

Azure Apps Lab

MANUAL

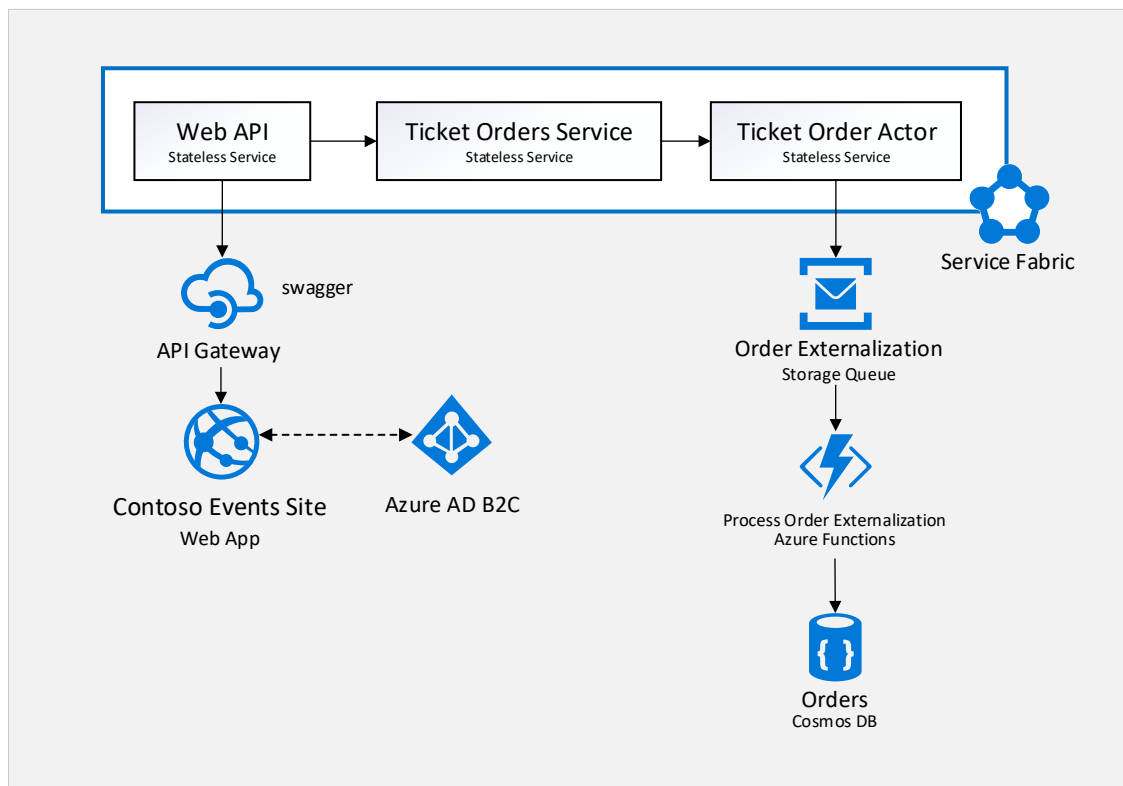
Overview

Contoso Events is a SaaS provider with 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 lab, you will construct and test an end-to-end POC for ticket ordering to demonstrate to Contoso Events a PaaS deployment. You will create resources that include the following services:

1. Service Fabric
2. API Management
3. Function Apps
4. App Services (Web App)
5. Storage Queues
6. Cosmos DB (Documents)
7. Azure Active Directory B2C

Solution Architecture



Application Architecture

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.

Prerequisites

Prior to beginning the lab, download and install the following:

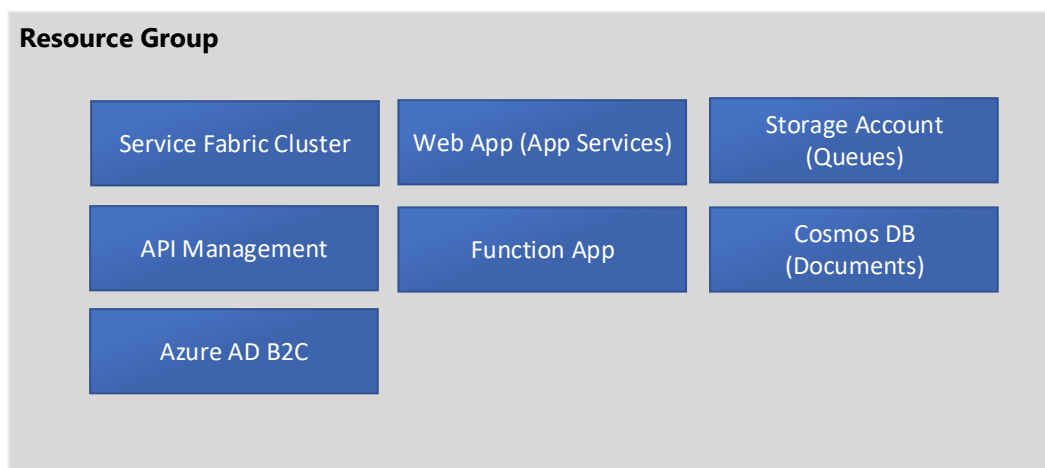
- [Visual Studio Code](#)
- [Azure Service Fabric SDK](#)
- Azure PowerShell 5.0.0 or later (if you have Windows 10, you have it)
- A browser (such as Chrome or Firefox)
(Internet Explorer **does not** work with Swagger commands)

Exercise 1: Setup

Duration: 15 minutes (10 minutes or more provisioning time)

Contoso Events has provided their app binaries for you. They have asked you to use this for deploying 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 deploy things. The following figure illustrates the resource group and resources you will create.



Task 1: Download and extract application bits

1. Download the zip file: <https://tinyurl.com/y9bwthyw>
2. Unzip the contents to your desktop to a folder you can use for the remainder of the lab.

Task 2: Deploy Resources

ARM templates have been created in advance for you stand up all that's needed in the lab. This simulates what a real-world deployment of Contoso Events cloud resources and demonstrates how an application can be quickly and easily deployed to Azure to create full environments in minutes.

During the deployment, you will create an Azure Resource Group that will hold all items for this exercise.

Important: This approach will make it easier to clean up later. Because you will want to include the Service Fabric Cluster that you create in the same Resource Group as other resources you create during the remaining exercises.

In this deployment, the following resources will be created:

- Service Fabric Cluster (5 nodes)
- API Management account
- Function App
- Web App (with App Service Plan)
- Storage Account (for Storage Queues)
- Cosmos DB Account

NOTE: You will manually create an Azure Active Directory B2C tenant for authentication. IMPORTANT: Most Azure resources require unique names. You will be asked to provide an *alias* during resource deployment. This will be used as the suffix to guarantee uniqueness.

Tasks to complete

8. In your browser, navigate to the **Azure Portal** <https://portal.azure.com>
9. Next, in a **new browser tab**, navigate to the ARM template:
<https://github.com/kevinhillinger/azure-apps-lab/tree/master/deploy>
10. Click on the Deploy to Azure button:



11. Select the subscription you will use for all the steps during the lab.
12. Create a new Resource Group named **contoso-events**
13. Select **West US 2** for the Resource Group **Location**
14. Fill in your company **Alias** (replacing <your_alias> in the text box)
15. Fill in your email address for the **Api Admin Email**
16. Leave the admin username and password as-is. The values are:

appsadmin | c0ntosoEven+s


Your screen should look similar to the following screenshot:

Home > Custom deployment

Custom deployment

Deploy from a custom template

TEMPLATE

 Customized template
6 resources

[Edit template](#) [Edit parameters](#) [Learn more](#)

BASICS

* Subscription ▼

* Resource group Create new Use existing

✓

* Location ▼

SETTINGS

Alias ?

Admin Username ?

Admin Password ?

Api Admin Email ?

17. Check the "I agree to the terms and conditions stated above" (you may need to scroll down)
18. Check Pin to Dashboard
19. Click Purchase

Exit criteria

NOTE: the deployment may take up to 10 minutes, so you may proceed to the next task, and return later to verify.

- Your Resource Group **contoso-events** is listed in the Azure Portal.


Resource groups
Microsoft

[+](#) Add [Assign Tags](#) [Columns](#) [Refresh](#)

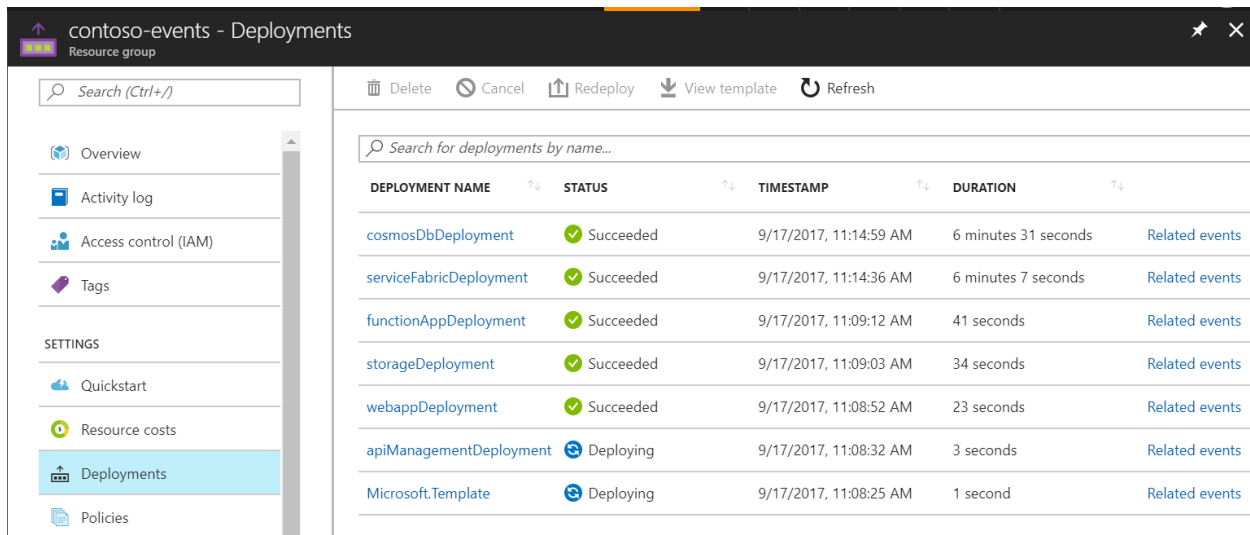
Subscriptions: 1 of 3 selected – Don't see a subscription? [Switch directories](#)

▼ ▼ ▼

1 items

<input type="checkbox"/>	NAME ↑↓	SUBSCRIPTION ↑↓	LOCATION ↑↓	
<input type="checkbox"/>	 contoso-events	Internal Subscription (Me)	East US 2	...

- Click on **contoso-events** in the list
- Then, click on **Deployments**
- Confirm that the deployment is successful



20. Verify the status of each deployment as **Succeeded** (as shown above)

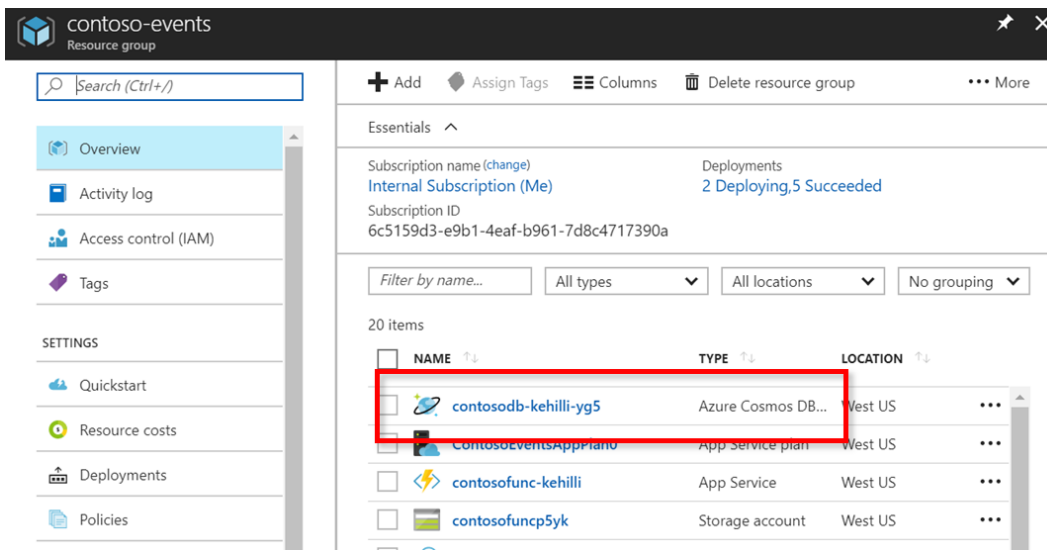
Task 3: Cosmos DB

In this section, you will setup the Cosmos DB database and collections that will be used to store events and collect ticket orders.

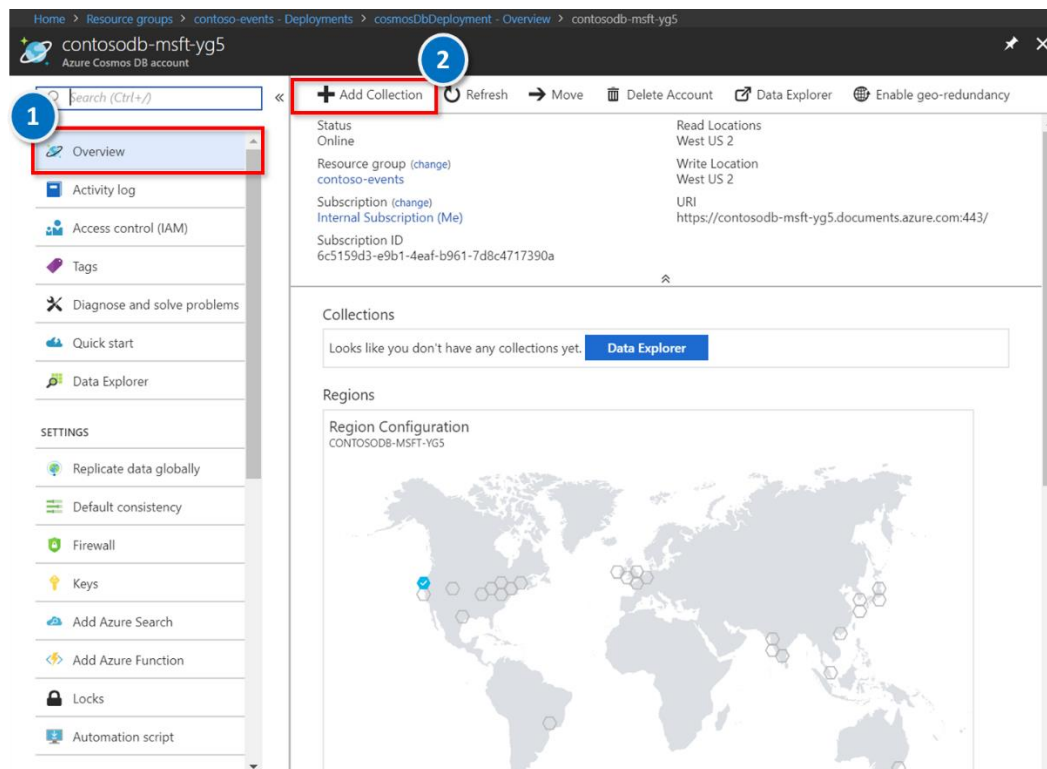
TIP: The deployment of the Cosmos DB account must be finished before completing this task. Please verify the account is available (see task 2).

