Store your access keys securely - for example, using Azure Key Vault - and don't share them. We recommend regenerating your access keys regularly. You are provided two access keys so that you can maintain connections using one key while regenerating the other.

When you regenerate your access keys, you must update any Azure resources and applications that access this storage account to use the new keys. This action will not interrupt access to disks from your virtual machines. [Learn more](#)

Storage account name contosoeventskb

Default keys

NAME	KEY	CONNECTION STRING
key1	Ibjjzzc1WHAyZR8wWjX2KG...	DefaultEndpointsProtocol=https;AccountName=contosoeventskb;AccountKey=Ibjjzzc1WHAyZR8wWjX2KG...
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=Ibjjzzc1WHAyZR8wWjX2KG...">
```

## 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 selected. Below it, a list of operations is shown:

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

The 'POST /api/orders' operation is highlighted with a red box. To its right, there is a 'Response Class (Status 200)' section. Under 'Parameters', there is a table:

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

The 'Model Schema' pane contains the following JSON 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  
}
```

Below the schema, there is a note: 'Click to set as parameter value'.

4. POST an order to the Contoso Events application. Copy this 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	<a href="#">Model</a>   <a href="#">Model Schema</a> <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**

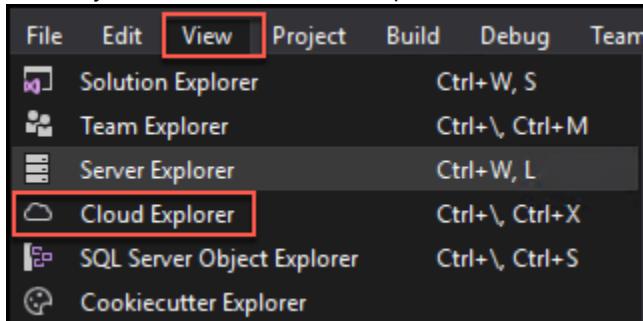
```
200
```

**Response Headers**

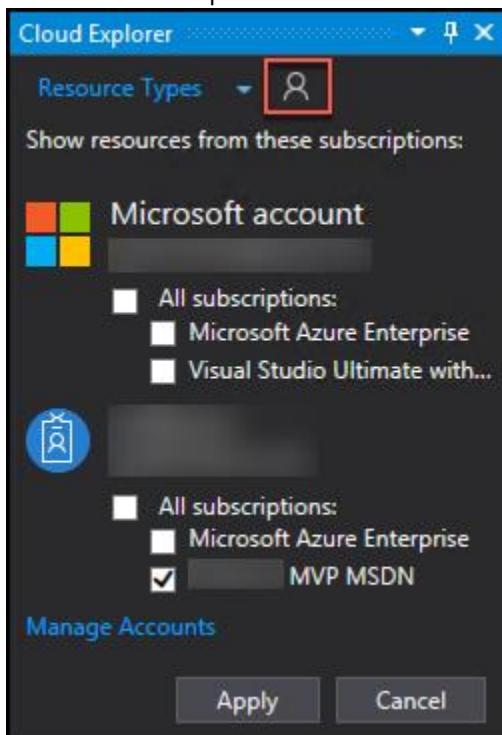
```
{
    "access-control-allow-origin": "*",
    "date": "Thu, 21 Dec 2017 16:03:15 GMT",
    "server": "Microsoft-HTTPAPI/2.0",
    "content-length": "38",
    "content-type": "application/json; charset=utf-8"
}
```

**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.

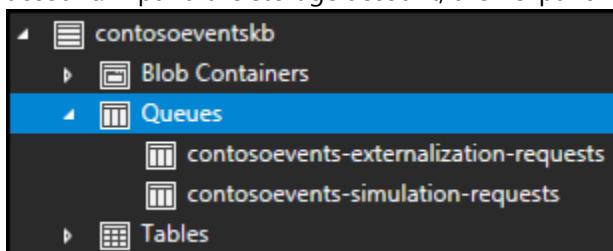
6. Now, you should have an order in the Service Fabric App, and it is being processed. In addition, the Ticket Order Actor should have sent the order to the externalization queue you set up in configuration earlier. The actor has pre-existing logic in place to write to this queue.
7. To verify that the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



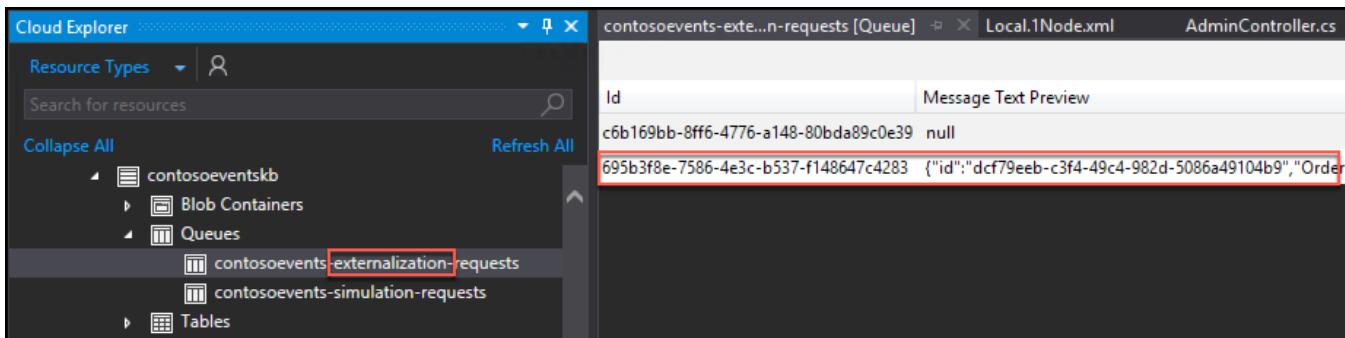
8. In the Cloud Explorer, select the Account Management icon, and select the check box next to the subscription you are using for this lab, then select Apply. You may need to re-enter your account credentials to see the resources under the subscription.



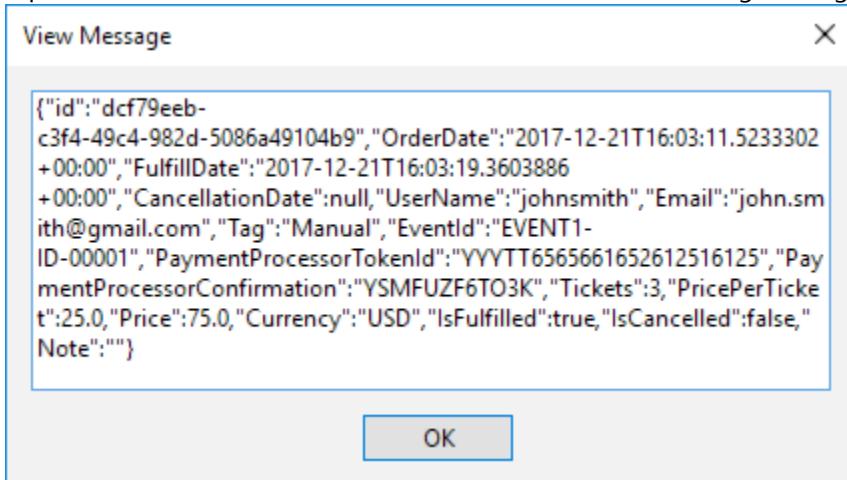
9. Now, expand Storage Accounts under your subscription in Cloud Explorer, and locate the Storage account you set up in [Exercise 1, Task 5](#). You may need to select Load more at the bottom of the list, if you don't see the storage account. Expand the storage account, then expand Queues.



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



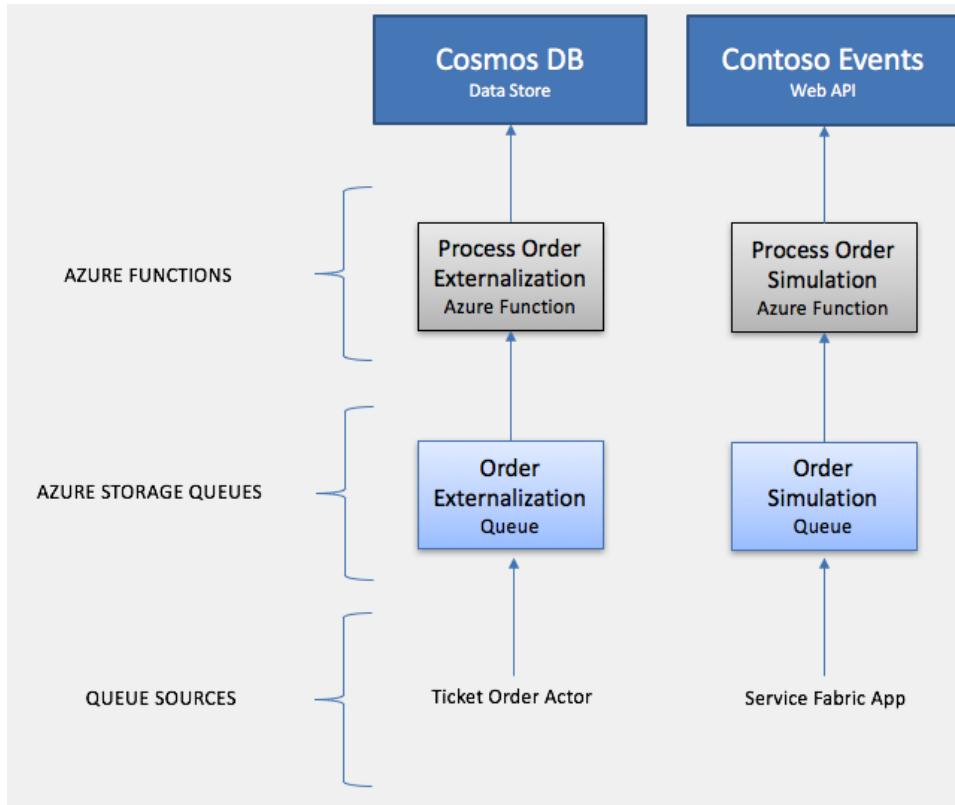
11. Now, double-click on the message, and you will see the contents of the message including the JSON representation of the order. Select OK to close the View Message dialog.



## 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 5](#), 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. It displays two keys: 'key1' and 'key2'. The 'key1' row shows the connection string 'DefaultEndpointsProtocol=https;AccountName=co...' with a copy icon. The 'key2' row shows a different connection string with a copy icon. The 'key1' connection string is highlighted with a red box.

3. Now, browse to the Function App you created in [Exercise 1, Step 4](#).

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

The screenshot shows the Azure portal interface for a function app named 'contosoeventsfn-kb'. On the left, there's a sidebar with a search bar and dropdown for 'All subscriptions'. Below that is a tree view with 'Function Apps' expanded, showing 'contosoeventsfn-kb' selected (highlighted with a red box). Underneath are 'Functions', 'Proxies', and 'Slots (preview)'. On the right, the 'Overview' tab is active, showing the app is 'Running'. Below it, the 'Configured features' section has a link to 'Function app settings' and 'Application settings' (also highlighted with a red box).

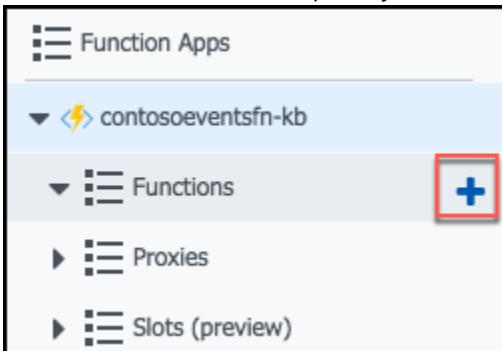
5. On the Application Settings tab, scroll down and select +Add new setting under Application Settings, then enter contosoeventsstore in the name textbox, and paste the key1 Connection String value you copied from your Storage account into the value textbox.

The screenshot shows the 'Application settings' tab in the Azure portal. It lists several environment variables: 'AzureWebJobsDashboard', 'AzureWebJobsStorage', 'FUNCTIONS\_EXTENSION\_VERSION', 'WEBSITE\_CONTENTAZUREFILECONNEC...', 'WEBSITE\_CONTENTSHARE', and 'WEBSITE\_NODE\_DEFAULT\_VERSION'. At the bottom, there's a row for 'contosoeventsstore' with a value field containing a long connection string. Below this row is a blue button labeled '+ Add new setting' (also highlighted with a red box).

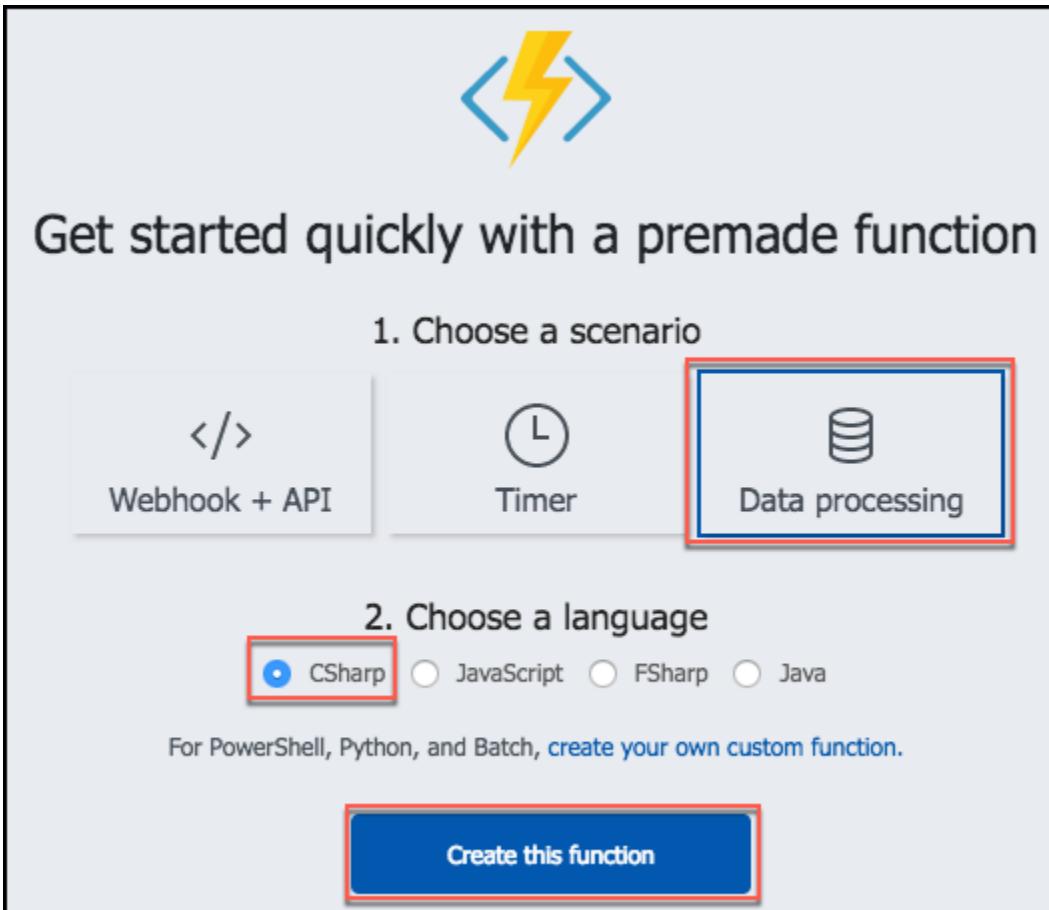
6. Scroll back to the top of the Application Settings tab, and select Save to apply the change.



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.



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 navigation sidebar with 'Function Apps' selected. Under 'Function Apps', 'contosoeventsfn-kb' is expanded, showing 'Functions' (with 'QueueTriggerCSharp1' listed), 'Integrate', 'Manage', 'Monitor', 'Proxies', and 'Slots (preview)'. On the right, the main content area has tabs for 'Overview' and 'Platform features', with 'Platform features' currently selected. A search bar is at the top of this section. Below it are several sections: 'GENERAL SETTINGS' (Function app settings, Application settings, Properties, Backups, All settings), 'NETWORKING' (Networking, SSL, Custom domains, Authentication / Authorization, Managed service identity, Push notifications), 'API' (API definition, CORS), 'APP SERVICE PLAN' (App Service plan, Quotas), 'RESOURCE MANAGEMENT' (Activity log, Access control (IAM), Tags, Locks, Automation script), and 'DEVELOPMENT TOOLS' (Logic Apps, Console, Advanced tools (Kudu), App Service Editor, Resource Explorer, Extensions). The 'Console' link under 'Development Tools' is highlighted with a red box.

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

The screenshot shows a Kudu console window with a complex, abstract logo at the top. Below it, the text 'Manage your web app environment by running common' and 'require elevated privileges won't work.' is displayed in blue. The command prompt shows the path 'D:\home\site\wwwroot'. The user types 'ls' and presses enter. The output shows two files: 'QueueTriggerCSharp1' and 'host.json', both highlighted in green.

```
D:\home\site\wwwroot

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

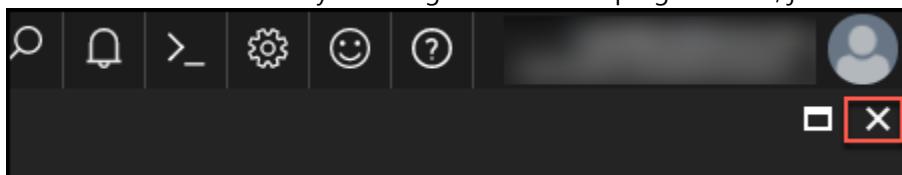
12. Next, paste the following command into the console to rename the QueueTriggerCSharp1 function to ProcessOrderExternalizations.

```
rename QueueTriggerCSharp1 ProcessOrderExternalizations
```

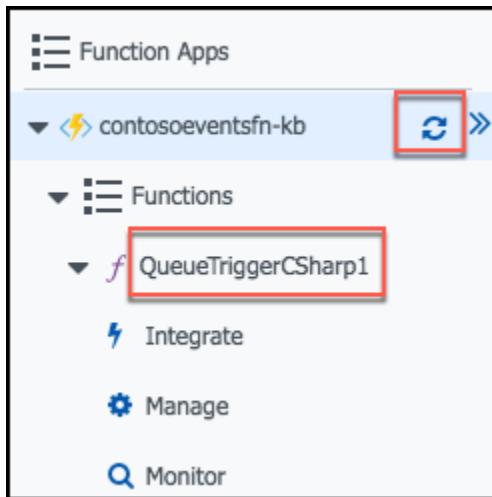
13. Typing ls at the command prompt again will show the function has been renamed.

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

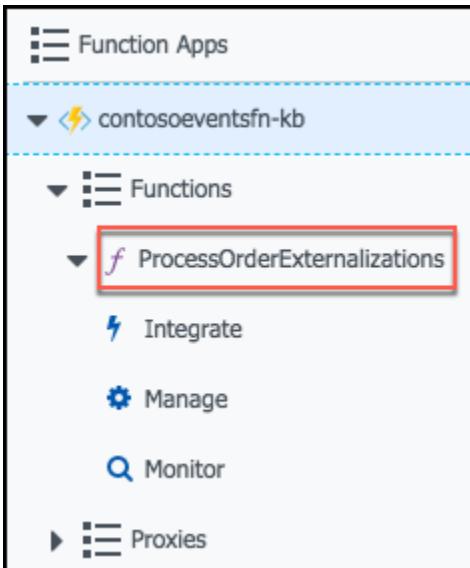
14. Exit the Console window by selecting the X in the top right corner, just below your account name.



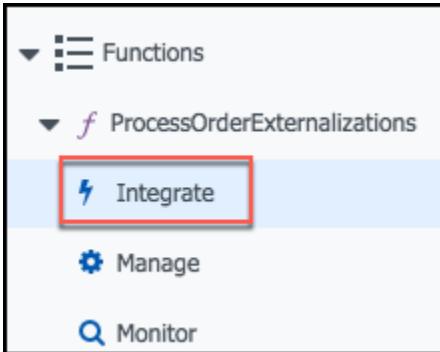
15. Back on the Function app page, move your mouse cursor over your function app in the left-hand menu, and select the Refresh icon. Note the function name is still QueueTriggerCSharp1 before refreshing.



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

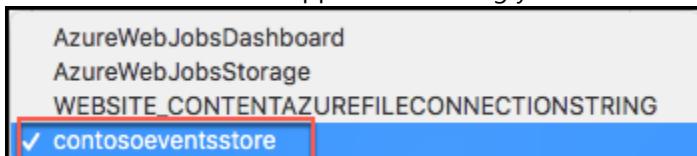


17. Under the ProcessOrderExternalizations function, select Integrate.

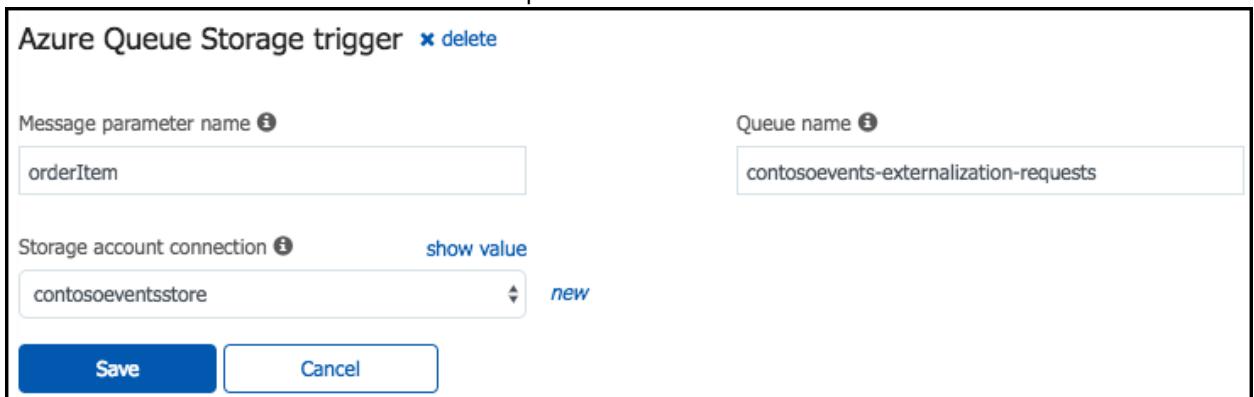


18. On the Integrate screen, set the following:

- 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.



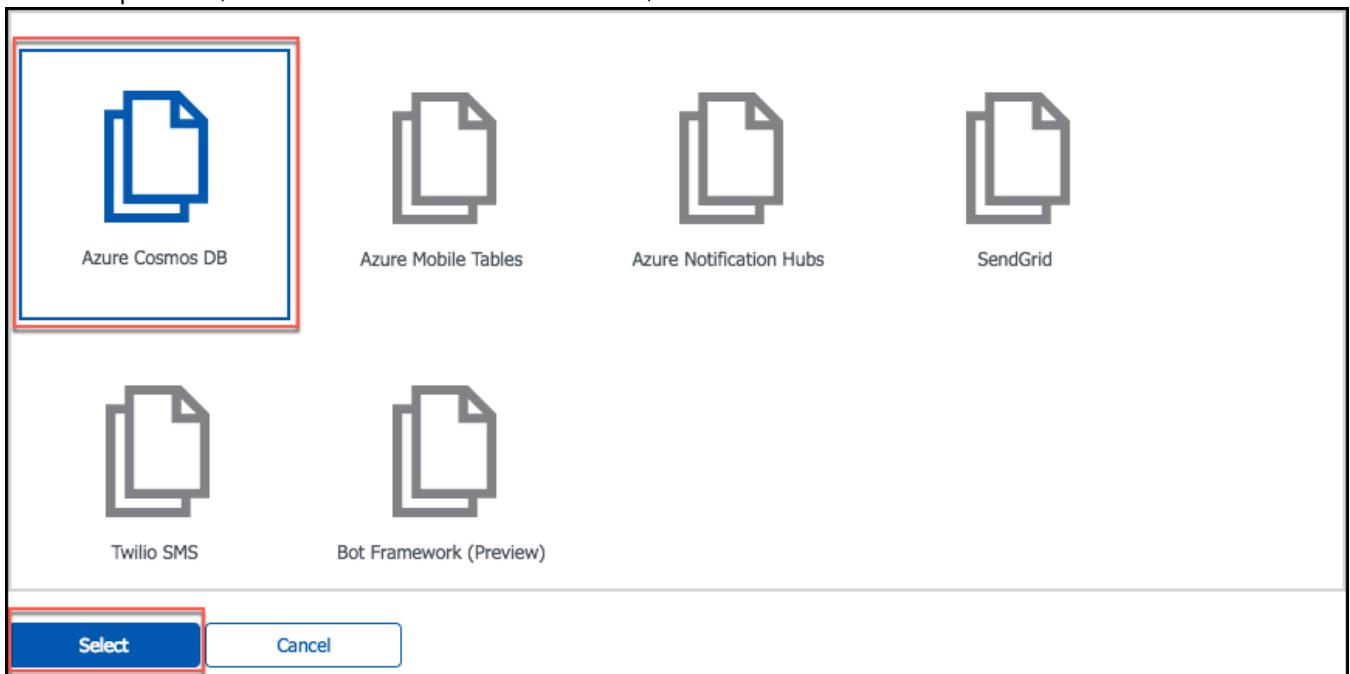
- Queue name: Enter the name of your externalization queue (from Cloud explorer in Visual Studio). This should be contosoevents-externalization-requests.



- d. Select Save.
19. While still on the Integrate screen, select +New Output.

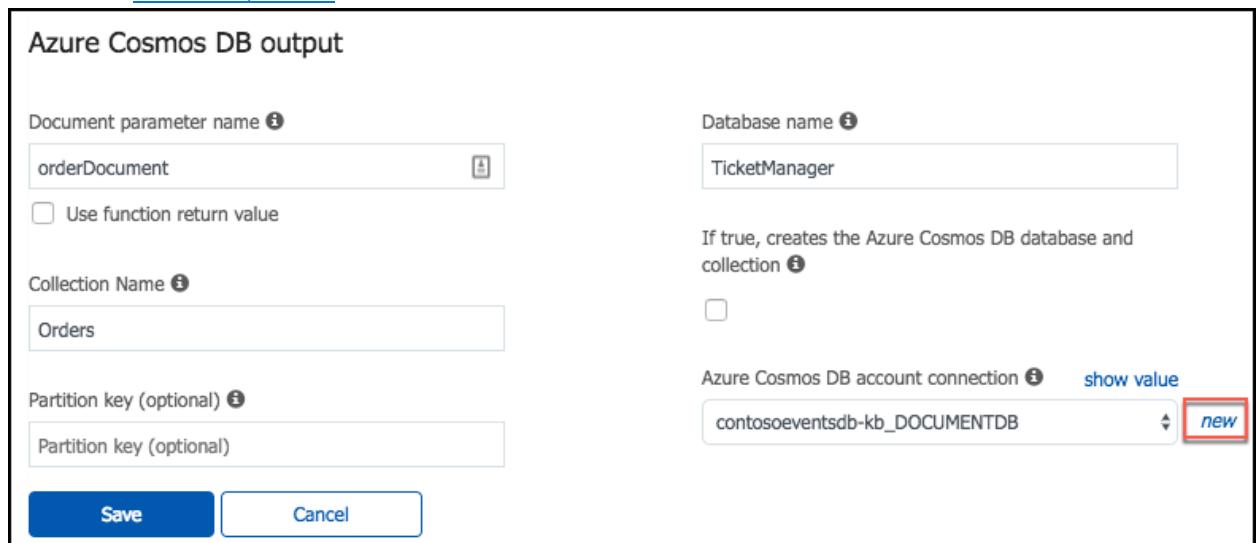


20. In the outputs box, locate and select Azure Cosmos DB, then select Select.



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 6](#).



- 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' section of 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 'orderItem' and 'orderDocument' 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. It lists several items: 'ProcessOrderExternalizations' (selected and highlighted with a red box), 'Integrate', 'Manage', and 'Monitor'.

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 displays a project structure under 'ContosoEventsPOC'. Under 'Azure Functions', there are files: 'ProcessLogMessage.cs' (highlighted with a red box) and 'ProcessTicketOrderExternalizationEvent.cs' (also highlighted with a red box). Other files like 'ProcessTicketOrderSimulationRequest.ps1' are also listed.

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' editor in the Azure portal. The code block contains the following C# code:

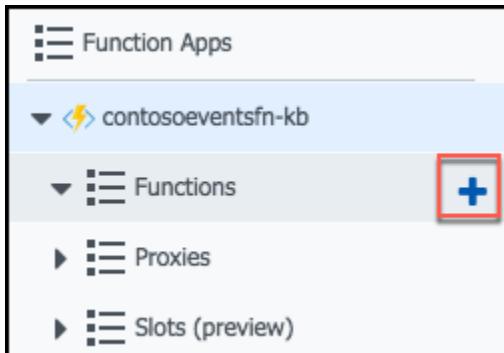
```