Tasks to complete

1. Azure Portal, navigate to the **contoso-events** resource group and choose the *Cosmos DB* resource that was created.



2. Once in the account, choose **Overview** and **+Add Collection**



3. Enter **"TicketManager"** for the **Database Id**
4. Enter Orders for the **Collection Id**
5. Storage capacity set to Fixed (10GB)
6. Select **Fixed** for **Storage Capacity**, and leave the keys blank (don't add a unique key)

Add Collection
✕

* Database id ⓘ

Create new Use existing

TicketManager

* Collection Id ⓘ

Orders

* Storage capacity ⓘ

* Throughput (400 - 10,000 RU/s) ⓘ

5000

Estimated spend (USD): \$0.40 hourly / \$9.60 daily.
Choose unlimited storage capacity for more than 10,000 RU/s.

Unique keys ⓘ

+ Add unique key

7. Click **OK**
8. **Repeat steps 2-7.** This time, add a collection called **"Events"**, and choose **Use existing** and choose the **TicketManager** database

Exit criteria

- You will be able to see that the two collections exist in the new database.
- Confirm in the **Overview** area of the Cosmos DB account. It should look like below:

Collections

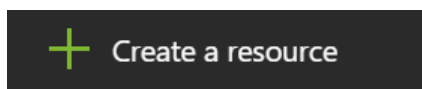
ID	DATABASE	THROUGHPUT (RU/S)
Orders	TicketManager	5000
Events	TicketManager	5000

Task 4: Azure Active Directory B2C

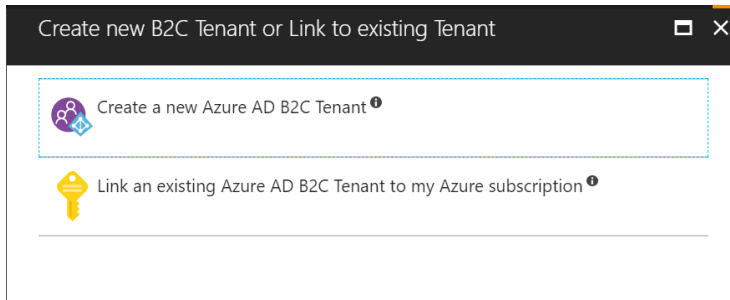
In this section you will provision an Azure Active Directory B2C tenant that demonstrates cloud-based identity to Contoso Events. Rather than using their existing on premises SQL Server solution directly to IaaS, Azure AD B2C will handle identity.

Tasks to complete

1. Click the + New in the Azure Portal.



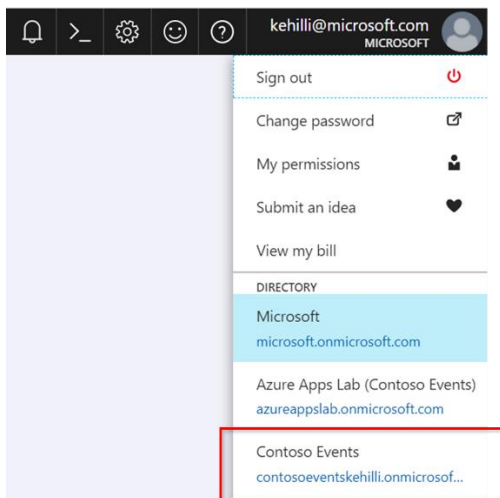
2. Search "Azure Active Directory B2C" in the search and select **Create a new Azure AD B2C Tenant**



3. In the next dialog, type "Contoso Events (Azure Apps)" for the Name
4. Next, enter "contosoevents" + [your_alias] for the initial domain name (**must be unique**)
5. Choose United States for the region if not already selected.
6. Click **Create**

Exit criteria

- Confirm the tenant is provisioned by refreshing the Portal and expanding the user profile menu at the top right. The tenant should be in the DIRECTORY list.



Exercise 2: Placing ticket orders

Duration: 30 minutes

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 an Order to the Orders collection of the Cosmos DB TicketManager database.

You will be setting the keys' values in configuration to light up this feature.

Task 1: Set up the Ticket Order Sync queue

The purpose of this task is to complete features of the Contoso Events application so that placing an order commits it to the backend data store. You will update the configuration settings to correctly reference the Azure resources you previously created.

Update Service Fabric settings

1. Open Visual Studio Code
2. Choose File → Open Folder...
3. Locate the folder you unzipped the application in Exercise 1, Task 1
4. Open **Cloud.xml** from the Service Fabric\ApplicationParameters folder
5. You will be updating the configuration parameters for the following items with your own:

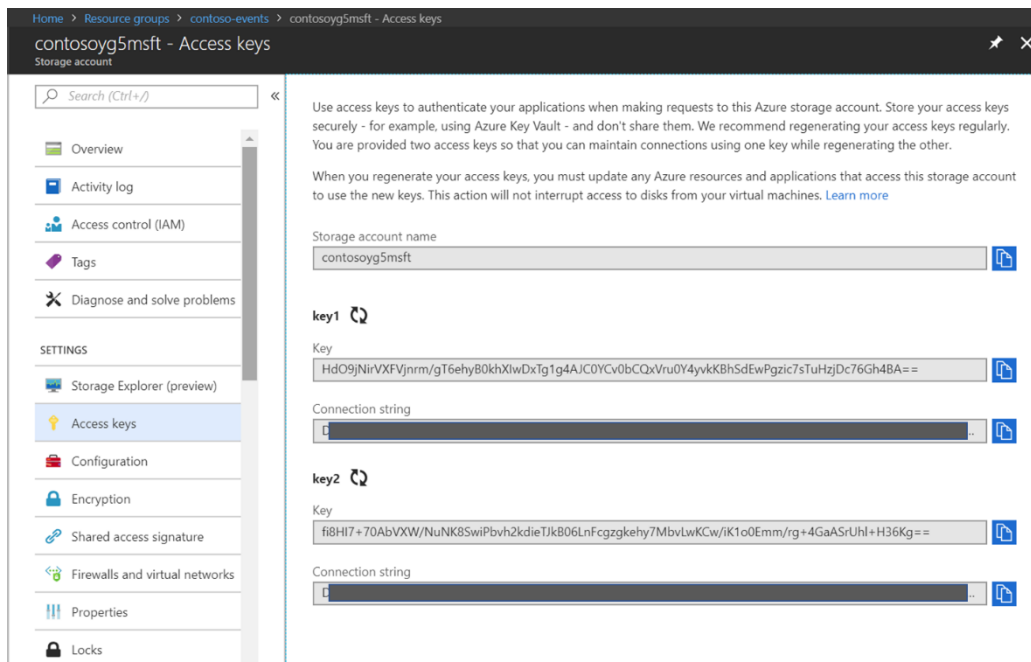
```
<Parameter Name="DataStorageEndpointUri" Value="" />
<Parameter Name="DataStoragePrimaryKey" Value="" />
<Parameter Name="DataStorageDatabaseName" Value="TicketManager" />
<Parameter Name="DataStorageEventsCollectionName" Value="Events" />
<Parameter Name="DataStorageOrdersCollectionName" Value="Orders"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

Cosmos DB settings

1. From the Azure Portal, browse to the Cosmos DB you created previously
2. Set **DataStorageEndpointUri** to the Cosmos DB endpoint **URI** (found in **Overview**).
3. Set **DataStoragePrimaryKey** to the Cosmos DB Primary Key. (The key can be retrieved by going to the account and selecting **Keys** in the menu)

Storage settings

1. From the Azure Portal, browse to the Storage account that was created. It should be in the format of "contoso" + [3 random characters] + [your alias].
2. Under **SETTINGS**, select **Keys**
3. Copy the Connection String value for **key1**



4. Set the value for the **StorageConnectionString** with the value you copied from the portal.

Task 2: Publish the Service Fabric application

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

Update publish profile settings

1. From with Visual Studio Code, open ServiceFabric\PublishProfiles\Cloud.xml
2. Update **ConnectionString** attribute value
3. Replace **[CLUSTER_NAME]** and **[LOCATION]** with the values from your Service Fabric cluster

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <PublishProfile xmlns="http://schemas.microsoft.com/2015/05/fabrictools">
3   <ClusterConnectionParameters ConnectionEndpoint="[CLUSTER_NAME].[LOCATION].cloudapp.azure.com:19000" />
4   <ApplicationParameterFile Path="..\ApplicationParameters\Cloud.xml" />
5 </PublishProfile>

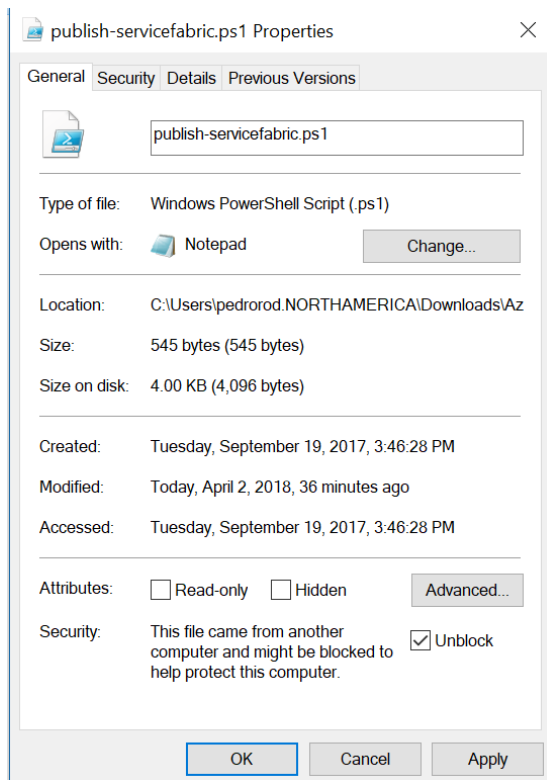
```

Tasks to complete

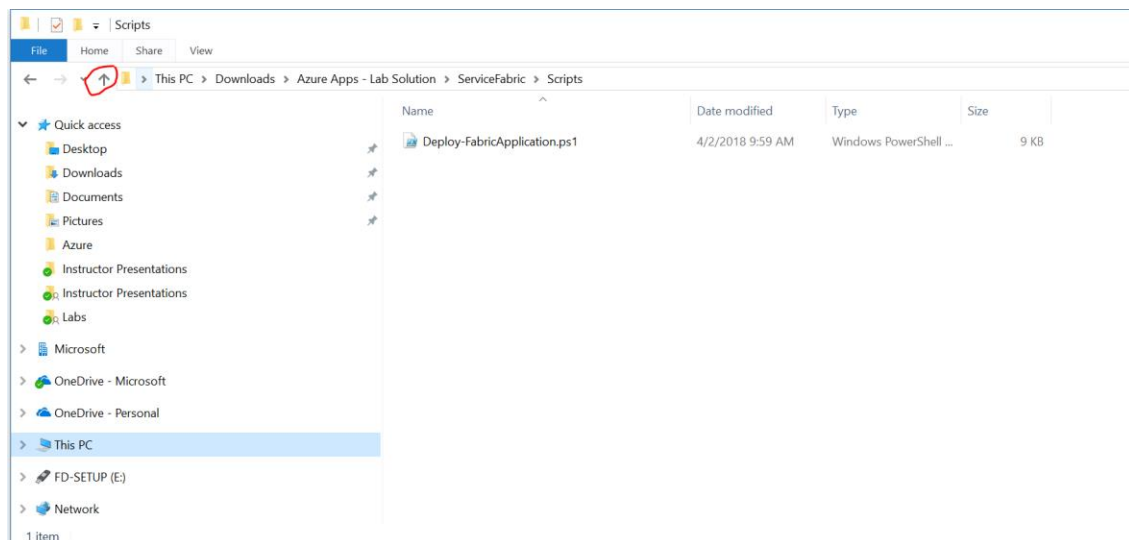
1. From Windows Explorer, open the folder you unzipped in Setup, Task 1.
2. Locate the publish-servicefabric.ps1 file under ServiceFabric\Scripts.

NOTE: Because this file came from the Internet, you need to unblock it.

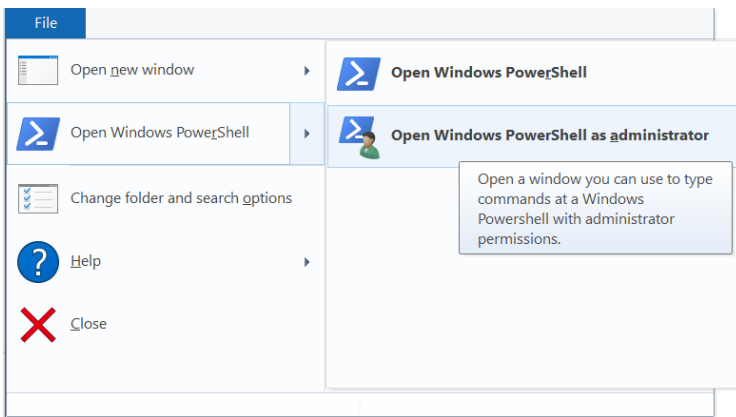
3. Right click the Deploy-FabricApplication.ps1 and select Properties. Then click the Unblock checkbox and click OK:



4. Go back to the top level directory for the lab (Azure Apps – Lab Solution). You can click the arrow in the Windows Explorer twice:



5. Launch PowerShell as Administrator from this location:



6. Type the following into the terminal:

```
. .\publish-servicefabric.ps1
```

NOTE: Be sure to include the extra "." Dot at the beginning, otherwise the script will fail!