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 }

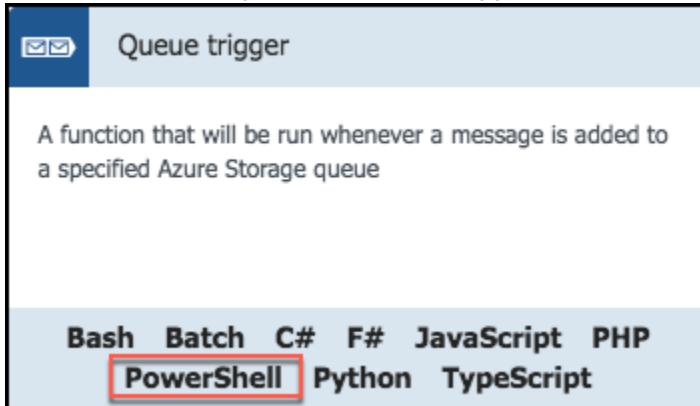
```

27. You will now create another function.

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



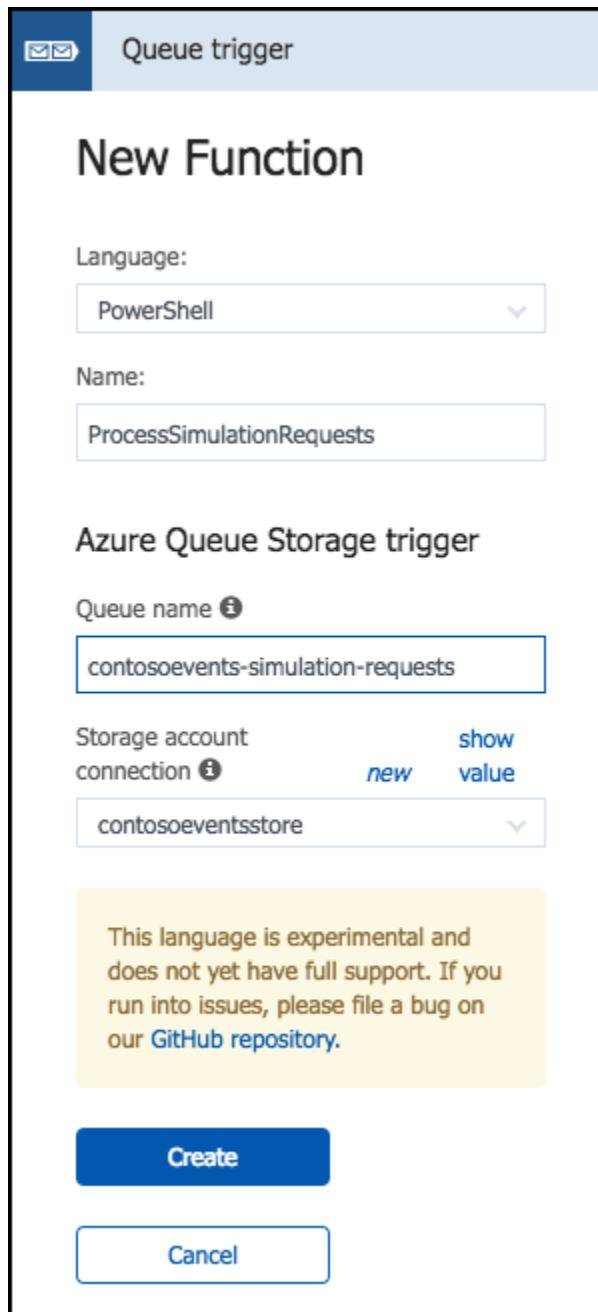
29. In the Choose a template... screen that appears, locate the Queue trigger box, and select PowerShell.



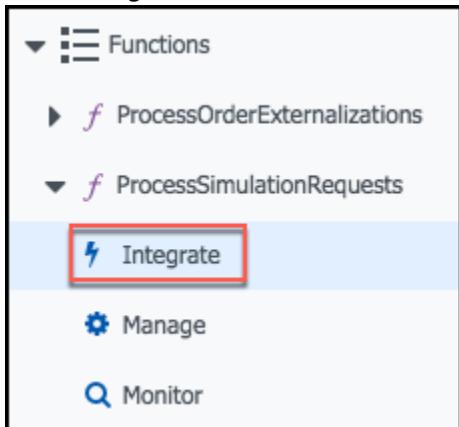
30. In the Queue trigger dialog, enter the following:

- 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.

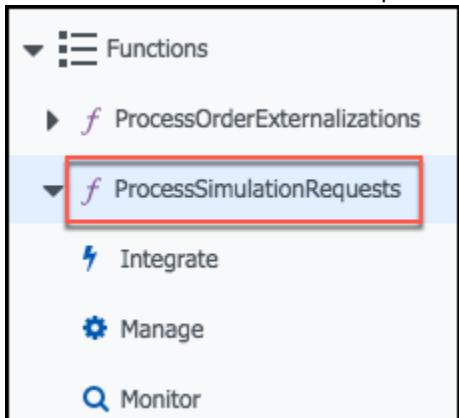


32. Make sure Azure Queue Storage is selected under Triggers, then enter the following:

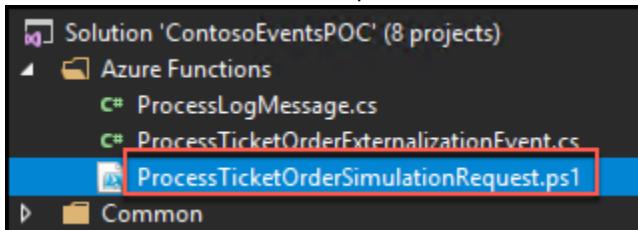
- Message parameter name: Enter simulationRequest
- Storage account connection: Leave set to contosoeventsstore
- Queue name: Leave as contosoevents-simulation-requests
- Select Save.

The image shows a screenshot of the Azure Queue Storage trigger configuration dialog. At the top, there are three tabs: 'Triggers', 'Inputs', and 'Outputs'. The 'Triggers' tab is active, showing the 'Azure Queue Storage (triggerInput)' trigger selected. The 'Inputs' tab has a '+ New Input' button. The 'Outputs' tab has a '+ New Output' button. Below the tabs, 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 a 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' panel. The 'Logs' tab is highlighted with a red box. To its right are buttons for 'Pause', 'Clear', 'Copy logs', and 'Expand'. 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-ca27f3ffff767)
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": "YYYTT6565661652612516125",
    "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-ca27f3ffff767, 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:

contosoeventsfn-kb - ProcessOrderExternalizations

Function Apps

All subscriptions

Function Apps

contosoeventsfn-kb

Functions

ProcessOrderExternalizations

Integrate

Manage

Monitor

ProcessSimulationRequests

Proxies

Slots (preview)

Success count since Dec 1st

4

Error count since Dec 1st

0

Invocation log Refresh

Function	Status	Details: Last ran (duration)
ProcessOrderExternalizations (null, {"DatabaseNa ...})	✓	7 minutes ago (0 ms)
ProcessOrderExternalizations ({"id": "eb6ea952-9d26-4d0e-86c8-f538b91d670 ...")	✓	7 minutes ago (94 ms) <span>▶</span>
ProcessOrderExternalizations ({"id": "56dab32a-41 ...")	✓	7 minutes ago (938 ms)
ProcessOrderExternalizations (null, {"DatabaseNa ...})	✓	7 minutes ago (1,300 ms)

Invocation details

Parameter	Value
orderItem	{"id": "eb6ea952-9d26-4d0e-86c8-f538b91d670 ...", "OrderDate": "2017-12-21T21:01:32.1573122+00:00", "CancellationDate": "00001", "PaymentProcessorTokenId": "YYYYT6565661652612516125", "TicketId": "EVENT1-ID-00001", "TicketOrderReceived": "2017-12-21T21:01:33.331687+00:00", "TicketStatus": "PENDING", "UserName": "johnsmith", "Email": "john.smith@gmail.com", "Tag": "Manual", "EventId": "71F9CIR86MN", "TicketNumber": "2"} orderDocument: {"DatabaseName": "TicketManager", "Collection": "TicketOrder", "TicketOrder": "TicketOrder"} log: null _context: 4b7fd8e4-e74c-491c-9662-913dbf74b507

Logs