7. Output should be shown similar to the below screenshot:

```
creating application...
ApplicationName       : fabric:/ContosoEventsApp
ApplicationTypeName   : ContosoEventsAppType
ApplicationTypeVersion : 1.0.0
ApplicationParameters : { "ActorBackupReminderPeriodicInMinutes" = "60";
                           "statefulServiceLoopPause" = "500";
                           "IsAzureTableStorageLogging" = "false";
                           "DataStorageLogMessagesCollectionName" = "LogMessages";
                           "IsTicketAvailabilityCheck" = "true";
                           "TicketOrderService_PartitionCount" = "5";
                           "HealthIssuesTimeToLive" = "20";
                           "TicketOrderActorService_PartitionCount" = "5";
                           "ActorBackupReminderDueInMinutes" = "10";
                           "TicketOrderService_MinReplicasetSize" = "3";
                           "BackupQueueName" = "";
                           "DataStoragePrimaryKey" = "";
                           "IsAzureFunctionLogging" = "false";
                           "SimulationQueueName" = "ticketorders-simulation-requests";
                           "EventActorService_PartitionCount" = "1";
                           "WebApi_InstanceCount" = "5";
                           "EmailServerPassword" = "";
                           "TicketOrderService_TargetReplicasetSize" = "3";
                           "LogQueueName" = "";
                           "EmailServerUserName" = "";
                           "ExternalizationQueueName" = "ticketorders-externalization";
                           "DataStorageEventsCollectionName" = "Events";
                           "DataStorageEndpointUri" = "";
                           "DataStorageDatabaseName" = "TicketManager";
                           "StorageConnectionString" = "";
                           "LogsStorageTableName" = "prodlogs";
                           "EmailServerUrl" = "smtp.gmail.com";
                           "EmailServerPort" = "587";
                           "IsEtwLogging" = "false";
                           "DataStorageOrdersCollectionName" = "orders" }
create application succeeded.
```

8. The process can take a few minutes. Once it's succeeded, proceed to the next task.

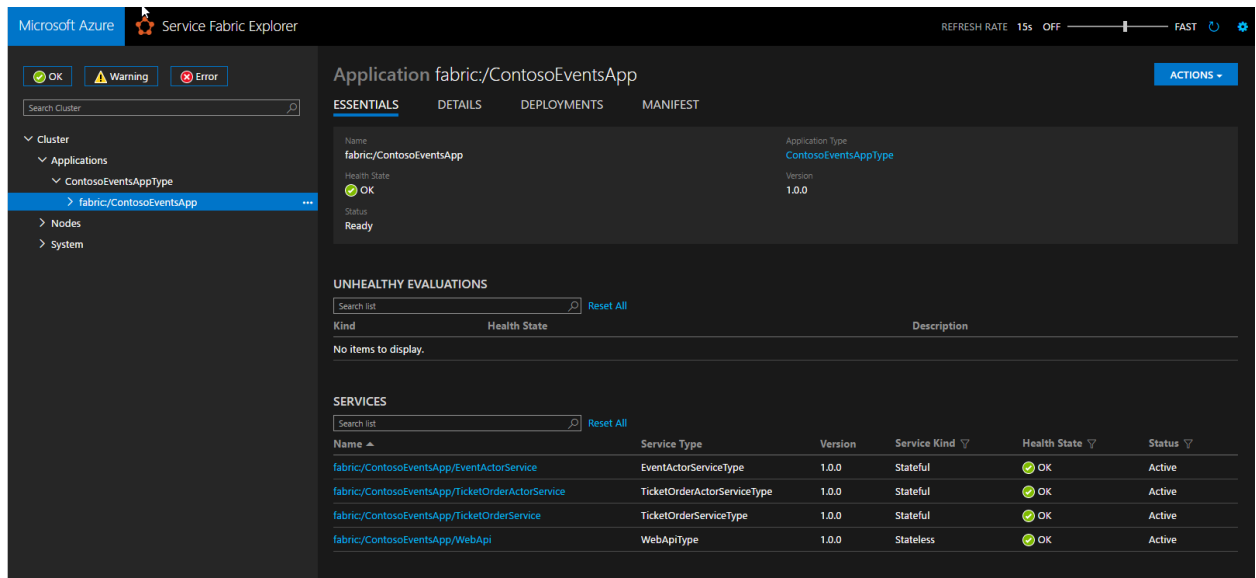
Task 3: Service Fabric Explorer

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

Tasks to complete

1. In a new browser tab, navigate to the Service Fabric Explorer for the cluster at:

<http://<clusterName>.<location>.cloudapp.azure.com:19080/Explorer/index.html>.



The screenshot shows the Service Fabric Explorer interface for the application fabric:/ContosoEventsApp. The interface is divided into several sections:

- Header:** Microsoft Azure Service Fabric Explorer, REFRESH RATE 15s OFF, FAST.
- Navigation:** Cluster, Applications, ContosoEventsAppType, fabric/ContosoEventsApp, Nodes, System.
- Application Overview:** Name: fabric/ContosoEventsApp, Application Type: ContosoEventsAppType, Health State: OK, Status: Ready.
- UNHEALTHY EVALUATIONS:** Search list, Reset All, No items to display.
- SERVICES:** Search list, Reset All, Table with columns: Name, Service Type, Version, Service Kind, Health State, Status.

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

7. 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

Exit criteria

- If you are able to access the Service Fabric Explorer, your environment is in a good state to continue.

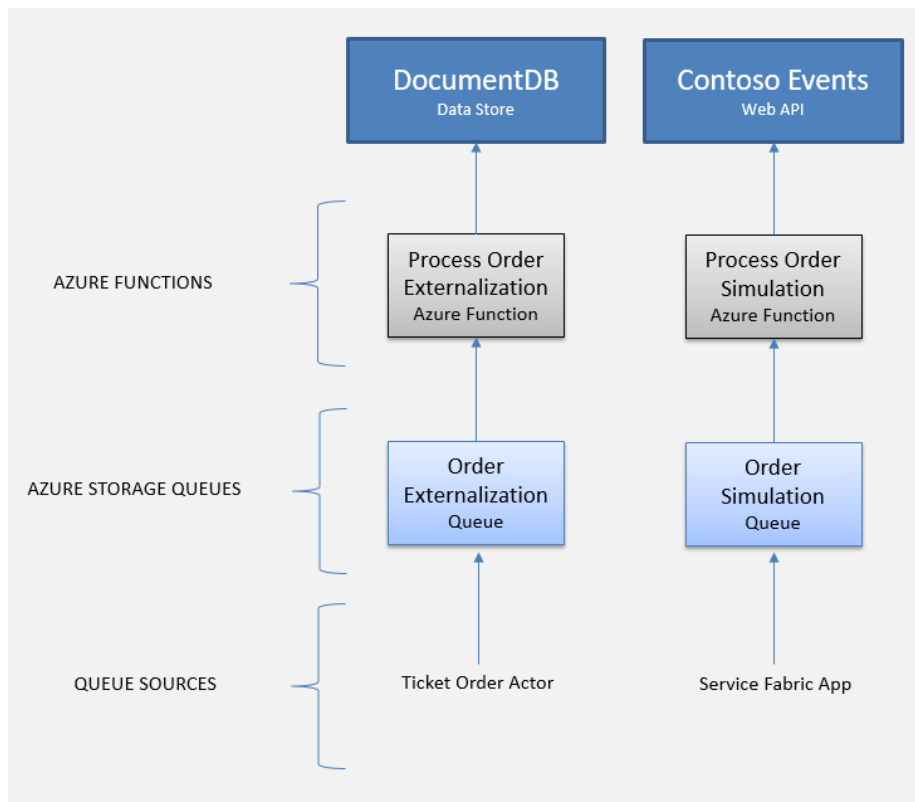
Task 4: Set up the Functions App

The purpose of this task is to 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.

In this task you will also create a second function that will be used to generate load against the system at runtime. This will be used later for Exercise 5 where the system will be load tested.

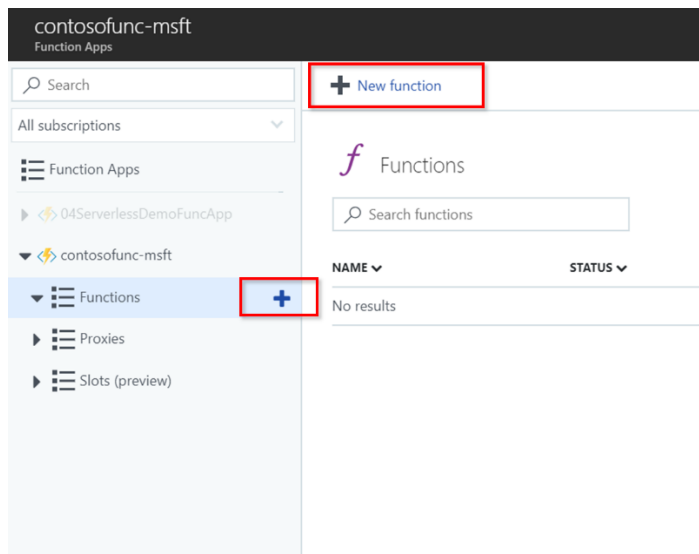
NOTE: Exercise 5 is a bonus exercise and is optional if you have time remaining.

Overall, the Contoso Events Ticketing subsystem has the following queue and function architecture and this should help visualize how the queues and functions are related:



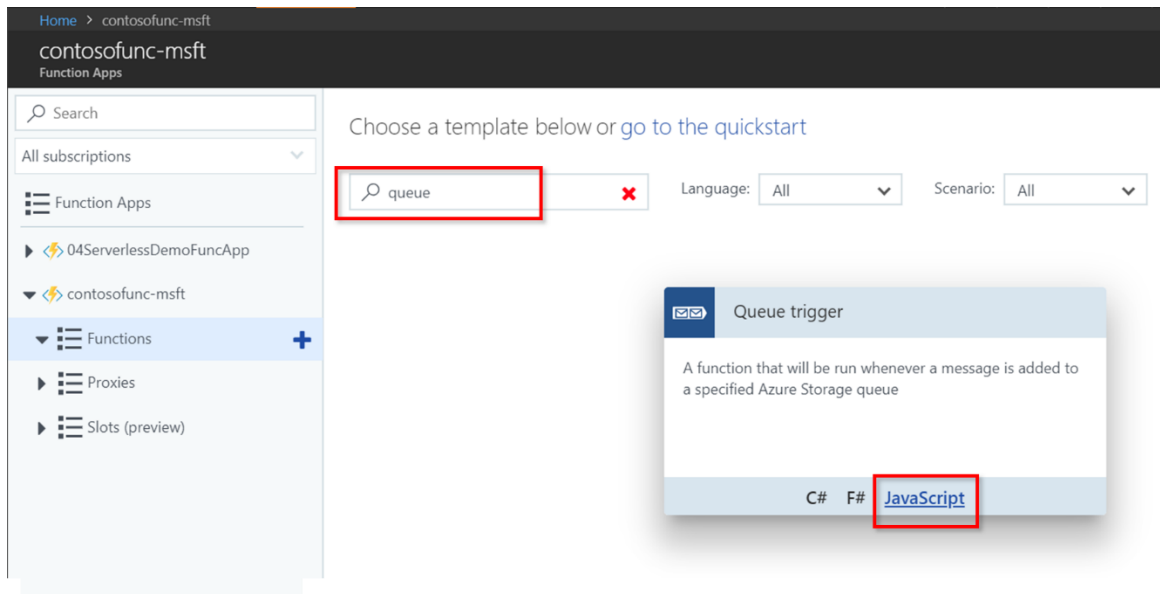
Tasks to complete

1. From the Azure Portal, access the Function App you created previously.
2. Expand the Function App, and click on the “+” icon to the right of **Functions** or click on **New Function**.
3. Under “**Choose a language**”, click on create your own custom function

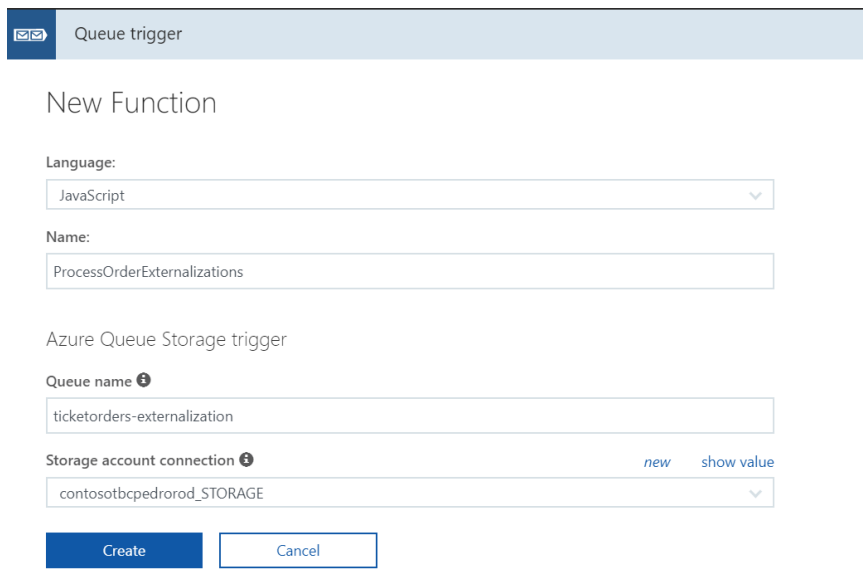


4. When you are prompted, enter “queue” in the search, and select the **Queue Trigger**

- click on **JavaScript**.



- Input "ProcessOrderExternalizations" for the **Name**
- Input the externalization queue name, "ticketorders-externalization" for the **Queue name**.



- Select the Storage account you created previously by clicking on **new**. This is where the queue will be located.
- Click **Create**

Name your function

ProcessOrderExternalizations

Configure

Azure Storage Queue trigger (myQueueItem)

Queue name

ticketorders-externalization

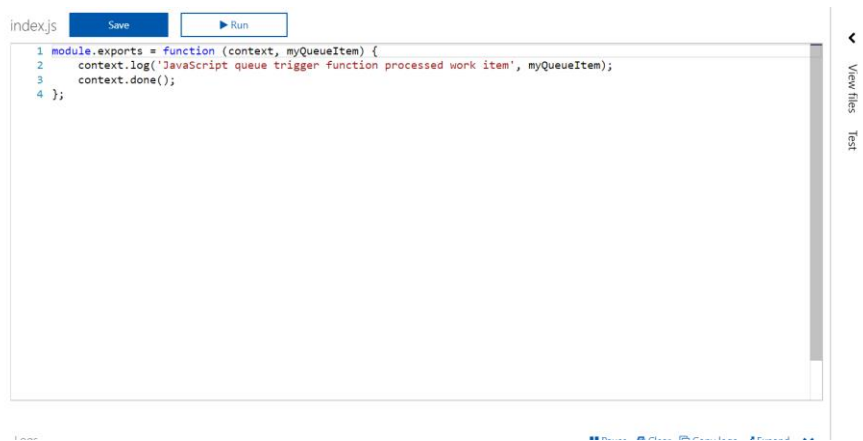
Storage account connection

contosoeventssoll_STORAGE

[select](#)

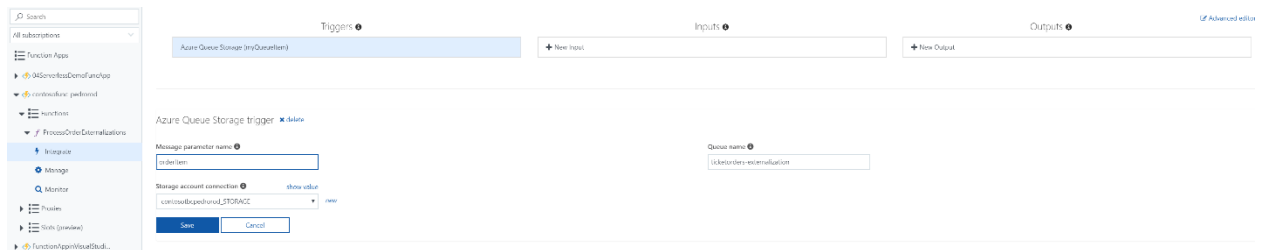
Create

10. The screen should look similar to the below screenshot:



```
index.js Save Run
1 module.exports = function (context, myQueueItem) {
2   context.log('JavaScript queue trigger function processed work item', myQueueItem);
3   context.done();
4 };
```

11. Click the **Integrate** link on the left.



The screenshot shows the Azure portal interface for configuring a function trigger. On the left, a navigation pane includes a search bar and a list of options: Function Apps, All subscriptions, 0ASServiceBusDemoFunction, contosoeventssoll-paefnord, Function, ProcessOrderExternalizations, Integrate (highlighted), Monitor, Pools, and FunctionApp/FossilFuel... The main area displays the configuration for an 'Azure Queue Storage trigger'. It includes sections for Triggers (with 'Azure Queue Storage (myQueueItem)' selected), Inputs (with '+ New Input' button), and Outputs (with '+ New Output' button). The configuration details for the trigger are shown below, including a 'Message parameter name' field set to 'orderItem', a 'Storage account connection' dropdown set to 'contosoeventssoll_STORAGE', and a 'Queue name' field set to 'ticketorders-externalization'. 'Save' and 'Cancel' buttons are at the bottom.

12. For the Trigger set the **Message parameter name** to "orderItem".

13. Click **Save** (IMPORTANT or the message parameter name will not be saved!).

14. Click + New Output

+ New Output

15. Click Azure Cosmos DB and click **Select**.

Now you will supply parameters for the Cosmos DB output.

16. Enter the **Document parameter name** "orderDocument".

17. For Collection Name enter "Orders".

18. For the database enter "TicketManager".

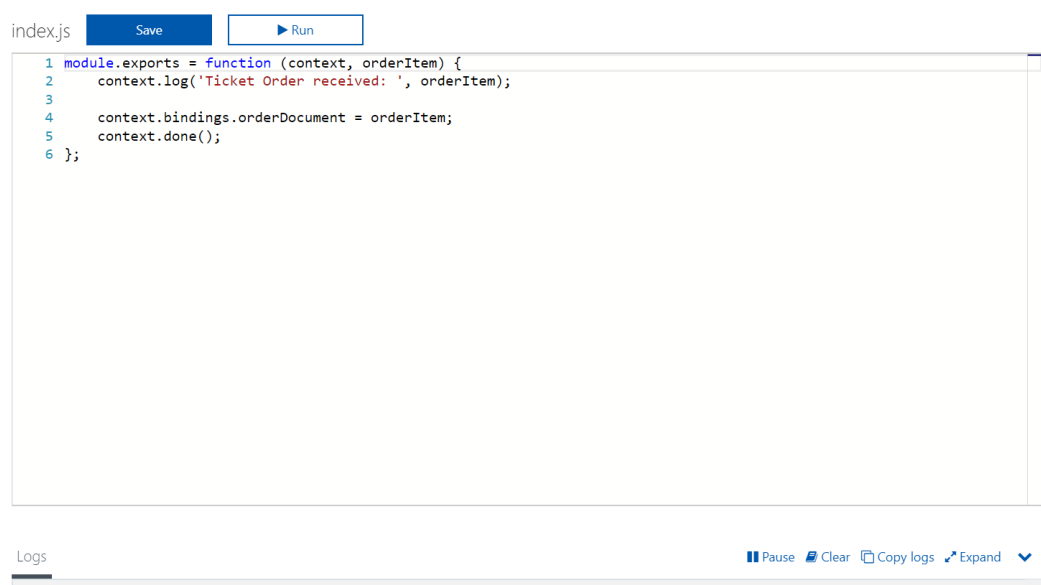
19. For the **Cosmos DB account connection**, click on *new* and choose the account created earlier to establish a connection to the account

Azure Cosmos DB output [✕ delete](#)

<p>Document parameter name ⓘ</p> <input type="text" value="orderDocument"/> <p><input type="checkbox"/> Use function return value</p> <p>Collection Name ⓘ</p> <input type="text" value="Orders"/> <p>Partition key (optional) ⓘ</p> <input type="text" value="Partition key (optional)"/>	<p>Database name ⓘ</p> <input type="text" value="TicketManager"/> <p>If true, creates the Azure Cosmos DB database and collection ⓘ</p> <input type="checkbox"/> <p>Azure Cosmos DB account connection ⓘ show value</p> <input style="border-bottom: 1px solid #ccc;" type="text" value="contosodb-msft-yg5_DOCUMENTDB"/> new
--	---

20. Click **Save**.
21. Select the function in the menu (the name of the function with an f icon).
22. Using Visual Studio Code, locate ProcessTicketOrderExternalizationEvent.js in the **\FunctionApp** folder. Copy and paste this code into the Code window.

```
module.exports = function (context, orderItem) {  
    context.log('Ticket Order received: ', orderItem);  
  
    context.bindings.orderDocument = orderItem;  
    context.done();  
};
```



```
index.js Save Run  
1 module.exports = function (context, orderItem) {  
2   context.log('Ticket Order received: ', orderItem);  
3  
4   context.bindings.orderDocument = orderItem;  
5   context.done();  
6 };
```

Logs Pause Clear Copy logs Expand

23. Click **Save**.

24. From the Logs section note that the function compiled successfully.

```
Logs || Pause  Clear  Copy logs  Expand  v  
2017-09-17T18:56:29 Welcome, you are now connected to log-streaming service.  
2017-09-17T18:57:29 No new trace in the past 1 min(s).  
2017-09-17T18:58:21.351 Script for function 'ProcessOrderExternalizations' changed. Reloading.  
2017-09-17T18:58:21.468 Compilation succeeded.
```

Note: Because the *ProcessOrdersExternalization* function is set up, you will be able to process an order and see that it is saved to the Orders collection of Cosmos DB.

25. Click the **+** on the **Functions**.

26. Now you will create another function for the load simulation you will use later. You will follow the same steps in this section with the following differences:

a. From the templates scenario enable **Experimental Language Support**:

Choose a template below or [go to the quickstart](#)

Language: Scenario: Experimental Language Support: Enabled

b. Select the template QueueTrigger – PowerShell (instead of JavaScript)

Queue trigger

New Function

Language:

Name:

Azure Queue Storage trigger

Queue name [?]

Storage account connection [?] new show value

This language is experimental and does not yet have full support. If you run into issues, please file a bug on our [GitHub repository](#).

- c. Set the function name to "ProcessSimulationRequests".
- d. Set the queue name to "ticketorders-simulation-requests".
- e. Set the Storage account connection to the same storage account as before
- f. Click **Create**

- g. Click on the **Integrate** link for the function.
- h. Choose the message parameter name "simulationRequest".
- i. In this case there is no need to set up an output so you can skip this step.
- j. Click **Save**.
- k. Click on the Function link on the left to return to the code screen.
- l. From Visual Studio Code, find the file *ProcessOrderTicketSimulationRequest.ps1* in the **FunctionApp** folder.
- m. Copy the contents of this file into the code text area and click **Save**.
- n. The final setting to update is the API Management key. You will return to this when you set up the API Management service.

Exit criteria

- If each function successfully compiled as you went through these steps, you are ready to proceed to the next exercise. Review and confirm you matched the case on function and parameter names above (everything is case-sensitive).

Task 5: Test an order from the cluster

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

Tasks to complete

1. From the browser, navigate to the Swagger endpoint for the Web API exposed by the hosted Service Fabric cluster using port 8082. The URL is made of:

<http://<cluster-name>.<location>.cloudapp.azure.com:8082/swagger/ui/index>

For example:

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

- Expand the Orders API and expand the POST /api/orders API operation as shown in the following screen shot.

The screenshot shows the Swagger UI interface for ContosoEvents.WebApi. The 'Orders' section is expanded, and the 'POST /api/orders' endpoint is selected. The 'Parameters' table shows a required 'order' parameter of type 'body'. A red callout bubble with the text 'paste order request' points to the 'order' parameter value field. To the right, the 'Model Schema' is displayed as a JSON object with various fields like 'id', 'orderDate', 'fulfillDate', etc.

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

```
{
  "id": "string",
  "orderDate": "2016-07-19T13:29:21.140Z",
  "fulfillDate": "2016-07-19T13:29:21.140Z",
  "cancellationDate": "2016-07-19T13:29:21.140Z",
  "userName": "string",
  "email": "string",
  "tag": "string",
  "eventId": "string",
  "paymentProcessorTokenId": "string",
}
```

- Copy and paste the following JSON to in the parameter text area, then click **Try it out**.

```
{
  "UserName": "johnsmith",
  "Email": "john.smith@gmail.com",
  "Tag": "Manual",
  "EventId": "EVENT1-ID-00001",
  "PaymentProcessorTokenId": "YYTT6565661652612516125",
  "Tickets": 3
}
```

The screenshot shows an API testing tool interface for a POST request to `/api/orders`. The response class is `Status 200` and the response content type is `application/json`. The parameters section shows a table with columns for Parameter, Value, Description, Parameter Type, and Data Type. The parameter `order` has a value of a JSON object and a parameter type of `body`. The value is displayed in a text area with a scroll bar. Below the text area is a dropdown menu for the parameter content type, set to `application/json`. To the right of the parameter value is a `Model` section with a `Model Schema` tab. The schema is a JSON object with the following properties: `orderDate`, `fulfillDate`, `cancellationDate`, `userName`, `email`, `tag`, `eventId`, `paymentProcessorTokenId`, `paymentProcessorConfirmation`, `tickets`, and `isFulfilled`. Below the parameters section is a `Response Messages` section with a table of HTTP status codes, reasons, and response models. The table has columns for HTTP Status Code, Reason, Response Model, and Headers. The first row is `400` with reason `An exception occurred`. The second row is `404` with reason `NotFound`. A `Try it out` button is located below the table and is highlighted with a red box.

Parameter	Value	Description	Parameter Type	Data Type
order	<pre>{ "OrderDate": "2016-07-18T17:32:41+00:00", "UserName": "johnsmith", "FirstName": "John", "LastName": "Smith", "Email": "john.smith@gmail.com" }</pre>		body	Model Model Schema

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

Try it out

- This should return successfully with HTTP 200. The response includes a unique order id that clients could use to track the order.

Response Messages

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

Try it out! [Hide Response](#)

Curl

```
curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/json' -d '{
  "OrderDate": "2016-07-18T17:32:41+00:00",
  "UserName": "johnsmith",
  "FirstName": "John",
  "LastName": "Smith",
  "Email": "john.smith@gmail.com",
  "Tag": "Manual",
  "EventId": "EVENT1-ID-00001",
  "PaymentProcessorTokenId": "YYT6565661652612516125",
  "PaymentProcessorConfirmation": "162561216516251244542354235",
  "Tickets": 3
}' 'http://localhost:8082/api/orders'
```

Request URL

http://localhost:8082/api/orders

Response Body

```
"b805565f-ade0-47bb-85c0-79798d4cd009"
```

Response Code

200

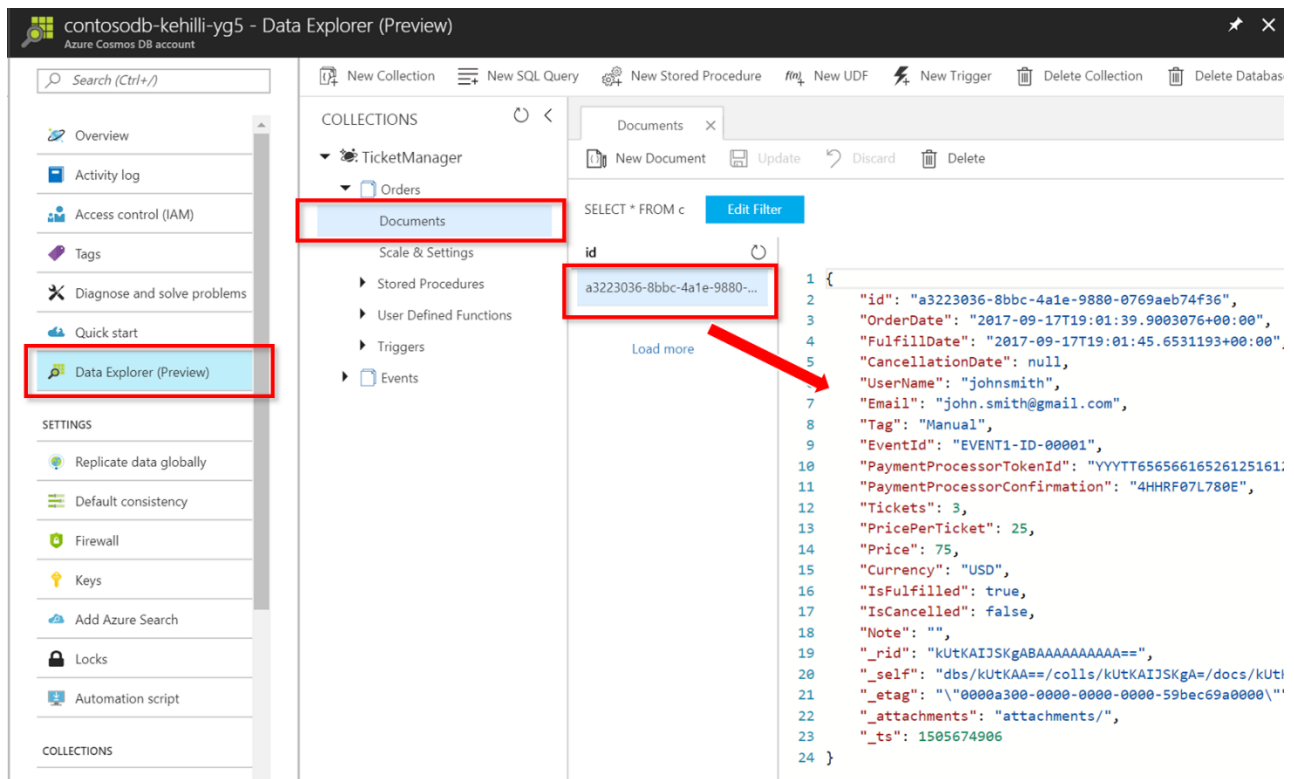
Response Headers

```
{
  "content-length": "38",
  "content-type": "application/json; charset=utf-8",
  "server": "Microsoft-HTTPAPI/2.0",
  "date": "Tue, 19 Jul 2016 16:50:30 GMT"
}
```

Note: A red box highlights the response body value, and a callout bubble points to it with the text "This is the ORDER ID".

Exit criteria

- Verify that the order has persisted to the Orders collection. From the Azure Portal, find your Cosmos DB account previously created.
- Click **Data Explorer** in the menu, then **Documents**
- Under **id**, click on an order document to view the contents to the right.

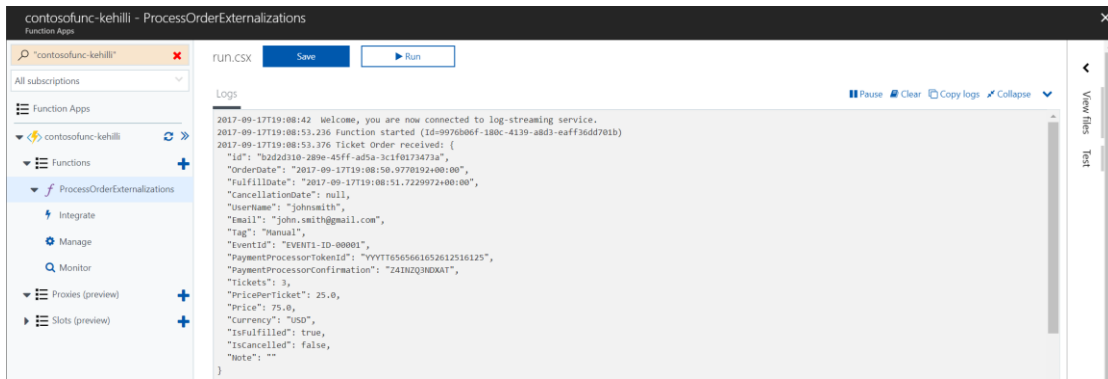


Task 6: Test order data sync

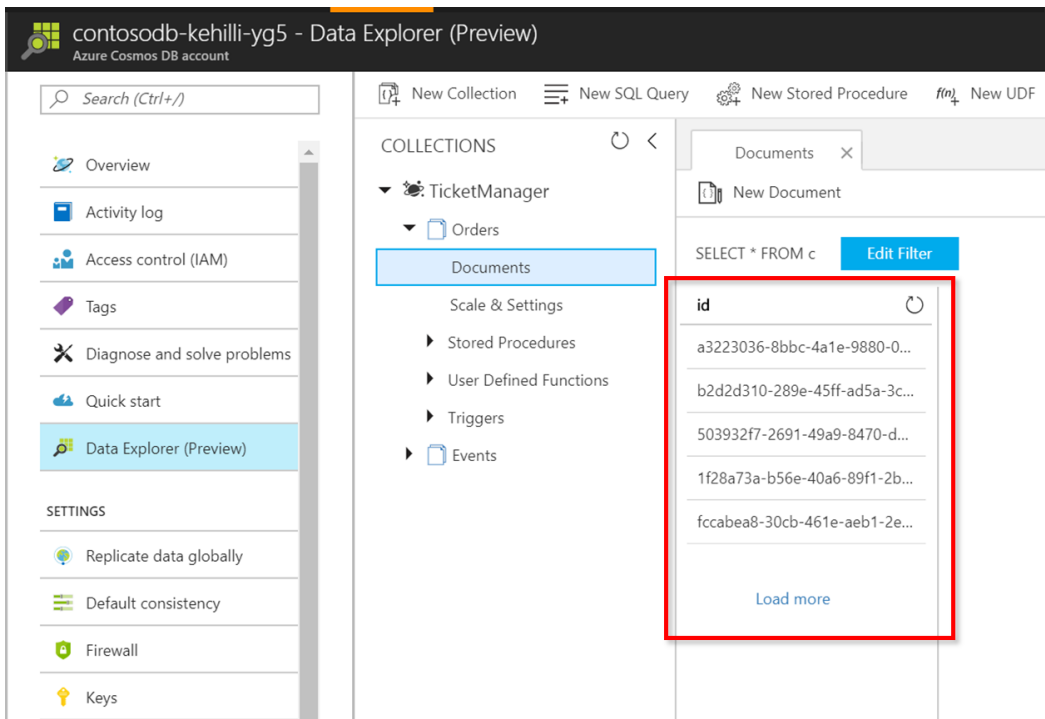
In this task you will test the ticket order processing backend to validate that orders are queued and processed by the TicketOrderProcessing function – ultimately saving the order to the Orders collection of the Cosmos DB instance.