```
Ticket Order received: {
  "id": "eb6ea952-9d26-4d0e-86c8-f538b91d670 ...",
  "OrderDate": "2017-12-21T21:01:32.1573122+00:00",
  "CancellationDate": null,
  "PaymentProcessorTokenId": "YYYYT6565661652612516125",
  "TicketId": "EVENT1-ID-00001",
  "TicketOrderReceived": "2017-12-21T21:01:33.331687+00:00",
  "TicketStatus": "PENDING",
  "UserName": "johnsmith",
  "Email": "john.smith@gmail.com",
  "Tag": "Manual",
  "EventId": "71F9CIR86MN",
  "TicketNumber": "2"
}
```

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 top navigation bar includes 'Add Collection', 'Refresh', 'Move', 'Delete Account', and 'Data Explorer'. A red box highlights the 'Data Explorer' button. On the left, there's a sidebar with 'Overview' (which is currently selected and highlighted in blue), 'Activity log', 'Access control (IAM)', and 'Tags'. The main content area displays account status ('Online'), resource group ('hands-on-labs'), subscription information, and a 'Subscription ID' field.

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 for the 'contosoeventsdb-kb' account. The top navigation bar includes 'New Collection', 'New SQL Query' (highlighted with a red box), 'New Stored Procedure', 'New UDF', 'New Trigger', 'Delete Collection', and a trash bin icon. The left sidebar has 'Data Explorer' selected. The main area shows the 'COLLECTIONS' section with 'TicketManager' expanded, showing 'Orders' (highlighted with a red box) and other options like 'Documents', 'Scale & Settings', etc. The 'Orders' section shows 'Stored Procedures', 'User Defined Functions', 'Triggers', 'Events', and 'ToDoList'. To the right, the 'Query 1' window contains the SQL query: '1 SELECT \* FROM c WHERE c.id = '56dab32a-4154-4ea5-befa-2eb324e142ee''. A red callout box points to this query text with the instruction 'replace with your order id'. Below the query, the results are shown: 'Results: 1 - 1 | Request Charge: 2.32 RUs | →'. The result pane shows a single document with fields: 'id', 'OrderDate', 'FulfillDate', and 'CancellationDate'.

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

## Exercise 4: Publish the Service Fabric Application

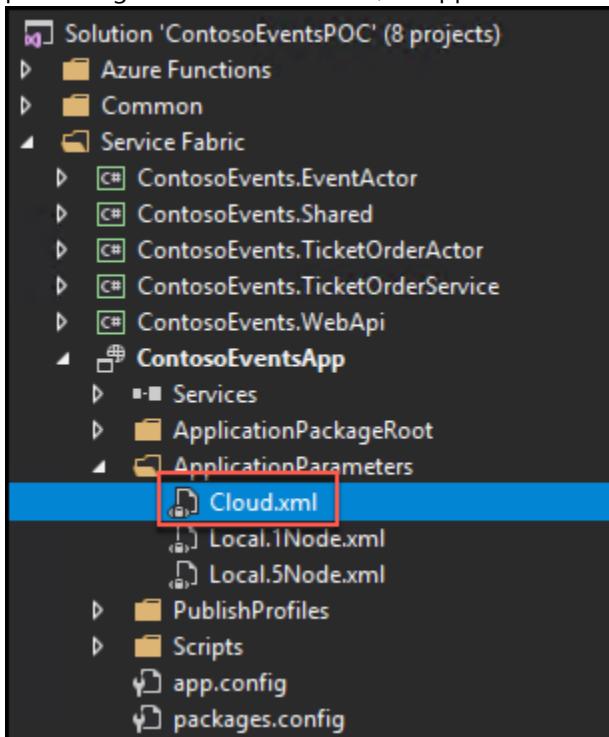
Duration: 15 minutes

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.

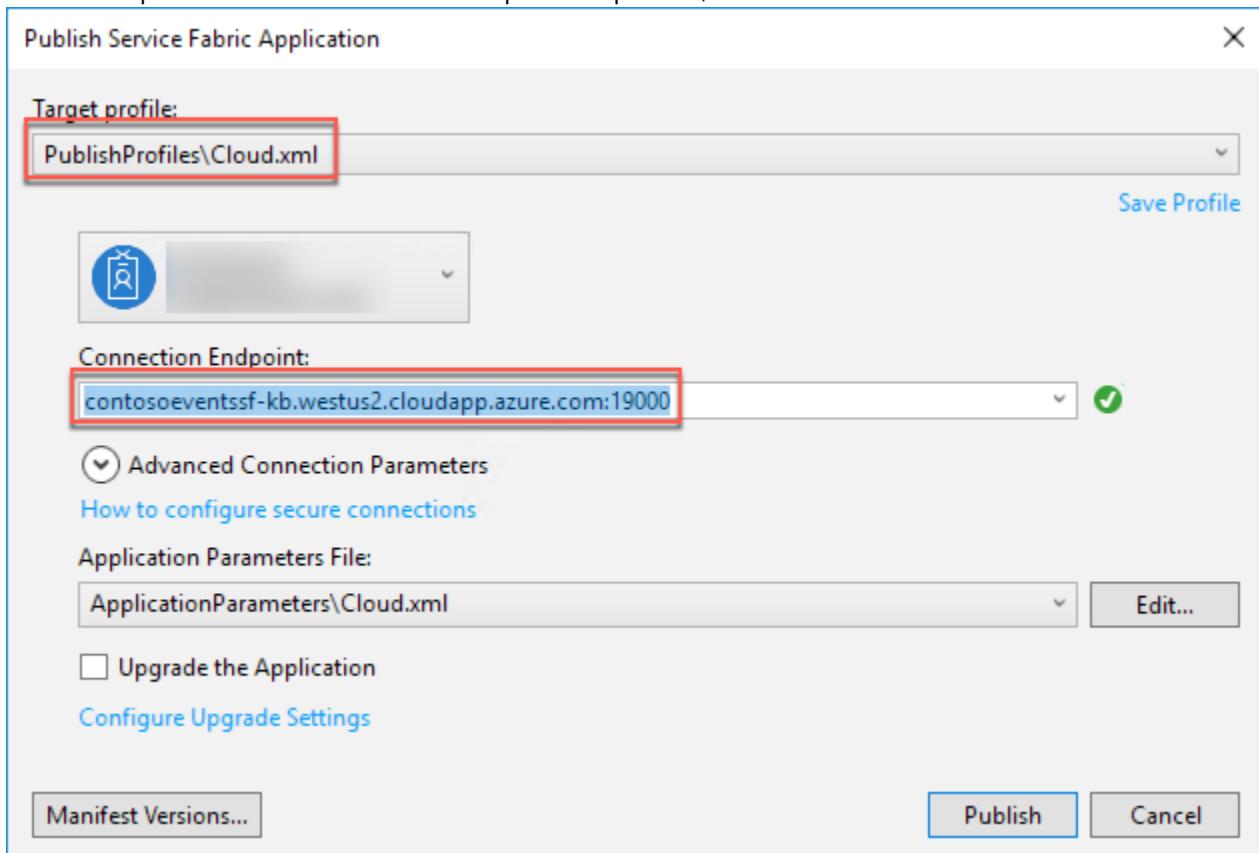
1. In Visual Studio on your Lab VM, within Solution Explorer, open Cloud.xml from the ApplicationParameters folder of the ContosoEventsApp project, under the Service Fabric folder. This file contains parameters we can use for publishing to the hosted cluster, as opposed to the local cluster.



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.
- ```
<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>, and an 'api\_key' input field. Below the navigation, there are three main categories: Admin, Events, and Orders. The Orders category is currently selected and has a red border around it. Under Orders, there are four GET methods listed: /api/orders/user/{username}, /api/orders/event/{eventid}, /api/orders/{id}, and /api/orders/stats. Below these is a POST method: /api/orders. The 'POST /api/orders' section is also highlighted with a red box around its 'Value' field. To the right of the 'Value' field, there is a detailed JSON schema for the 'order' parameter, which is marked as '(required)'. The schema includes fields like id, orderDate, fulfillDate, cancellationDate, userName, email, tag, eventId, paymentProcessorTokenId, paymentProcessorConfirmation, and tickets. A note at the bottom of the schema says '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 where the JSON should be pasted. The `Try it out!` button at the bottom left is also highlighted with a red box.

**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",     "Tag": "Manual",     "EventId": "EVENT1-ID-00001",     "PaymentProcessorTokenId":         "YYYTT6565661652612516125",     "Tickets": 3 }</pre>		body	Model   Model Schema

Parameter content type: `application/json`

**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 the 'Try it out!' interface for a POST request to the '/orders' endpoint. The request body is a JSON object containing user information and event details. The response shows a single item in the 'Response Body' section with the ID '3bb85105-55d4-4d04-b689-8085b0dfe9ee'. The 'Response Code' is 200. The 'Response Headers' section shows standard HTTP headers including 'Content-Type: application/json; charset=utf-8'.

```

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 the Azure Cosmos DB portal. The 'TicketManager' database is selected, and the 'Orders' collection is highlighted. A new SQL query is being run named 'Query 1'. The query is: 'SELECT \* FROM c WHERE c.id = '3bb85105-55d4-4d04-b689-8085b0dfe9ee''. The results show one document with the specified ID, matching the structure shown in the previous screenshot.

New Collection New SQL Query New Stored Procedure New UDF New Trigger Delete Collection

COLLECTIONS <

- TicketManager
  - Orders
  - 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 5: API Management

Duration: 15 minutes

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

### Task 1: Import API

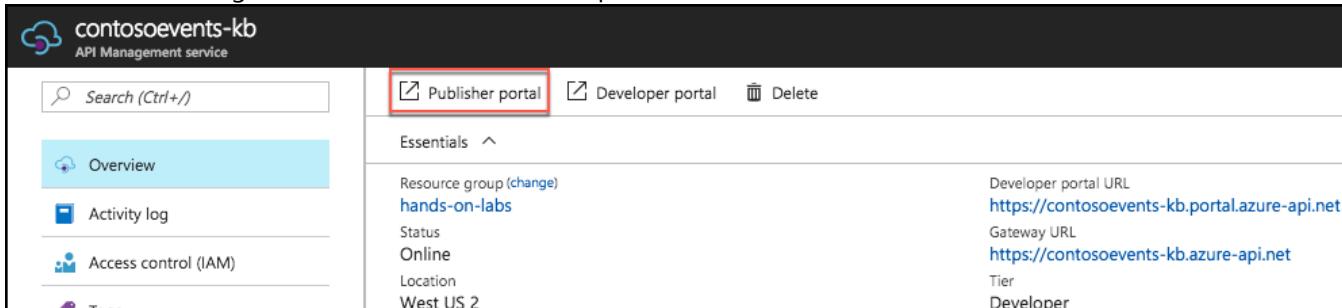
In this task, you will import the Web API description to your API Management service to create an endpoint.

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



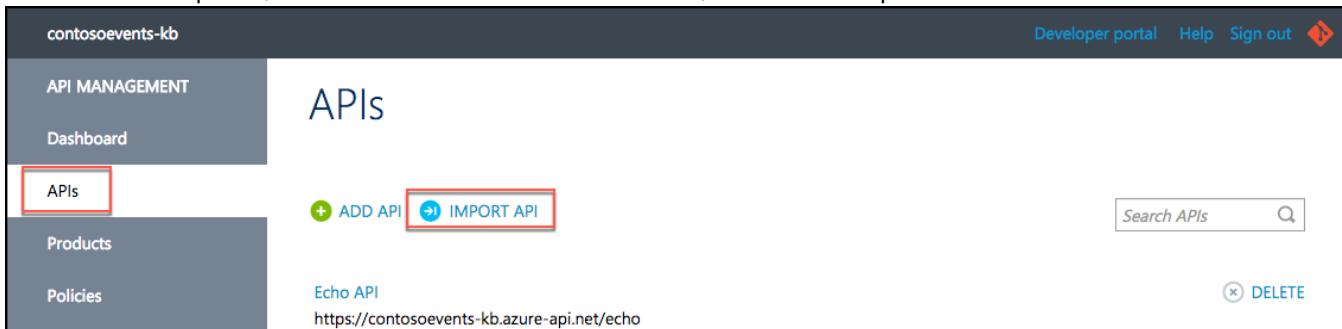
<input type="checkbox"/>	 contosoeventskb	Storage account	West US 2
<input type="checkbox"/>	 contosoevents-kb	API Management service	West US 2
<input type="checkbox"/>	 contosoeventsplan-kb	App Service plan	West US 2

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



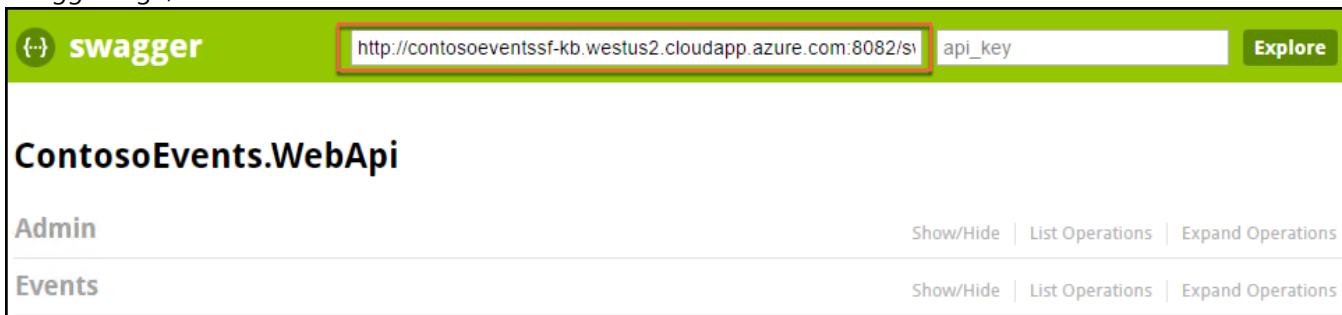
The screenshot shows the 'contosoevents-kb' API Management service blade. The left sidebar has 'Overview' selected. The main area shows 'Resource group (change) hands-on-labs' with 'Status Online' and 'Location West US 2'. To the right, it lists 'Developer portal URL https://contosoevents-kb.portal.azure-api.net' and 'Gateway URL https://contosoevents-kb.azure-api.net'. The 'Publisher portal' button in the toolbar is highlighted with a red box.

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



The screenshot shows the 'APIs' section of the Publisher portal. The left sidebar has 'API MANAGEMENT' selected, with 'APIs' highlighted with a red box. The main area shows a list of APIs, with 'Echo API' and its URL 'https://contosoevents-kb.azure-api.net/echo' listed. The 'IMPORT API' button 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 'ContosoEvents.WebApi'. The URL 'http://contosoeventssf-kb.westus2.cloudapp.azure.com:8082/swagger' is highlighted with a red box in the address bar. The main area shows the API documentation for 'ContosoEvents.WebApi' with 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 'API MANAGEMENT' section of the Azure portal. On the left, there's a sidebar with 'Dashboard', 'APIs', 'Products', 'Policies', and 'Analytics'. The main area is titled 'APIs - ContosoEvents.WebApi'. Below the title, there are tabs for 