Tasks to complete

1. From the Azure Portal, navigate to the **Function App**.
2. Select the **ProcessOrderExternalizations** function and expand Logs section below the code textarea.
3. Repeat the necessary steps to **“Test an order from the cluster”** to submit an order.
4. As orders are processed you will see activity in the function logs.



5. Note the order id of the function just processed. Use this information to verify the order is persisted to the Orders collection.
6. From the Azure Portal you can now confirm in the CosmosDB account's Data Explorer that the orders are indeed being stored



Exit criteria

- If the Cosmos DB query returns the order id specified, the order has been fully processed and persisted

Exercise 3: API Management

Duration: 15 minutes

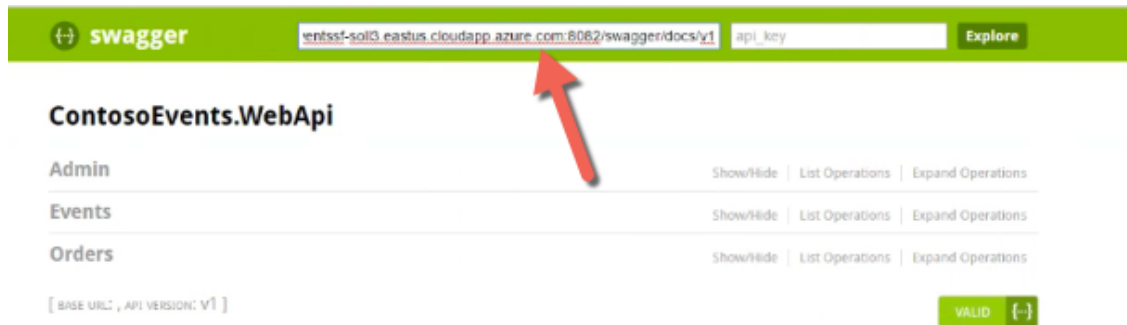
In this exercise you will configure the API Management service.

Task 1: Import API

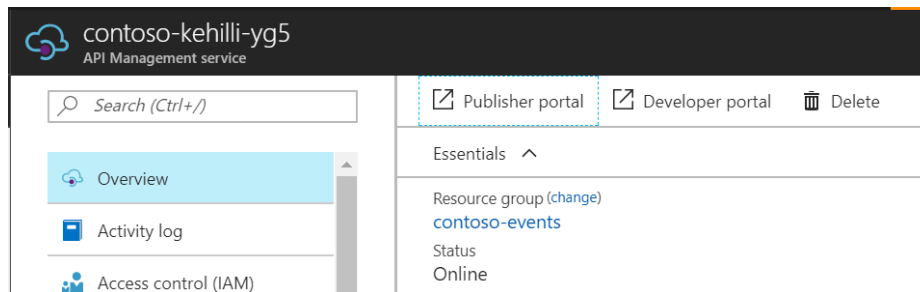
In this task you will import the Web API description to your API Management service to create an endpoint.

Tasks to complete

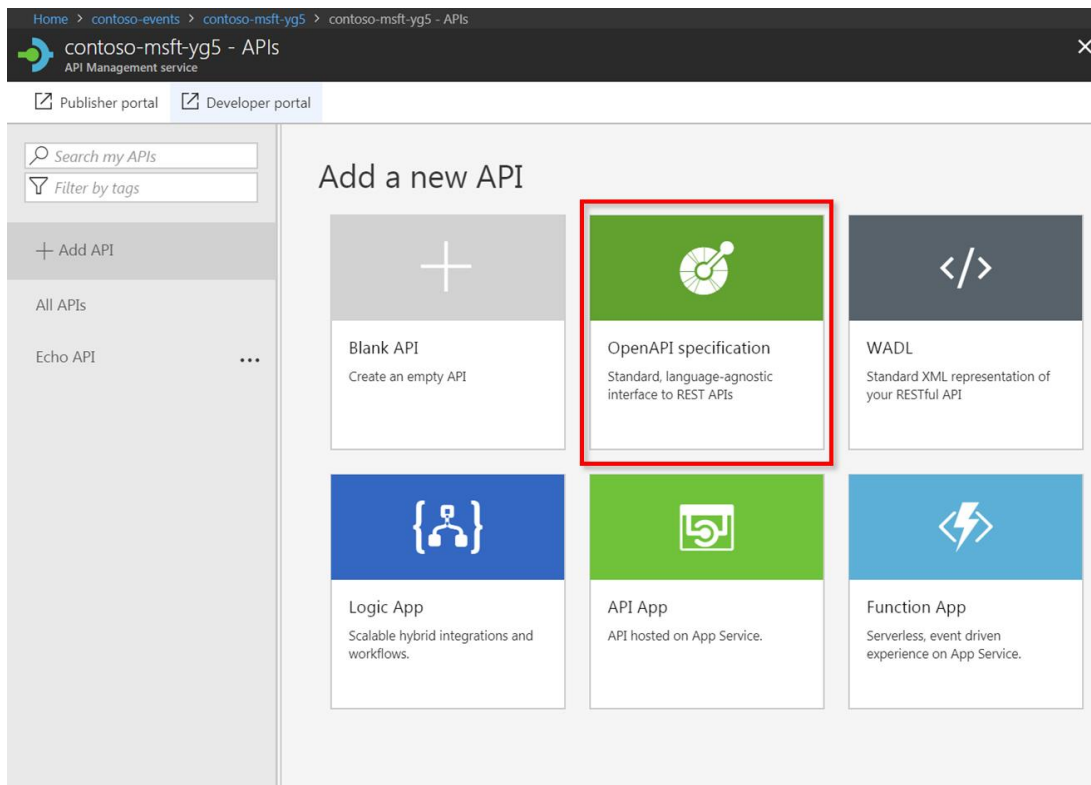
1. First, copy URL shown in the Swagger endpoint of the Service Fabric deployed Web API.



2. Next, from the Azure Portal, select the **API Management** service area
3. Click Publisher portal



4. Select **APIs**
5. Select **OpenAPI specification**



6. Paste the Swagger URL into the **OpenAPI specification field**
7. **Display Name** is "Events API"
8. Set the **Web API URL** suffix to "events" as shown in the following screen shot. Take note of what the URL will be as shown in the screenshot, such as <https://contosoeventsSUFFIX.azure-api.net/events/>. This is the URL you will use in your web site configuration in the next exercise.
9. Include product **Unlimited**.

Create from OpenAPI specification

* OpenAPI specification or

* Display name

* Name

Description

* URL scheme HTTP HTTPs Both

API URL suffix

Base URL

Tags PREVIEW

Products

Version this API?

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.

10. Click **Create**

Exit criteria

- You will see your API listed under APIs.

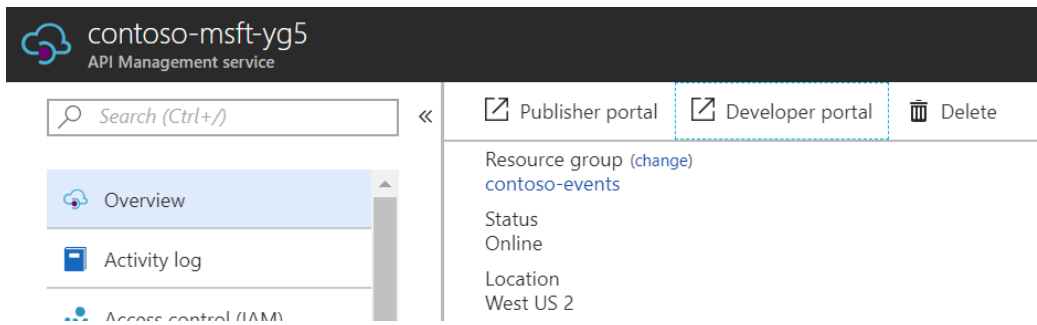
The screenshot shows the Azure API Management console for the service 'contoso-msft-yg5 - APIs'. The left sidebar contains a search bar for APIs, a filter by tags, and a list of APIs including 'Events API'. The main area shows the 'Events API' details, including a search bar for operations, a filter by tags, and a list of operations: 'CancelOrder', 'DeleteAllEvents', and 'DeleteAllLogM...'. The 'Frontend' tab is selected, showing a blank page with an edit icon.

Task 2: Retrieve the user subscription key

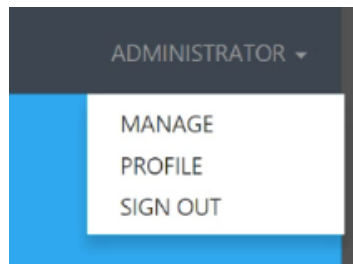
In this task you will retrieve the subscription key for the client applications to call the new API Management endpoint.

Tasks to complete

1. From the API Management dashboard in the portal, click the **Developer portal** menu item to navigate to the Developer portal as an Administrator with rights to complete the following steps.



2. Click the Administrator menu and then click Profile.



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

Your subscriptions

Subscription details			Product	State
Subscription name	Starter (default)	Rename	Starter	Active
Primary key	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	Show Regenerate		
Secondary key	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	Show Regenerate		
Subscription name	Unlimited (default)	Rename	Unlimited	Active
Primary key	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	Show Regenerate		
Secondary key	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	Show Regenerate		

Note: You would typically create a new product subscription 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.

4. Save this key to a text file or whatever is easiest for you. You will be using it in the next steps.

Unlimited (default)
7f0475b8897c43b6a9f25ebb7b3f0b53
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

Exit criteria

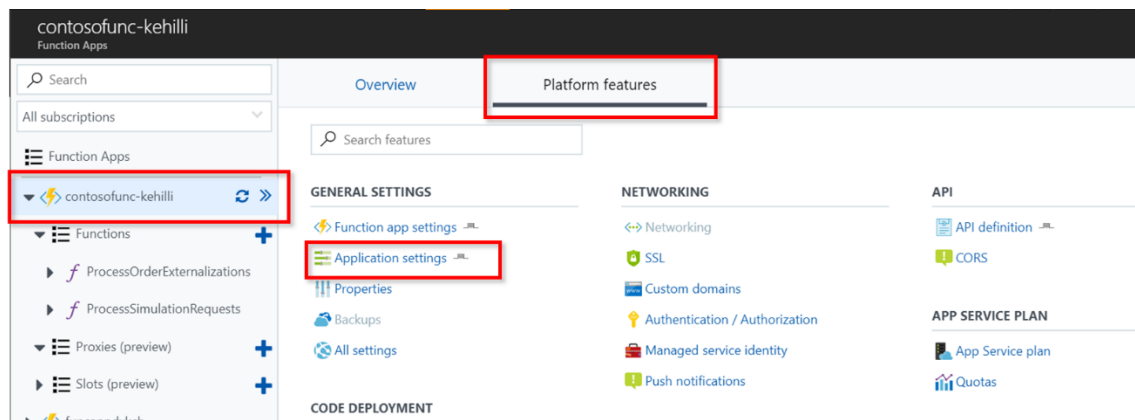
- You now have API Management application key you will need to configure the Function App settings for order load test simulation.

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

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

Tasks to complete

1. From the Azure Portal, browse to the Function App you created
2. You will create an application setting for the function to use the API Management subscription key.
3. Select Platform features
4. Select Application settings.



5. Scroll down and click on **+ Add new setting**
6. Enter "contosoeventsapimgrkey" for the name
7. For the value, place the key you saved for API Management.
8. Click **Save** (you may need to scroll back up to the top)

Application settings

AzureWebJobsDashboard	DefaultEndpointsProtocol=https;AccountName=contosofuncp5yk;AccountKey=M5pokVFWDu5NcYDjFVBKRWzi22w4rSF8...	✕
AzureWebJobsStorage	DefaultEndpointsProtocol=https;AccountName=contosofuncp5yk;AccountKey=M5pokVFWDu5NcYDjFVBKRWzi22w4rSF8...	✕
FUNCTIONS_EXTENSION_VERSION	~1	✕
WEBSITE_CONTENTAZUREFILECONNECTIONSTRING	DefaultEndpointsProtocol=https;AccountName=contosofuncp5yk;AccountKey=M5pokVFWDu5NcYDjFVBKRWzi22w4rSF8...	✕
WEBSITE_CONTENTSHARE	contosofuncp5yk	✕
WEBSITE_NODE_DEFAULT_VERSION	6.5.0	✕
contosoyg5kehilli_STORAGE	DefaultEndpointsProtocol=https;AccountName=contosoyg5kehilli;AccountKey=zRlle8DYOMzmk9lpvT9a7bwVeyilUM1Ru...	✕
contosodb-kehilli-yg5_DOCUMENTDB	AccountEndpoint=https://contosodb-kehilli-yg5.documents.azure.com:443;/AccountKey=Y5D9KPQCnTKy7mISAfZzGgtK...	✕
contosoeventsapimgrkey	009555db33754102b125662e54ddc498	✕

[+ Add new setting](#)

Exit criteria

- You will be able to issue a load test from the website in Exercise 5 and see that orders have been processed through the function – because it will have successfully called the API and you will see results in the load test status page.