'Summary', 'Settings', 'Operations', 'Security', 'Issues', and 'Products'. A red box highlights the 'ContosoEvents.WebApi' entry, which includes the URL 'https://contosoevents-kb.azure-api.net/events'. At the bottom, there are time filters: 'Today', 'Yesterday', 'Last 7 Days', 'Last 30 Days', and 'Last 90 Days'. On the right, there's a '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 'API Management service' overview blade. On the left, there's a sidebar with 'Search (Ctrl+ /)', 'Overview' (which is highlighted with a blue background), 'Activity log', 'Access control (IAM)', and 'Tags'. The main area shows 'Essentials' with a 'Resource group (change)' dropdown set to 'hands-on-labs', 'Status' as 'Online', 'Location' as 'West US 2', and developer portal URLs: 'Developer portal URL' as 'https://contosoevents-kb.portal.azure-api.net' and 'Gateway URL' as 'https://contosoevents-kb.azure-api.net'. On the right, there are buttons for 'Delete', 'Developer portal', and 'Edit'.

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

The screenshot shows the 'Contoso Events API' developer portal. At the top, there's a navigation bar with 'HOME', 'APIS', 'PRODUCTS', 'APPLICATIONS', 'ISSUES', and an 'ADMINISTRATOR' dropdown. The 'ADMINISTRATOR' dropdown is expanded, showing 'MANAGE PROFILE' (which is highlighted with a red box) and 'SIGN OUT'. Below the navigation, a blue banner says 'Welcome to the developer portal! Administrators can manage the APIs and customize the content in this portal. Learn more'.

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

The screenshot shows the 'Your subscriptions' page. It has a table with 'Subscription details' and 'Product' columns. The table contains two rows:

Subscription name	Product
Starter (default)	Rename Starter
Primary key	Show   Regenerate
Secondary key	Show   Regenerate
Unlimited (default)	Rename Unlimited
Primary key	Show   Regenerate (highlighted with a red box)
Secondary key	Show   Regenerate

- Save this key for next steps.

Subscription name	Unlimited (default)	Rename
Primary key	499c129b97fa448e94e78f48cbac3858	<a href="#">Hide</a>   <a href="#">Regenerate</a>
Secondary key	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	<a href="#">Show</a>   <a href="#">Regenerate</a>

- You now have API Management application key you will need to configure the Function App settings.

## 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.

- From the Azure Portal, browse to the Function App.
- You will create an Application setting for the function to provide it with the API Management consumer key.
- 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 a function app named "contosoeventsfn-kb". The left sidebar lists "Function Apps" and the selected app "contosoeventsfn-kb". Below the sidebar are links for "Functions", "Proxies", and "Slots (preview)". The main area has tabs for "Overview" and "Platform features", with "Overview" selected. It displays the status as "Running" and subscription information: "Solliance MVP MSDN" and "Subscription ID 30fc406c-c745-44f0-be2d-63b1c860cde0". The "Configured features" section contains two items: "Function app settings" and "Application settings", 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.

**Application settings**

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

+ Add new setting

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



## Exercise 6: Configure and publish the web application

Duration: 15 minutes

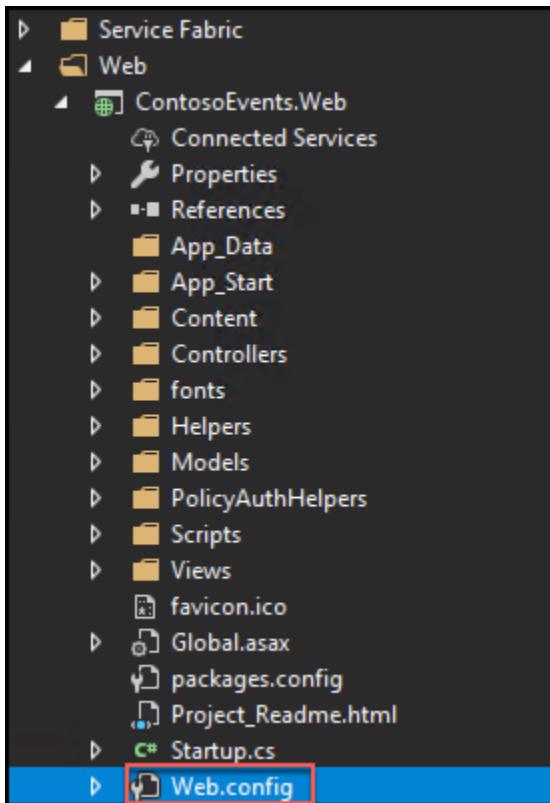
In this exercise, you will configure the website to communicate with the API Management service, deploy the application, and create an order.

### 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.

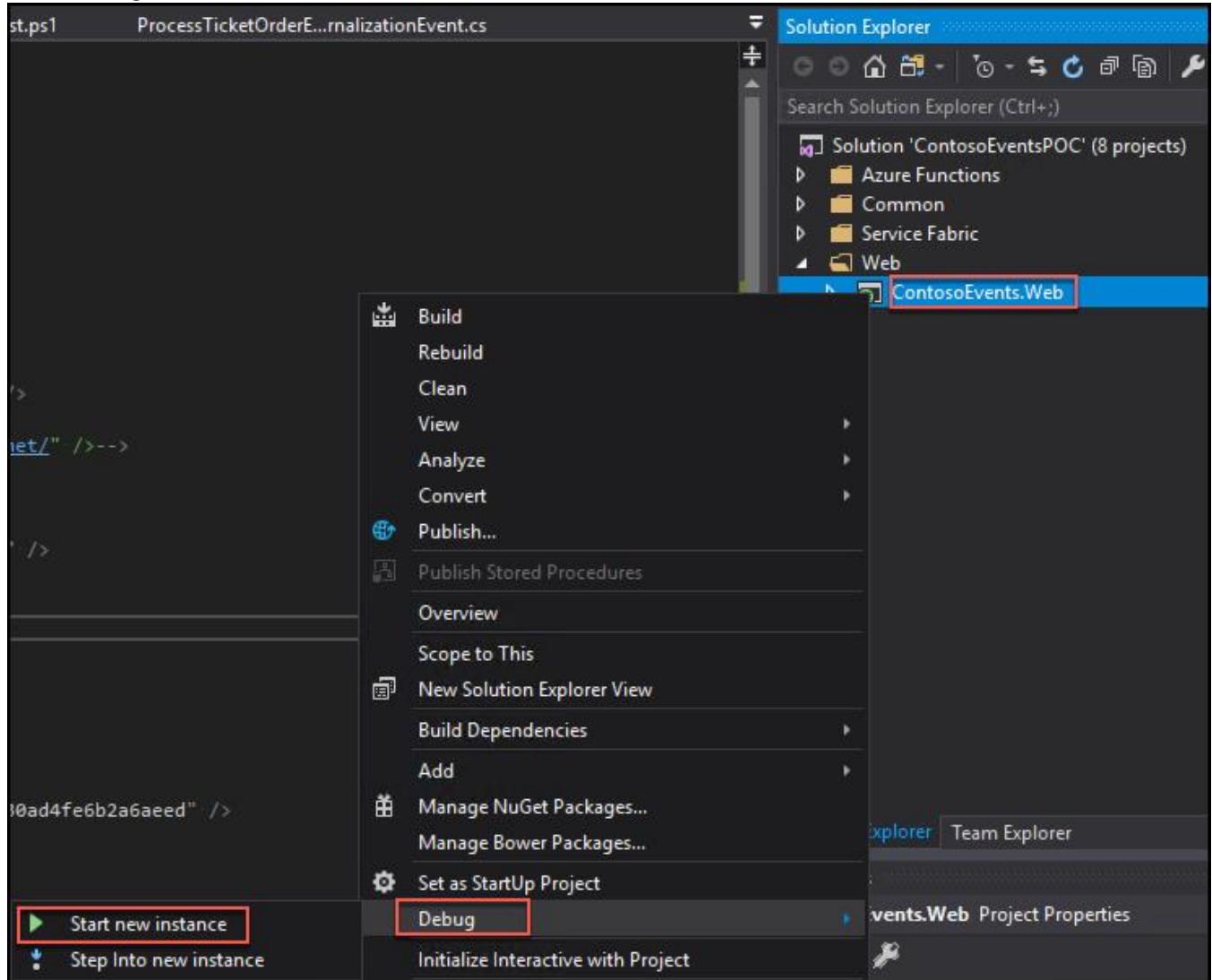
3. For the apimng:SubscriptionKey key, enter the subscription key you revealed in API Management developer portal ([Exercise 5, Task 2, Step 4](#)).
4. Save Web.config. You should have values for two of the API Management app settings.

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

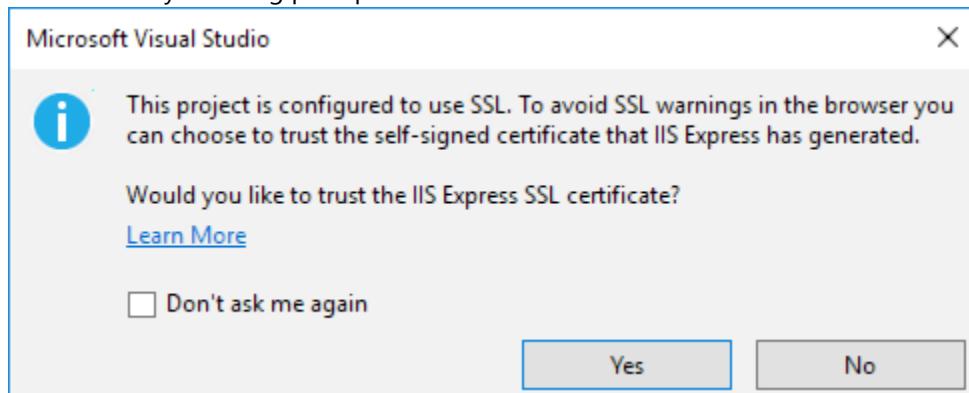
## Task 2: Running the web app and creating an order

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

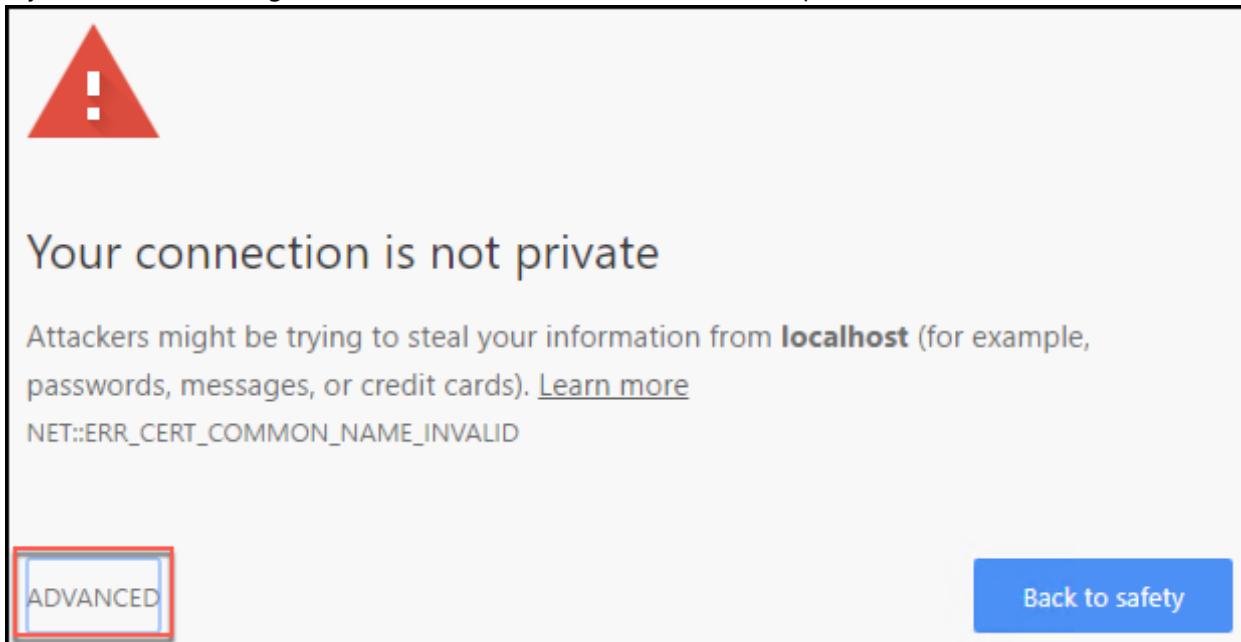
1. Using Solution Explorer in Visual Studio, expand the Web folder, then right-click the ContosoEvents.Web project, select Debug, and then Start new instance.



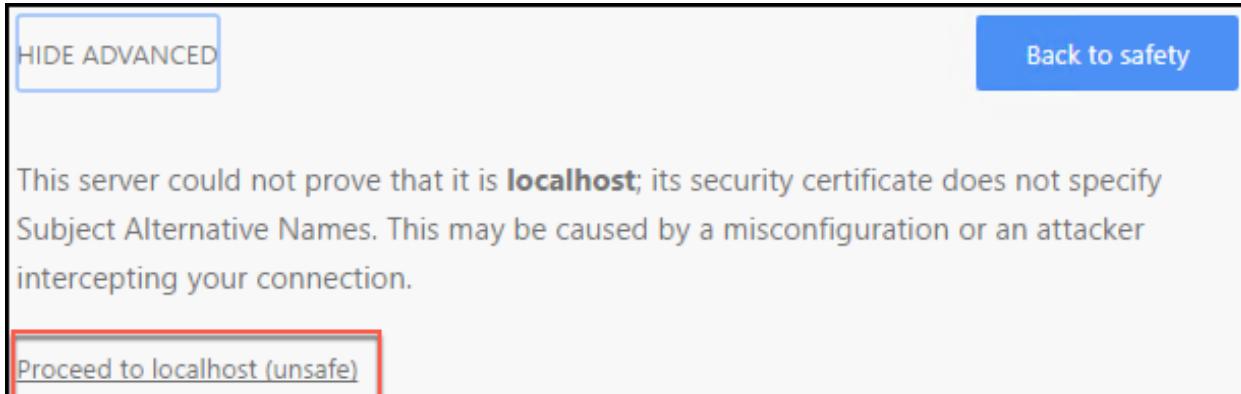
2. If prompted about whether you would like to trust the IIS Express SSL certificate, select Yes, then select Yes again at the Security Warning prompt.



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 overlay indicating a price of "25 USD". 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/may/2016

Time : 04:36

PricePerTicket : 25 USD

8. Enter values into the empty fields for your email, first name, last name, and Cardholder name.

<b>Billing</b>	<b>Credit Card</b>
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.

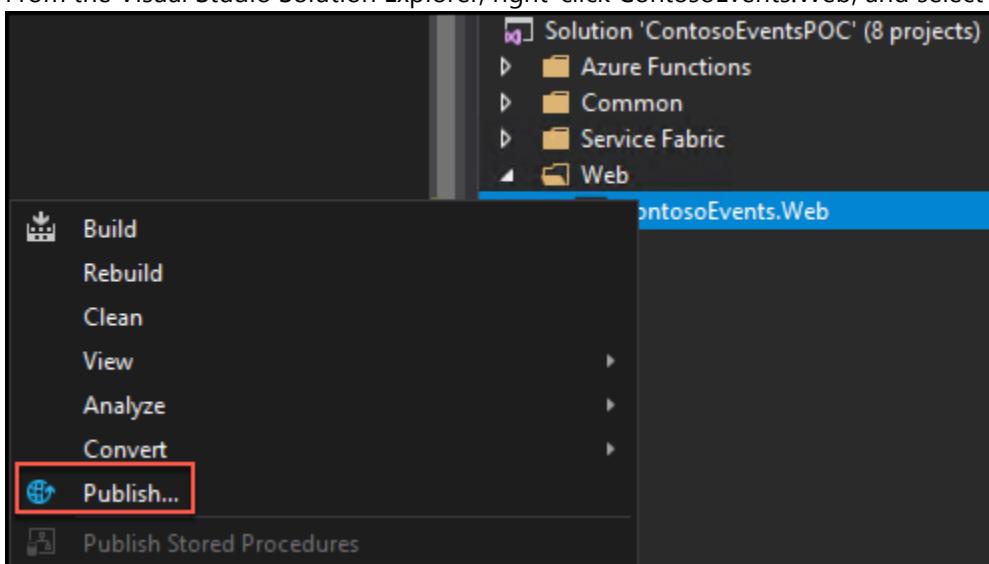


11. Close the web browser to stop debugging the application.

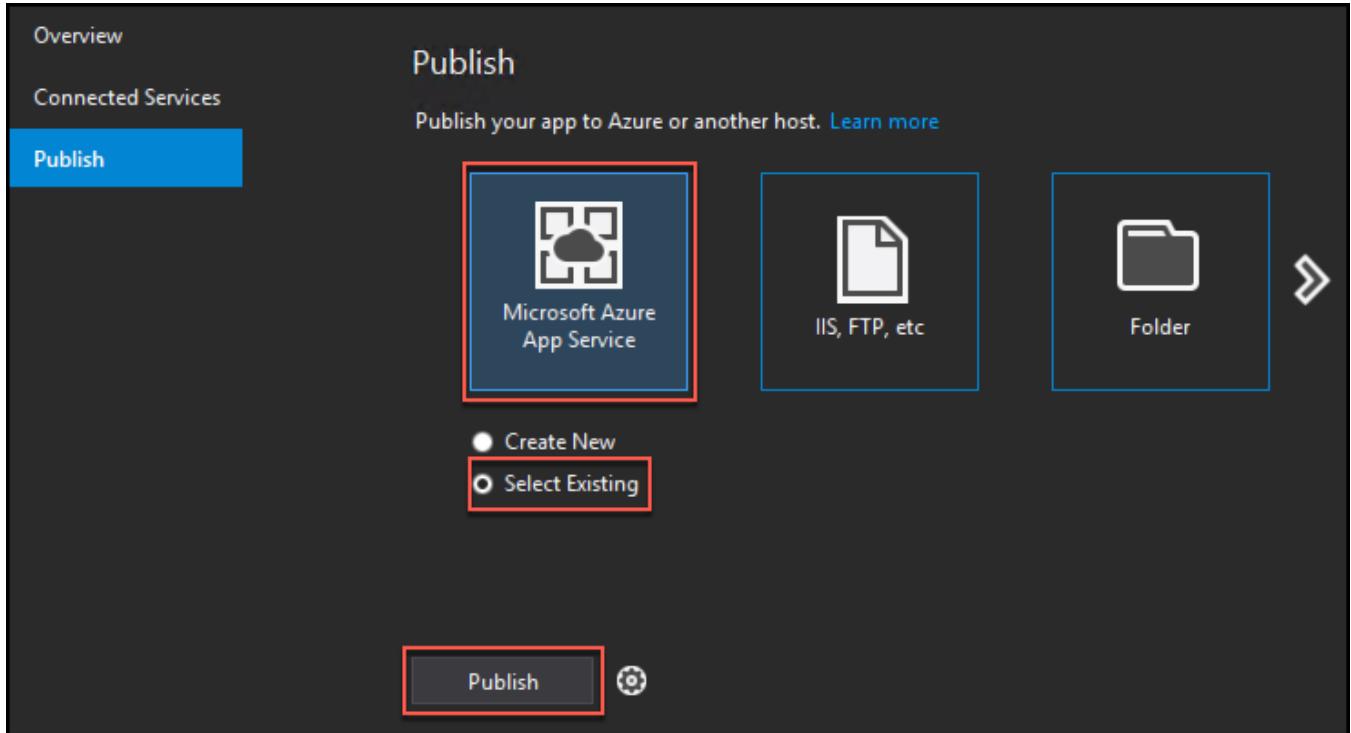
### Task 3: Publish the web app

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.

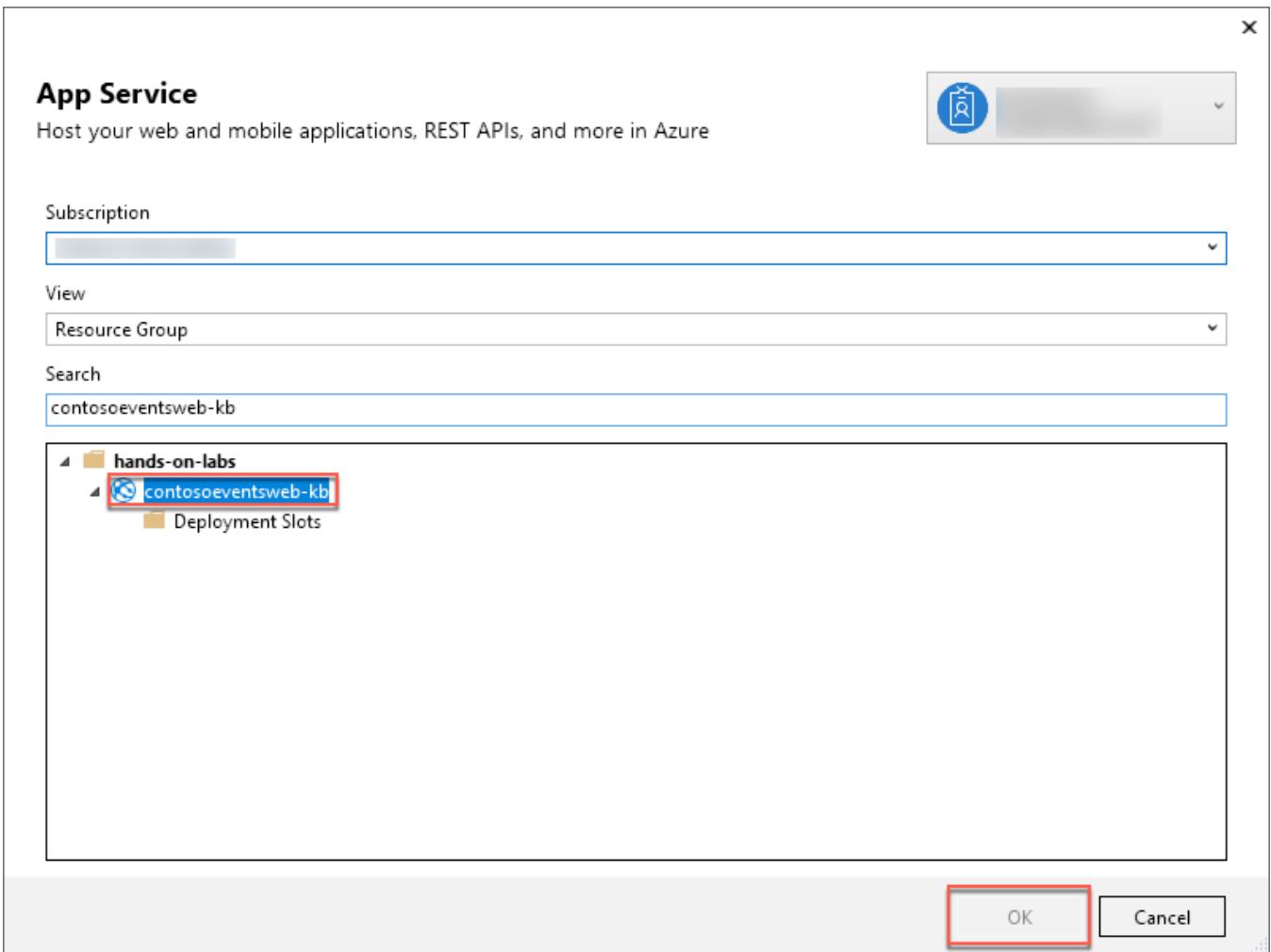


2. Select the Microsoft Azure App Service tile, choose Select Existing, then select Publish.

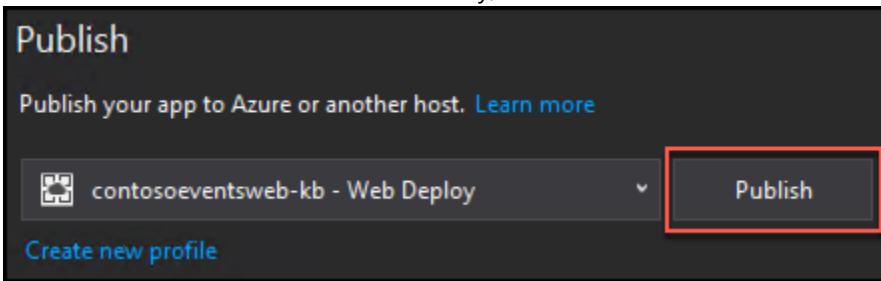


3. You may be prompted to log in to your Microsoft Account with access to the subscription where you created the resources for this hands-on lab. After logging in, you can select the subscription in the App Service screen.

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 7: 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.

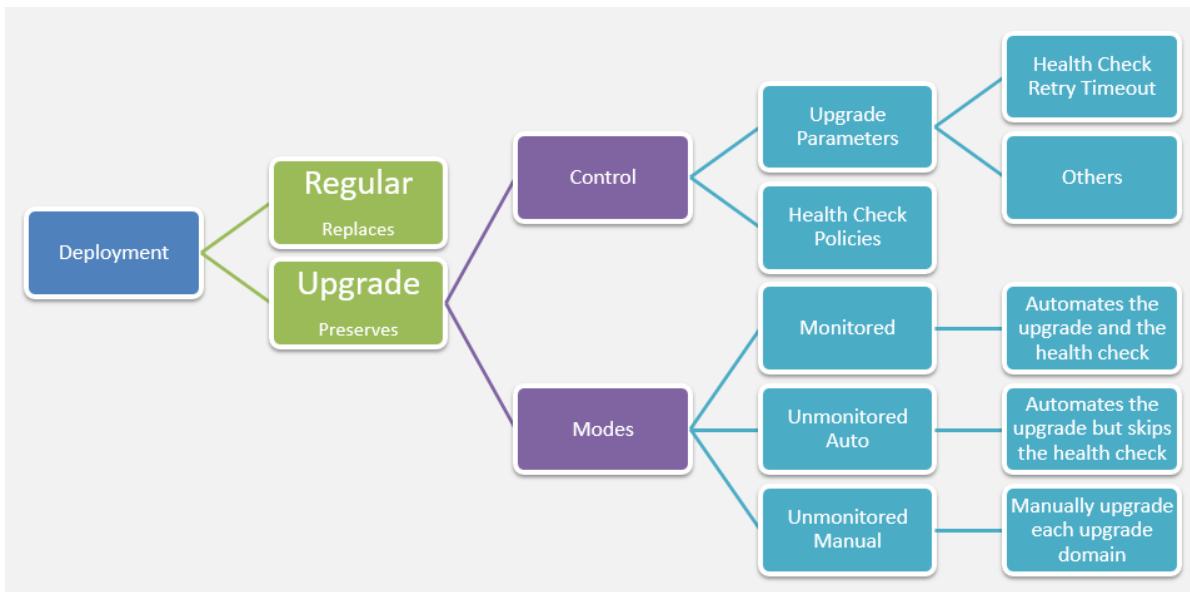
This task will illustrate the mechanism that Service Fabric provides for upgrading an application in production.

### 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:

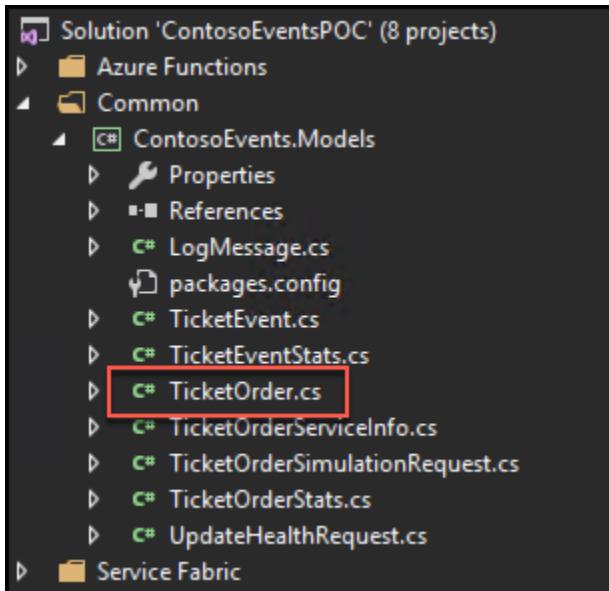


If you set the upgrade mode to Monitored, Service Fabric will be in full control of the upgrade process. There is a configurable time to wait after the upgrade has finished before Service Fabric evaluates the health of the application. This value defaults to 600 seconds.

## 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; }
```