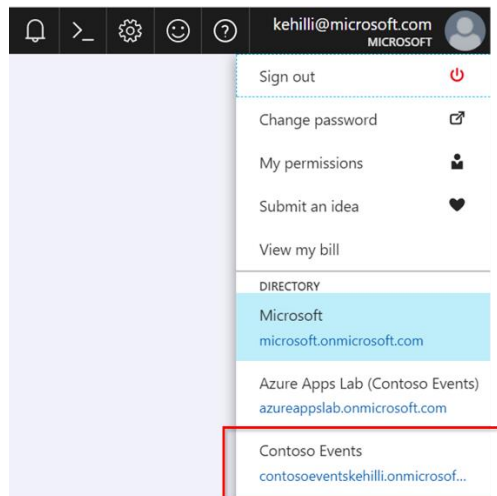
Exercise 4: Configure and publish the web application

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 Azure Active Directory B2C

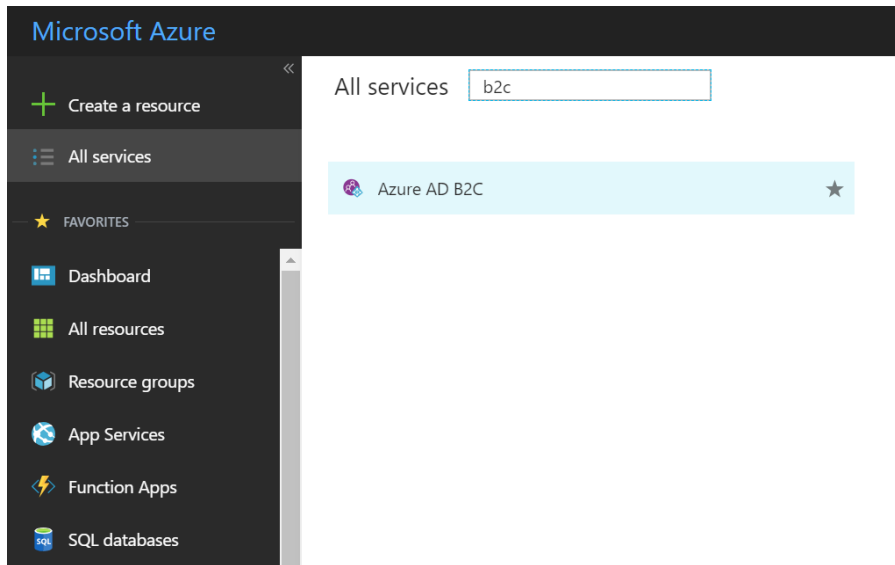
In this task you will set up the Azure Active Directory B2C directory for your application to integrate with it. Make sure you select the tenant under the DIRECTORY list in your profile's drop down menu (top right of the portal) before proceeding.



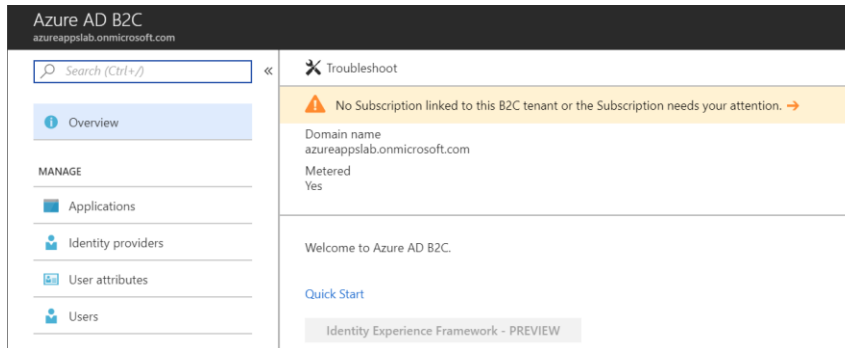
Tasks to complete

- From the Azure Portal browse to Azure B2C.

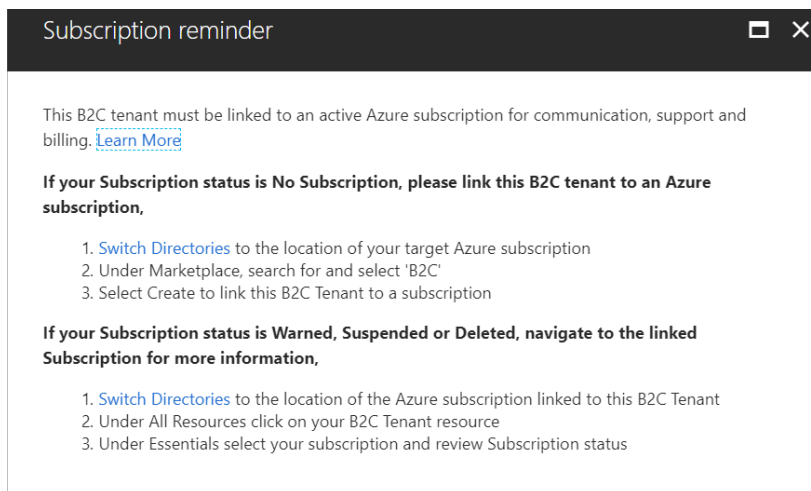
2. Select it to navigate to the Azure AD B2C Settings blade.



3. You should see the domain name you created earlier for your B2C directory.
4. If your B2C directory is not linked to a subscription, the below notice will appear:



5. Click on the yellow notice, and follow the instructions to link to your subscription



The linking screen will look similar to the below screen:

Create new B2C Tenant or Link to existing Tenant

Create a new Azure AD B2C Tenant ⓘ

Link an existing Azure AD B2C Tenant to my Azure subscription ⓘ

Azure AD B2C Tenant ⓘ
azureappslab.onmicrosoft.com

Azure AD B2C Resource name
azureappslab.onmicrosoft.com

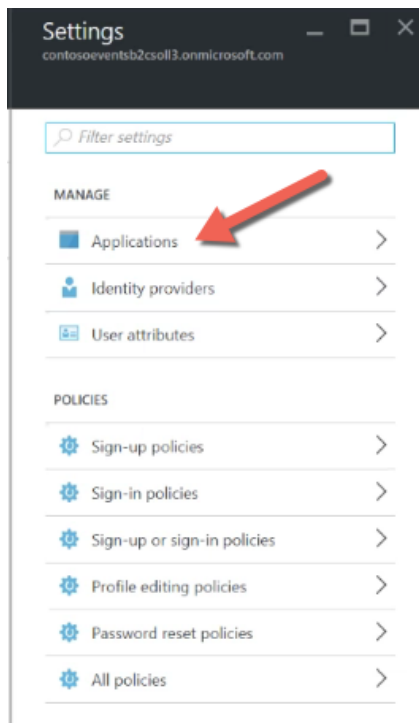
* Subscription
Internal Subscription (Me)

* Resource group
 Create new Use existing
contoso-events

Pin to dashboard

Create

6. (Once completed with the subscription linking if required, switch the directory back to the B2C directory in the portal)
7. From the Settings blade, select **Applications**.



8. Click + **Add**.
9. Enter the application name to "**Contoso Events - Ticketing**".
10. Select **Yes** for **include Web App / Web API**.
11. Select **Yes** for **Allow implicit flow**.
12. Add a reply URL for the hosted Web App as you named it. For example:

<https://contosoeventsweb-ALIAS.azurewebsites.net/>

Note: Make sure to include the closing "/" (slash) or the configuration will not work, and you MUST use HTTPS, not HTTP in the URL.

13. Click **Create**

New application

* Name ⓘ
Contoso Events - Ticketing ✓

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

Allow implicit flow ⓘ
 Yes No

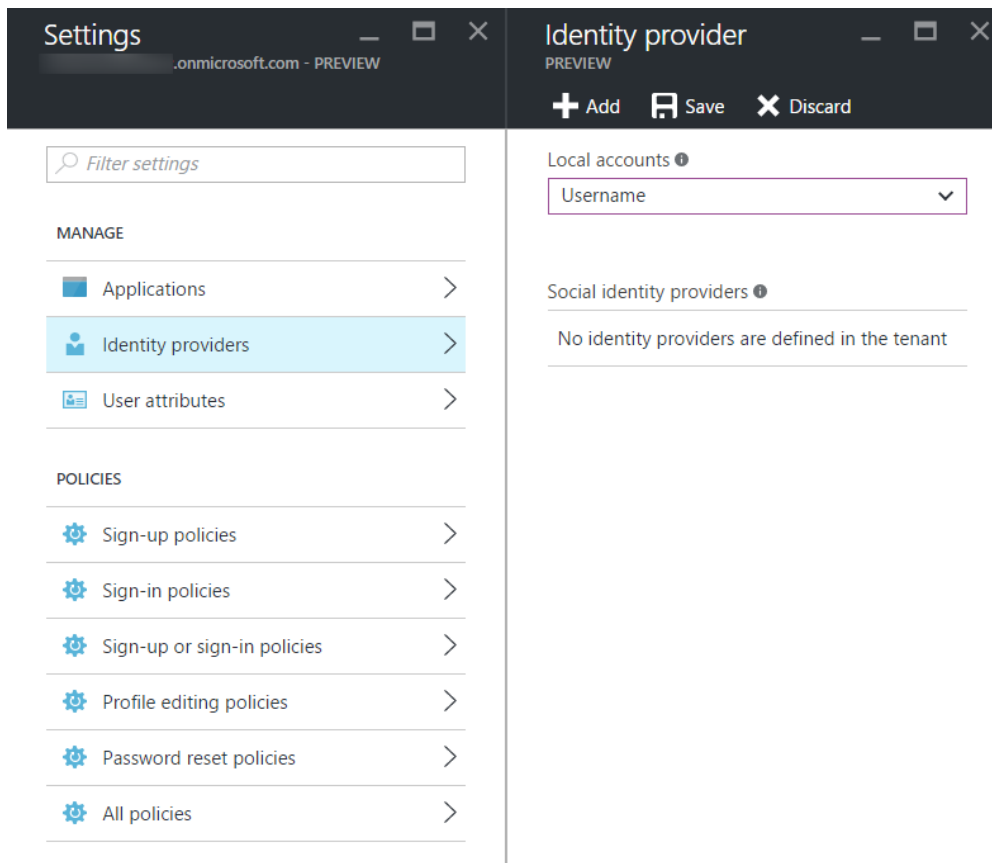
Redirect URIs must all belong to the same domain ⓘ

Reply URL ⓘ
https://contosoweb-msft.azurewebsites.net/ ...
 ...

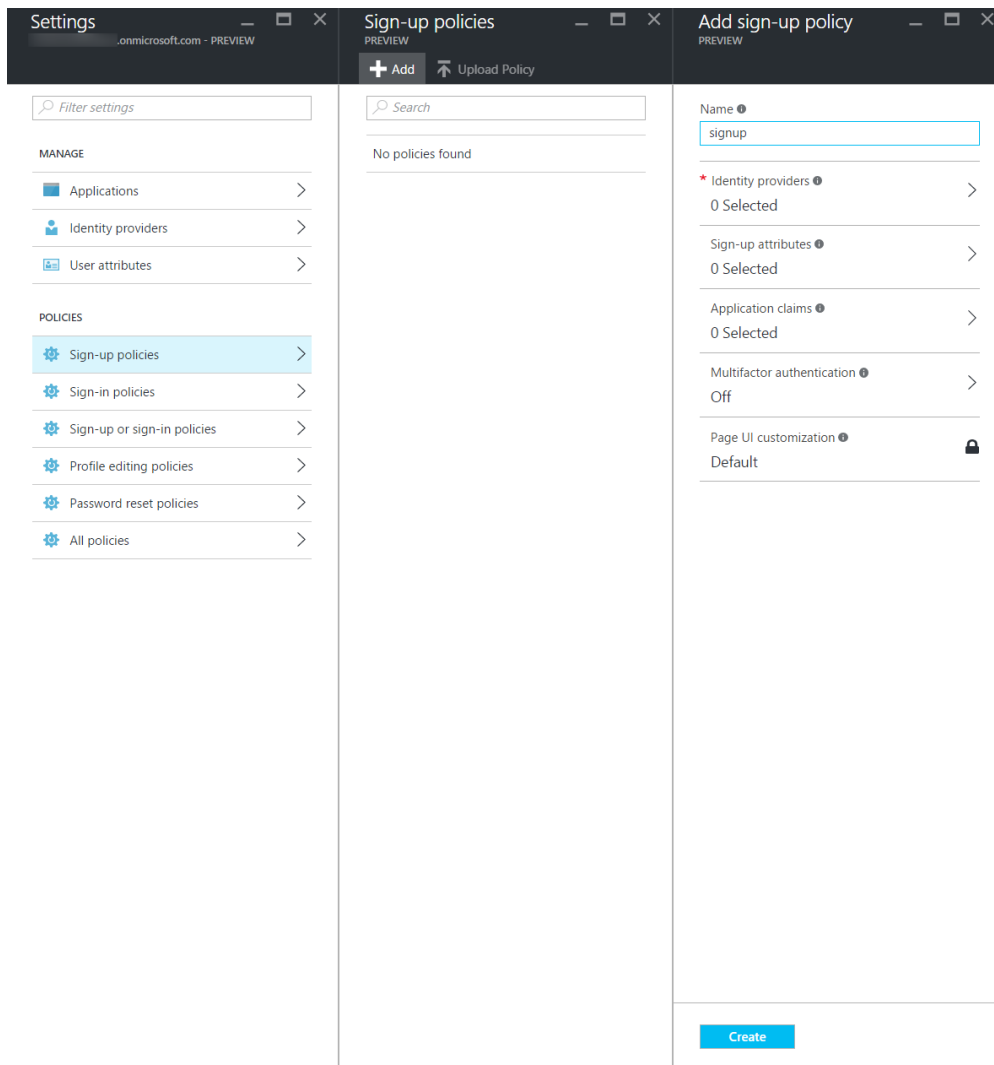
App ID URI (optional) ⓘ
https://azureappslab.onmicrosoft.com/

Native client
Include native client ⓘ
 Yes No

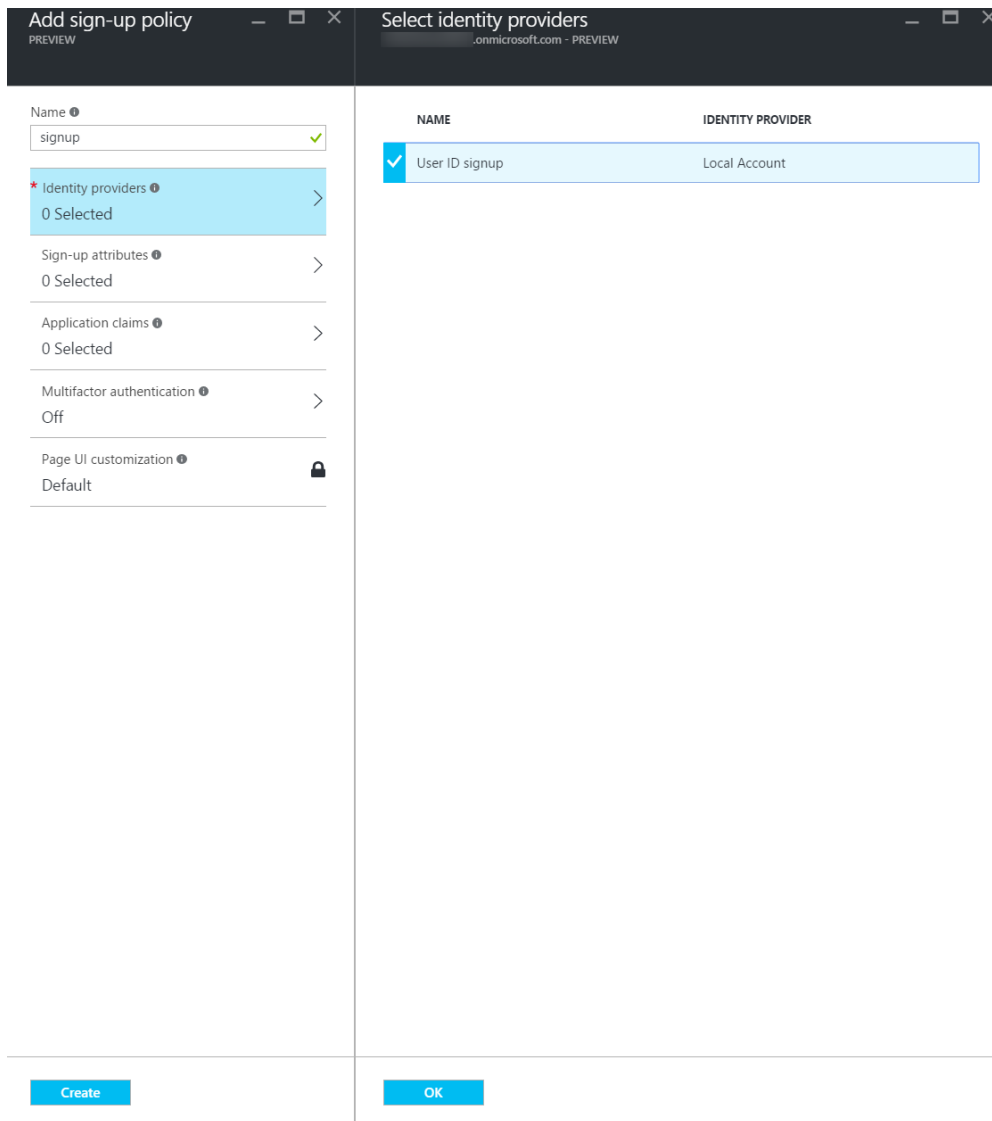
14. In the Settings blade, select **Identity providers**.
15. Select Username for **Local accounts**.
16. Click **Save**.