3. and

```
//TODO: Exercise 6 - Task 1  
public enum OrderStatuses  
{  
    Fulfilled,  
    TicketsExhausted,  
    CreditCardDenied,  
    Cancelled,  
    Invalid  
}
```

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

6. 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;
```

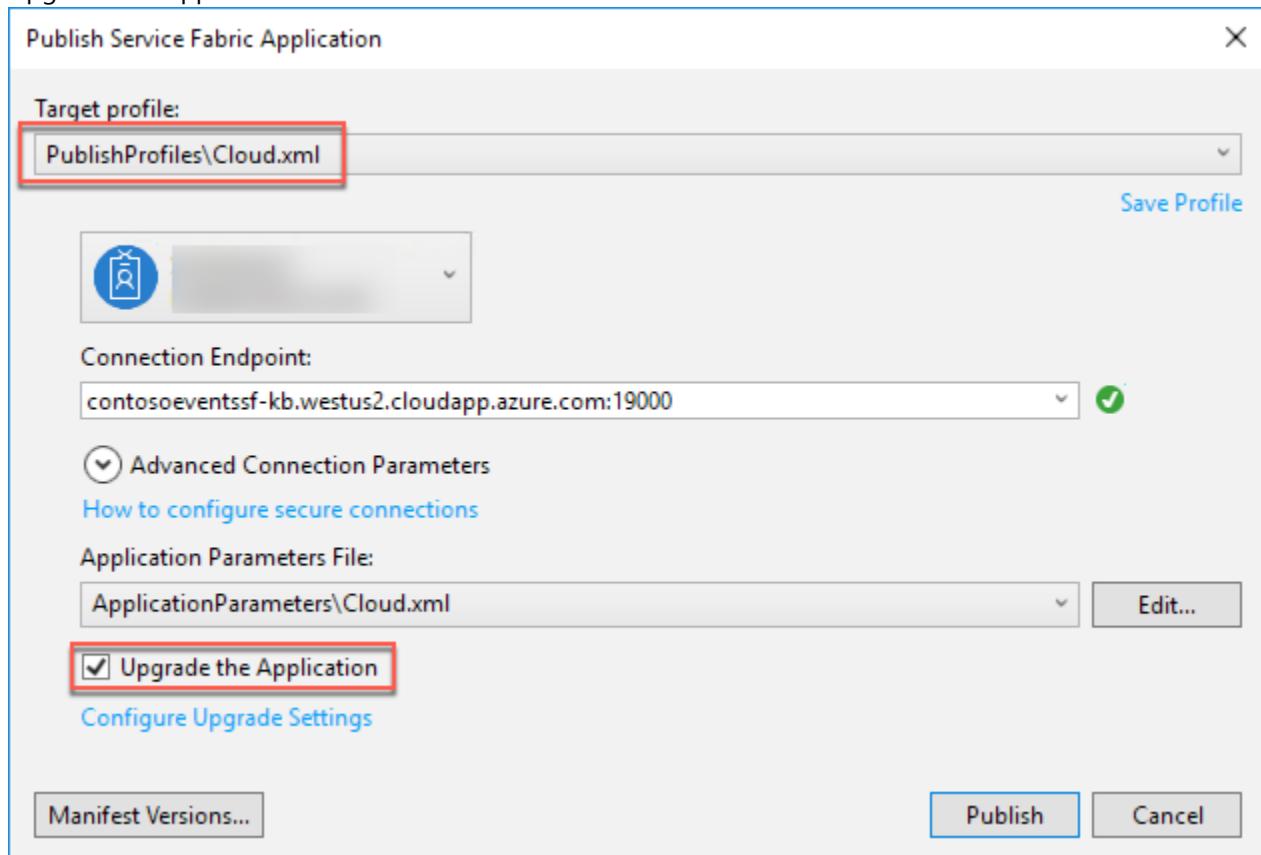
7. Save TicketOrderActor.cs.
8. After adding this field, the actor will now save it with each ticket order.
9. After making this change, rebuild the solution (Build menu, Rebuild solution), and verify that there are no errors.

### Task 3: Perform a smooth upgrade

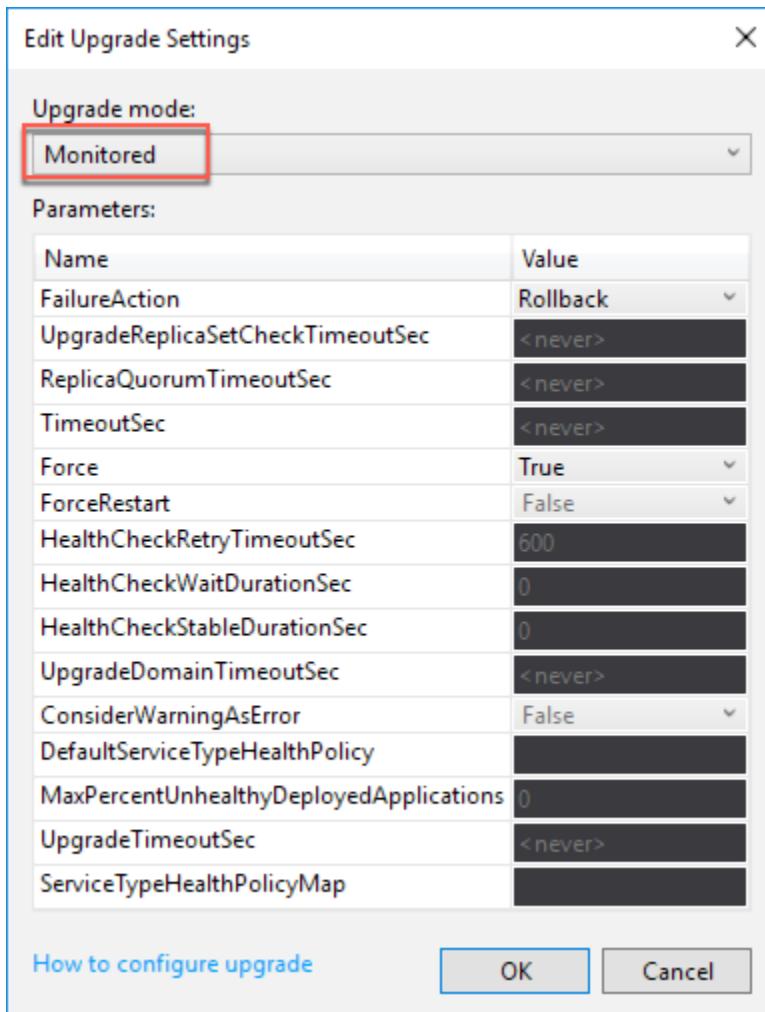
In this task, you will configure settings for the Service Fabric application to perform an upgrade.

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.

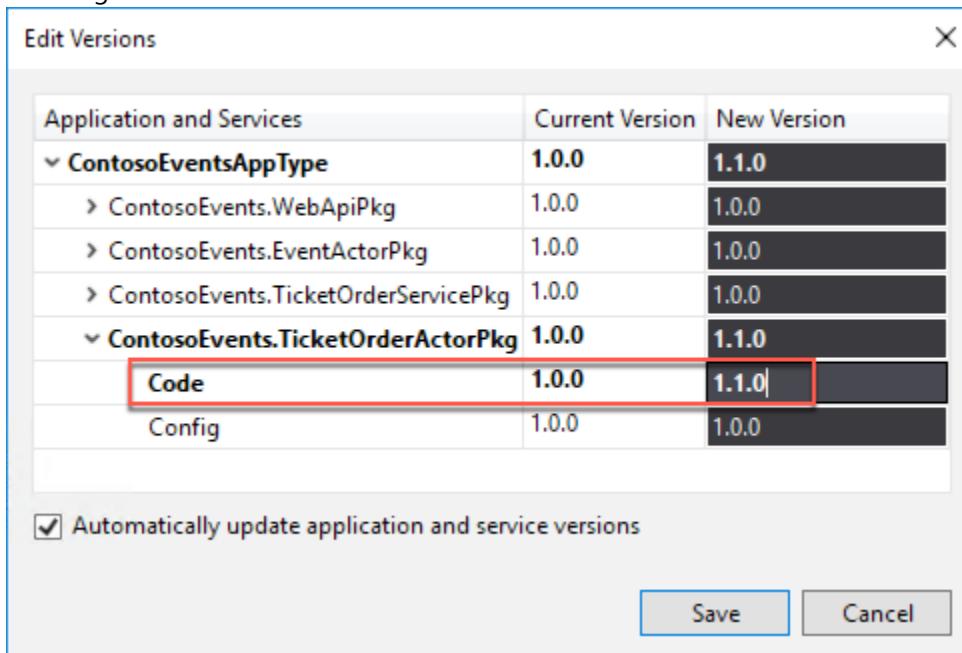


3. Select Configure Upgrade Settings, under Upgrade the Application. Select Monitored for the upgrade mode, then select OK.

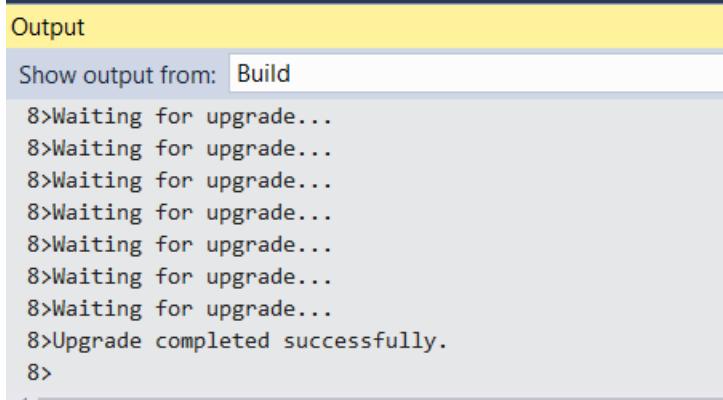


4. From the Publish Service Fabric Application dialog, select Manifest Versions.

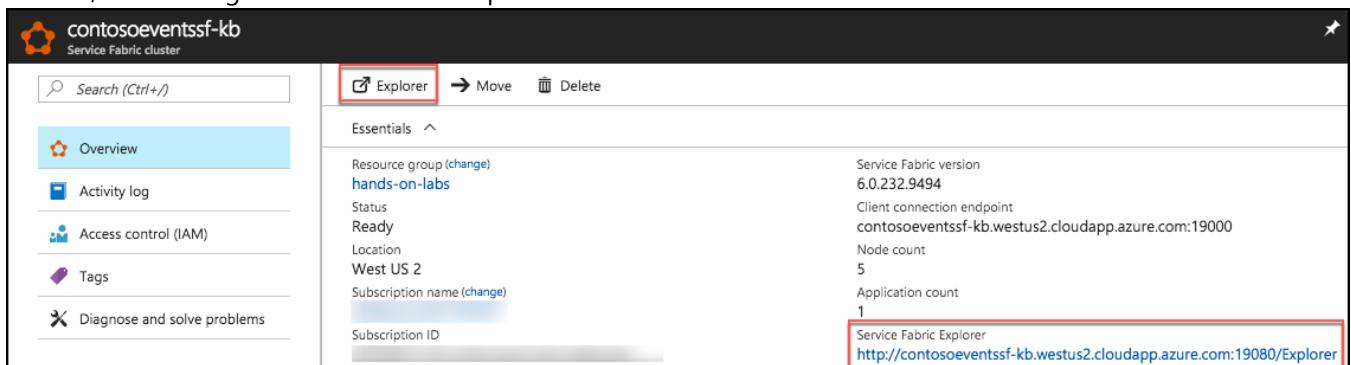
5. Change the TicketOrderActorPkg\Code New Version to 1.1.0. This change will force the actor package and the app to change to 1.1.0 as well.



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.



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.

The screenshot shows the Service Fabric Explorer interface for the application 'fabric:/ContosoEventsApp'. At the top, it displays the application's name, health state (OK), and status (Upgrading). A red arrow points to the 'Status' field. Below this, a section titled 'UPGRADE IN PROGRESS' shows the current version (1.0.0) and target version (1.1.0) in a table, with a progress bar indicating the upgrade progress from 0 to 1. A link 'Show upgrade details' is present. At the bottom, there is a section for 'UNHEALTHY EVALUATIONS'.

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 selected. The application's name, health state (OK), and status (Ready) are displayed. To the right, a box highlights the 'Application Type' (ContosoEventsAppType) and 'Version' (1.1.0). Below this, the 'UNHEALTHY EVALUATIONS' section shows no items. The 'SERVICES' section lists four services: EventActorService, TicketOrderActorService, TicketOrderService, and WebApi, each with its service type, version (1.0.0 or 1.1.0), service kind (Stateful or Stateless), and health state (OK). A red box highlights the 'Version' column for the services.

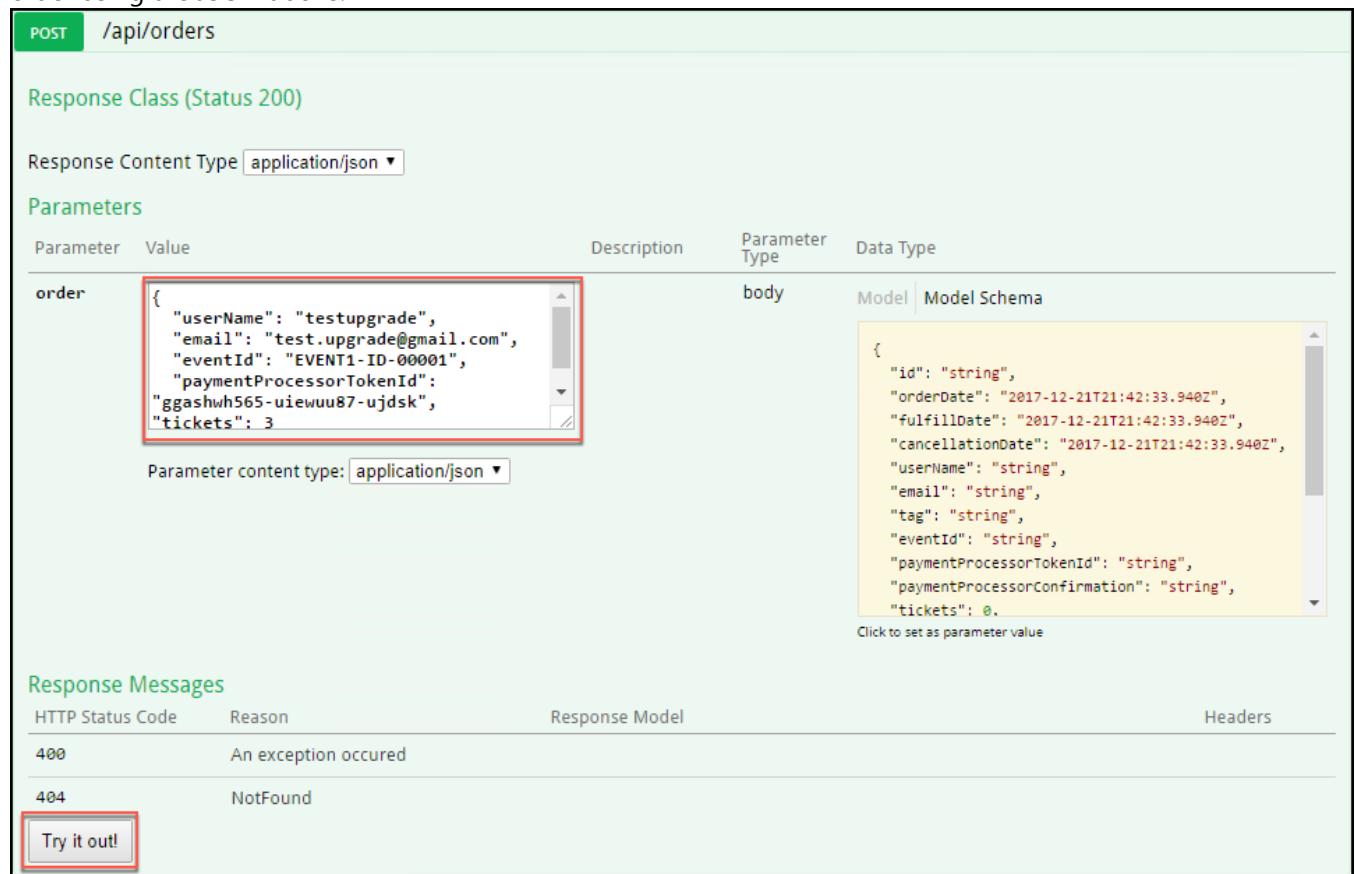
## Task 4: Submit a new order

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.



The screenshot shows the Swagger UI interface for a POST request to the /api/orders endpoint. The request body is set to the JSON provided in the text block above. A red box highlights the "Try it out!" button at the bottom left.

**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},
```

## 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. You should follow all steps provided after attending the Hands-on lab.

### Task 1: Delete the resource group

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.