17. In the Settings blade, select **Sign-up policies**.
18. Click **+ Add**.
19. Set the policy name to "**signup**".



20. Select Identity providers.
21. Select User ID signup.
22. Click **OK**.



23. Select Sign-up attributes.
24. Select Email Address, Given Name and Surname.
25. Click **OK**

Add sign-up policy

PREVIEW

Name ⓘ
signup ✓

* Identity providers ⓘ
1 Selected >

Sign-up attributes ⓘ
0 Selected >

Application claims ⓘ
0 Selected >

Multifactor authentication ⓘ
Off >

Page UI customization ⓘ
Default >

Create

Select sign-up attributes

PREVIEW

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

OK

26. Select Application Claims.

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

28. Click **OK**.

Add sign-up policy

PREVIEW

Name ⓘ
 ✓

* Identity providers ⓘ >
 1 Selected

Sign-up attributes ⓘ >
 3 Selected

Application claims ⓘ >
 0 Selected

Multifactor authentication ⓘ >
 Off

Page UI customization ⓘ >
 Default

[Create](#)

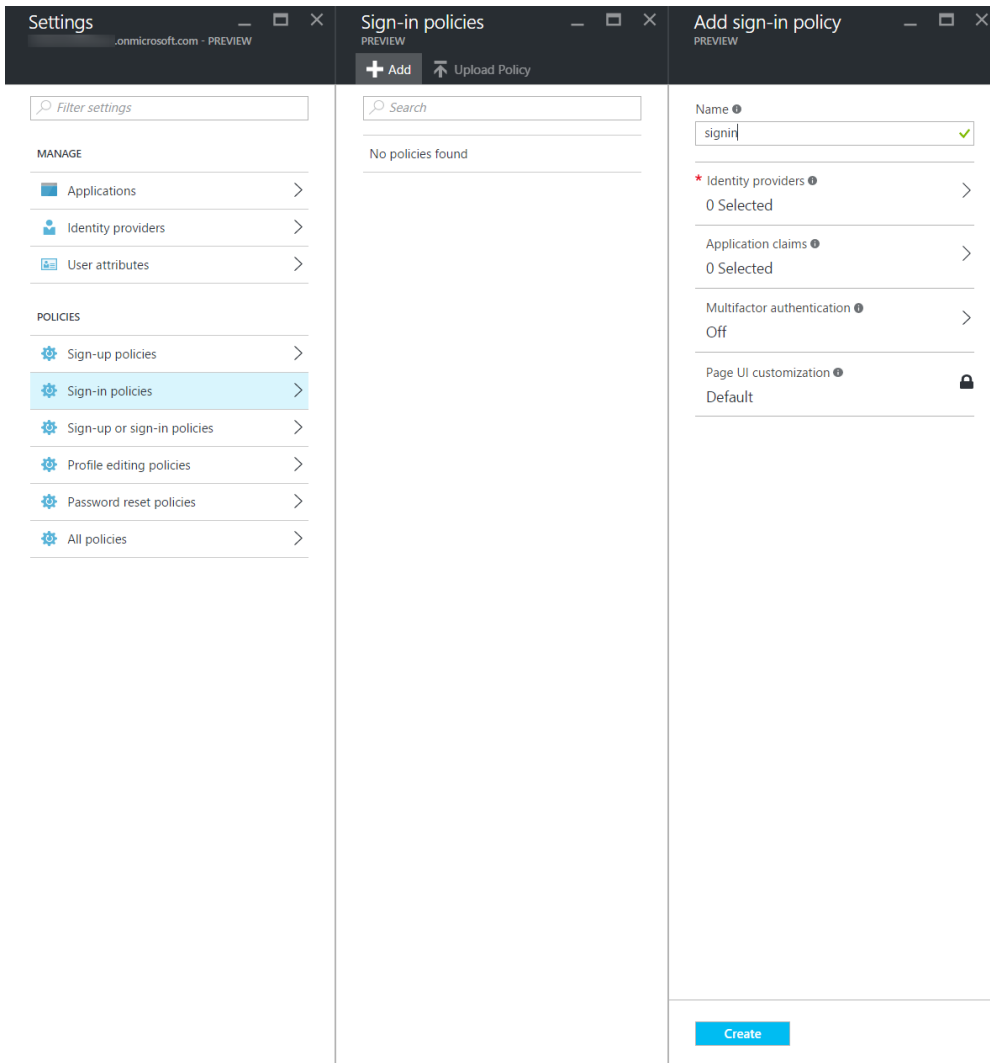
Select application claims

PREVIEW

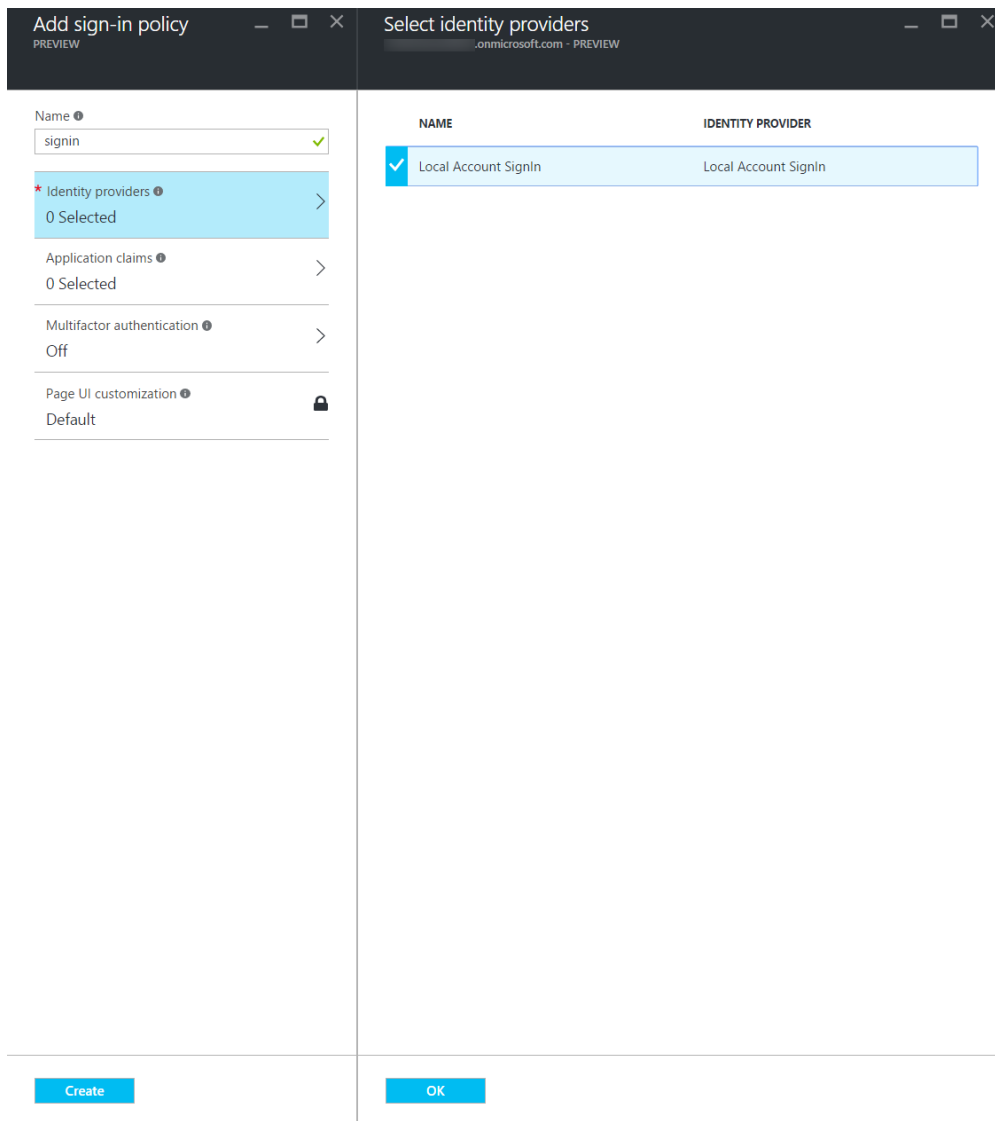
NAME	CLAIM TYPE	DATA TYPE	DESCRIPTION
<input checked="" type="checkbox"/> City	city	String	The city in which the user is located.
<input type="checkbox"/> Country/Region	country	String	The country/region in which the user is located.
<input type="checkbox"/> Display Name	displayName	String	
<input checked=""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

[OK](#)

29. Click **Create**.
30. In the Settings blade, select Sign-in policies.
31. Click **+ Add**.
32. Set the policy name to "**signin**".



33. Select Identity providers.
34. Select Local Account Signin.
35. Click **OK**.



36. Select Application Claims.
37. Select Email Addresses, Given Name, Surname and User's Object ID.
38. Click **OK**

Add sign-in policy
PREVIEW

Name ▼
signin ✓

* Identity providers ▼
1 Selected

Application claims ▼
0 Selected

Multifactor authentication ▼
Off

Page UI customization ▼
Default

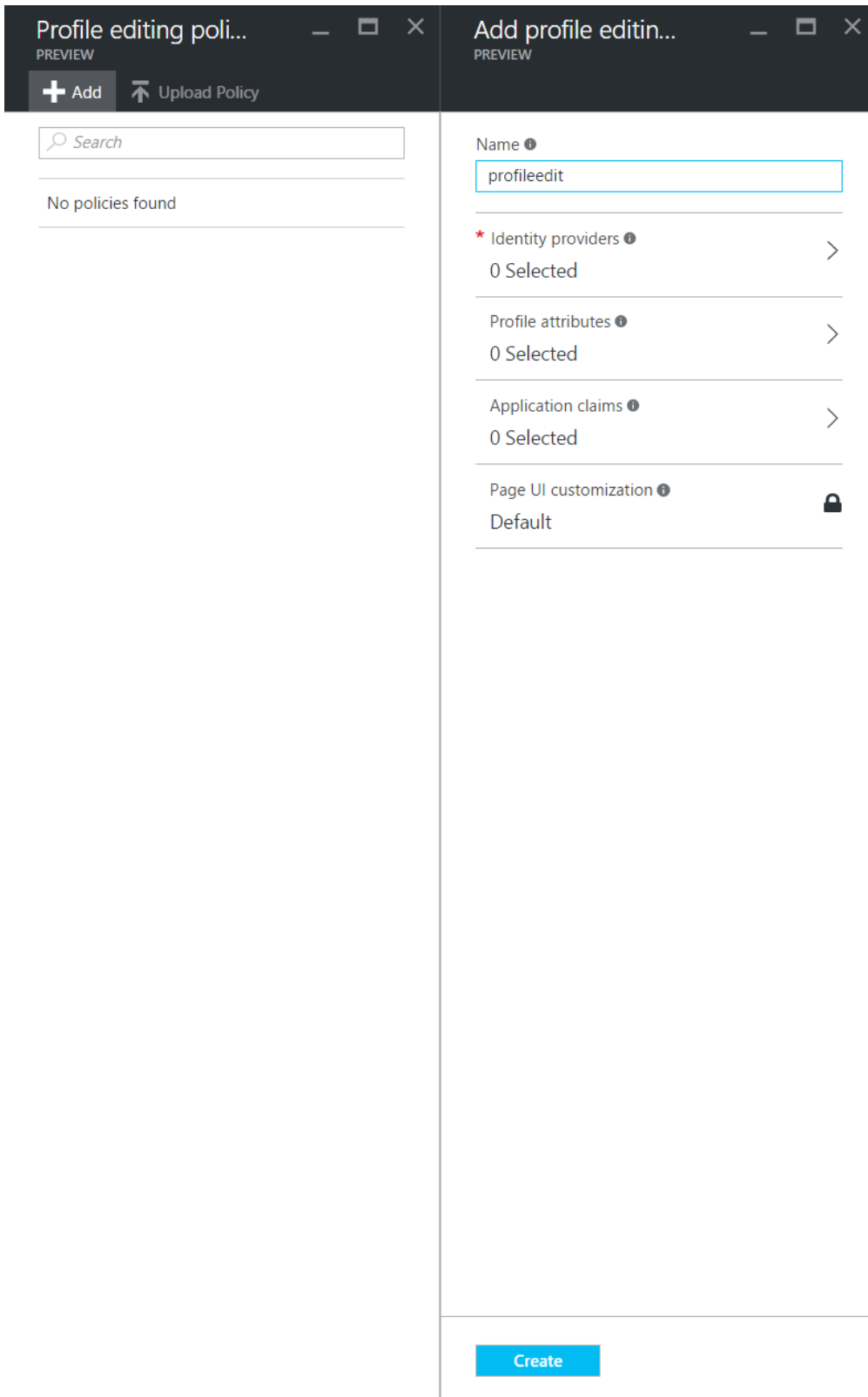
Create

Select application claims
PREVIEW

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

OK

39. Click **Create**.
40. In the Settings blade, select Profile editing policies.
41. Click **+ Add**.
42. Set the policy name to "**profileedit**".



43. Select Identity providers.
44. Select Local Account **Signin**.
45. Click **OK**.

Add profile editin...
PREVIEW

Name

profileedit

Identity providers

0 Selected

Profile attributes

0 Selected

Application claims

0 Selected

Page UI customization

Default

Create

Select identity providers
onmicrosoft.com - PREVIEW

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

OK

46. Select Profile attributes.
47. Select Given Name and Surname.
48. Click **OK**.

Add profile editin...
PREVIEW

Name ⌵
profileedit ✓

* Identity providers ⌵
1 Selected

Profile attributes ⌵
0 Selected

Application claims ⌵
0 Selected

Page UI customization ⌵
Default 🔒

Select profile attributes
PREVIEW

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

Create
OK

49. Select Application Claims.

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

51. Click **OK**.

Add profile editin...
PREVIEW

Name ●
profileedit ✓

* Identity providers ●
1 Selected >

Profile attributes ●
2 Selected >

Application claims ●
0 Selected >

Page UI customization ●
Default >

Create

Select application claims
PREVIEW

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

OK

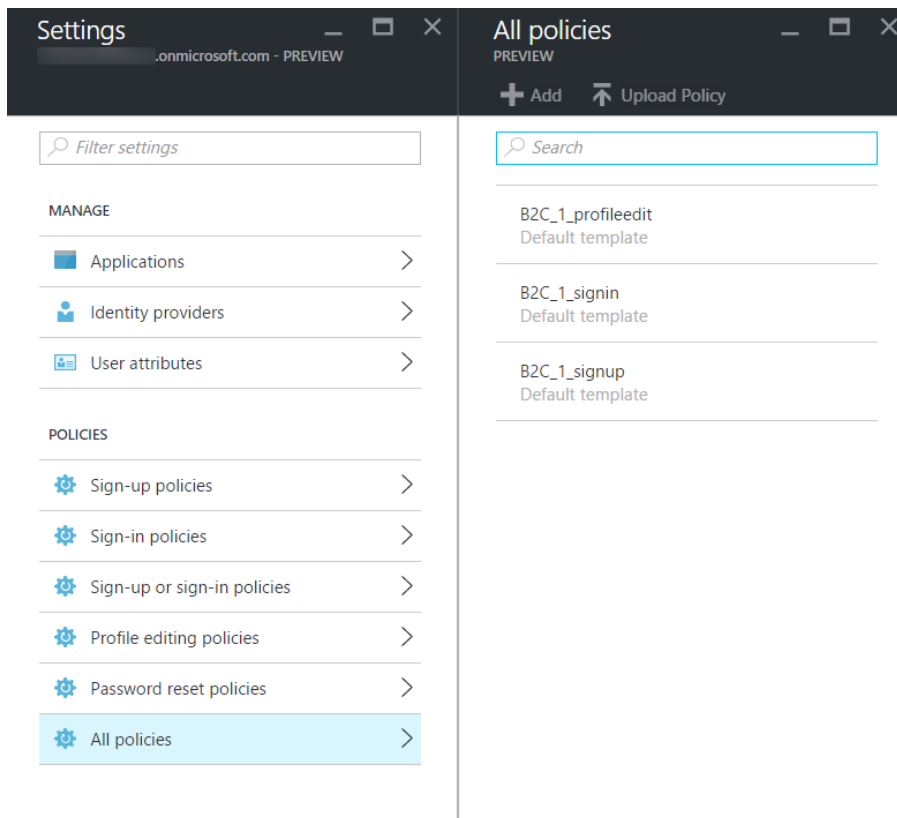
52. Click Create.

53. In the Settings blade, select Applications.

54. Select the created app.

55. Copy and paste the Application ID (to notepad for later) as you will need this for the next Task.

56. In the **Settings** blade, select **All Policies**.



Exit criteria

- Your B2C directory is ready for use. You should see three policies and the B2C instance. Take note of the names for these policies with the prefix 'B2C_1_' as these names will be confirmed in the web app's settings.

Task 2: 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.

Tasks to complete

1. Within Visual Studio Code, expand the **\Web App** folder and open Web.config. You will update these app settings in this file:

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

API Management

1. For the **apimng:BaseUrl** enter the base URL of the API you created in API Management such as <https://contosoeventsSUFFIX.azure-api.net/events/>
Note: Make sure to include the trailing "/" (slash)
2. For the **apimng:SubscriptionKey** enter the subscription key you revealed in API Management developer portal and saved earlier

Azure Active Directory B2C

8. Within the same Web.config file, you will update the following settings:

```
<add key="ida:Tenant" value="[your_domain].onmicrosoft.com" />
<add key="ida:ClientId" value="[application_id]" />
<add key="ida:RedirectUri" value="https://[webapp_host]/" />
```

9. Replace **[your_domain]** in **ida:Tenant** with the value you entered earlier in the B2C tenant creation task.
10. Take the **Application ID** you saved earlier, and paste it in as the value for **ida:ClientId**
11. Lastly, replace the value in **ida:RedirectUri** with the full URL of the Web App (which is also the Reply URL set in the B2C Application settings. (be sure to use HTTPS).

Exit criteria

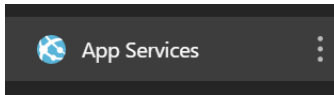
- You should have values for the API Management in app settings.
- The Azure AD B2C settings were applied in app settings.

Task 3: Publish the web app

In this task you will publish the Web application to the Web App instance that was created in App Services.

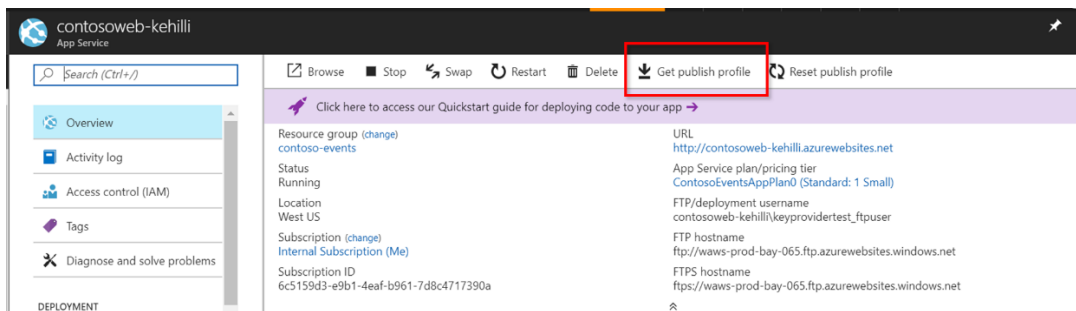
Tasks to complete

12. In the Azure Portal menu, click on **App Services**



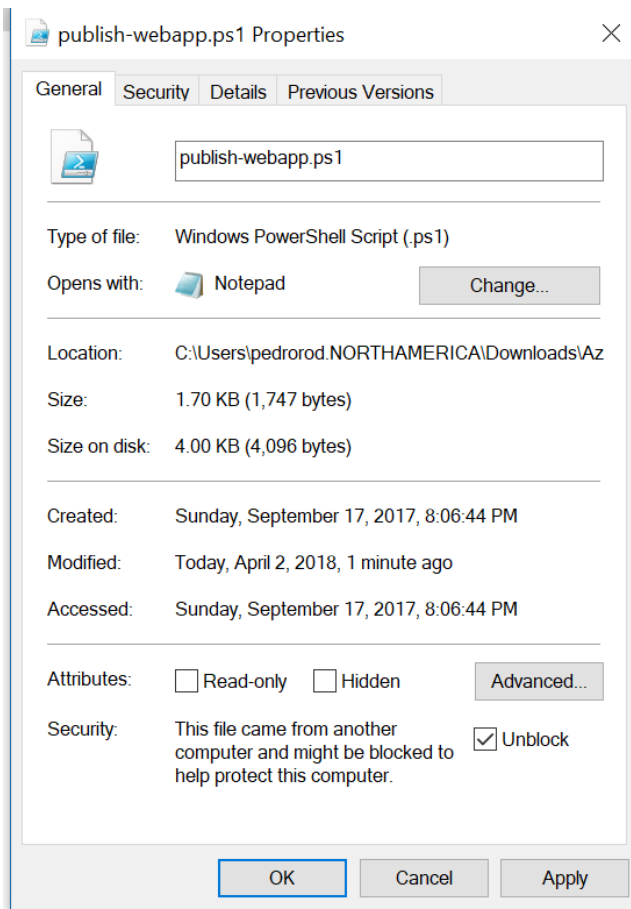
13. Next, download the publish profile to the folder that you unzipped to the desktop, and rename it **“site.PublishSettings”**

NOTE: the file should be in the same folder as “Service Fabric”, “Web App”, etc.



Name	Date modified	Type	Size
site.PublishSettings	17-Sep-17 3:32 PM	PUBLISHSETTINGS File	2 KB

14. Locate the publish-webapp.ps1 file within your lab solution. Unblock the file by right-clicking on the file, selecting properties and then clicking the Unblock checkbox:



15. Open PowerShell (or within Visual Studio Code), and execute the deployment script for the Web App by typing the following:

```
.\publish-webapp.ps1
```

16. Open PowerShell (or within Visual Studio Code), and execute the deployment script for the Web App.

Exit criteria

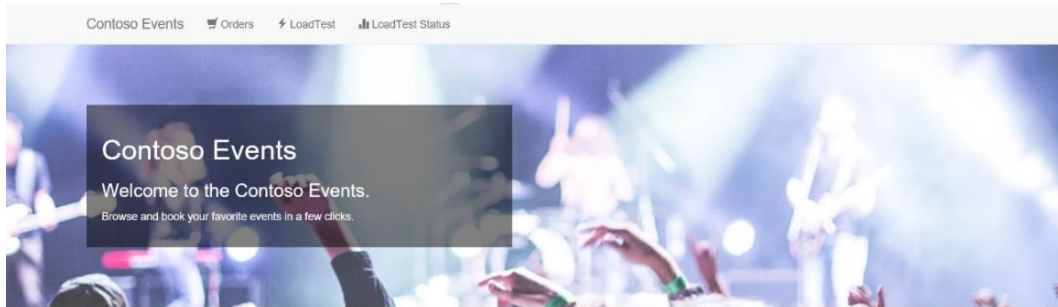
- When publishing is complete, launch a browser, and navigate to the deployed Web app home page.

Task 4: Running the Web App and creating an order

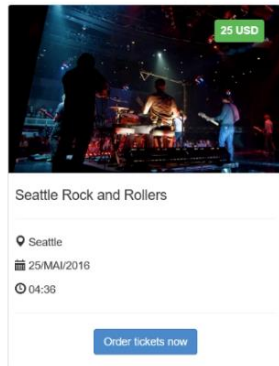
In this task you will test the Web application calls to API Management by creating an order through the UI.

Tasks to complete

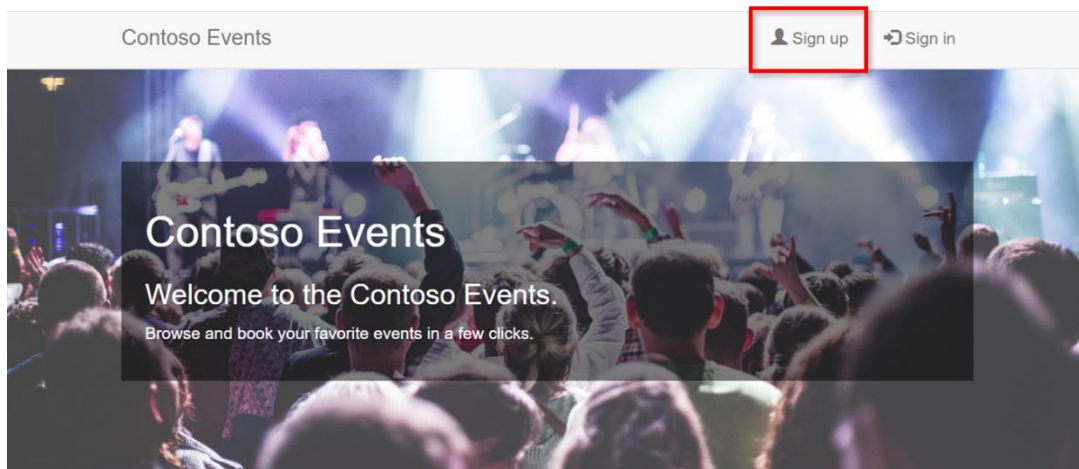
1. Using a browser, launch the website (use HTTPS).
2. When the application launches you will see the website home page as shown in the following screen shot.



On Sale Now: *Seattle Rock and Rollers*



3. Note the event presented on the home page has an Order Tickets Now button!
4. Create a new account to proceed, and click **Sign up**



On Sale Now: *Seattle Rock and Rollers*



- Once you have an account, Click **Order Tickets now** to place an order.
- Choose the number of tickets for the order, then scroll down to see the billing fields.

Seattle Rock and Rollers



📍 Location : Seattle

📅 Date : 25/may/2016

🕒 Time : 04:36

💰 PricePerTicket : 25 USD

Order

🎫 Number of tickets :

1

💰 Total Price : 25 USD

- Enter the empty fields with an email, first name and last name.

📄 Billing

Email :

johnsmith@contoso.com

Phone Number :

425 123 4567

First Name :

John

Last Name :

Smith

Address :

1 Microsoft Way

City :

Redmond

Postal Code :

WA 98052

Country :

US

🗳️ Credit Card

Cardholder Name :

John Smith

Card Number :

1234567890123456

Expiration Month :

abril

Expiration Year :

2017

Security Code :

123

Place Order

- Click **Place Order**

Exit criteria

- 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.



Exercise 5: Load testing (bonus)

Duration: 15 minutes

If you're done early, this is a bonus task that is optional

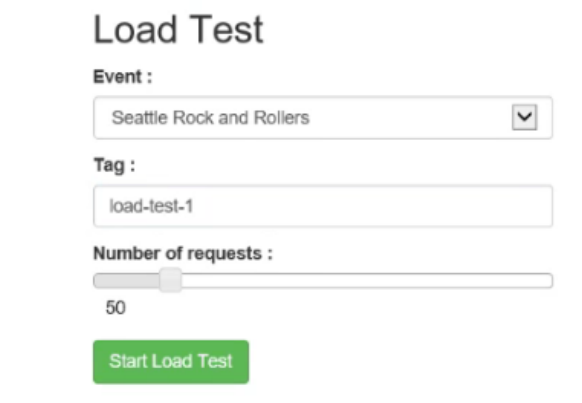
In this exercise you will perform a load tests against the Service Fabric Cluster and observe how messages are distributed across partitions.

Task 1: Simulate a 50 order request

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

Tasks to complete

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



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

Simulation Status

Partition Id	Partition Status	Node Name	Health State	Items in Queue
075e7e0f-3f3b-47a9-af64-0aeb30d01bf7	Ready	_Web_4	Ok	0
1b057210-b8cf-4234-8295-1d0b6caa9e8c	Ready	_Web_3	Ok	0
ec94a533-3165-4514-b4d4-07935048eb25	Ready	_Web_1	Ok	0
18a54270-0542-4990-8efc-bb61d1843108	Ready	_Web_2	Ok	0
5591d8d0-dd09-49b0-9acc-7098beb7ea63	Ready	_Web_0	Ok	0

4. While the load test is running, refresh this page and watch the changes to the Items in queue across partitions. It will fill while processing completes and then eventually drain.

Exit criteria

- After a few minutes you will see that the queues are drained and the orders were processed.

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

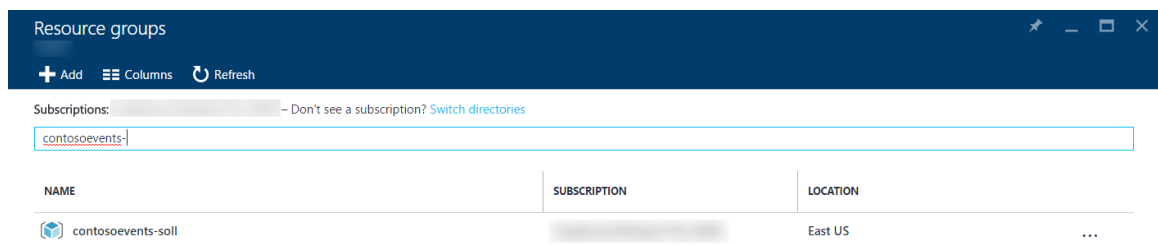
Exercise 6: Cleanup

Duration: 5 minutes

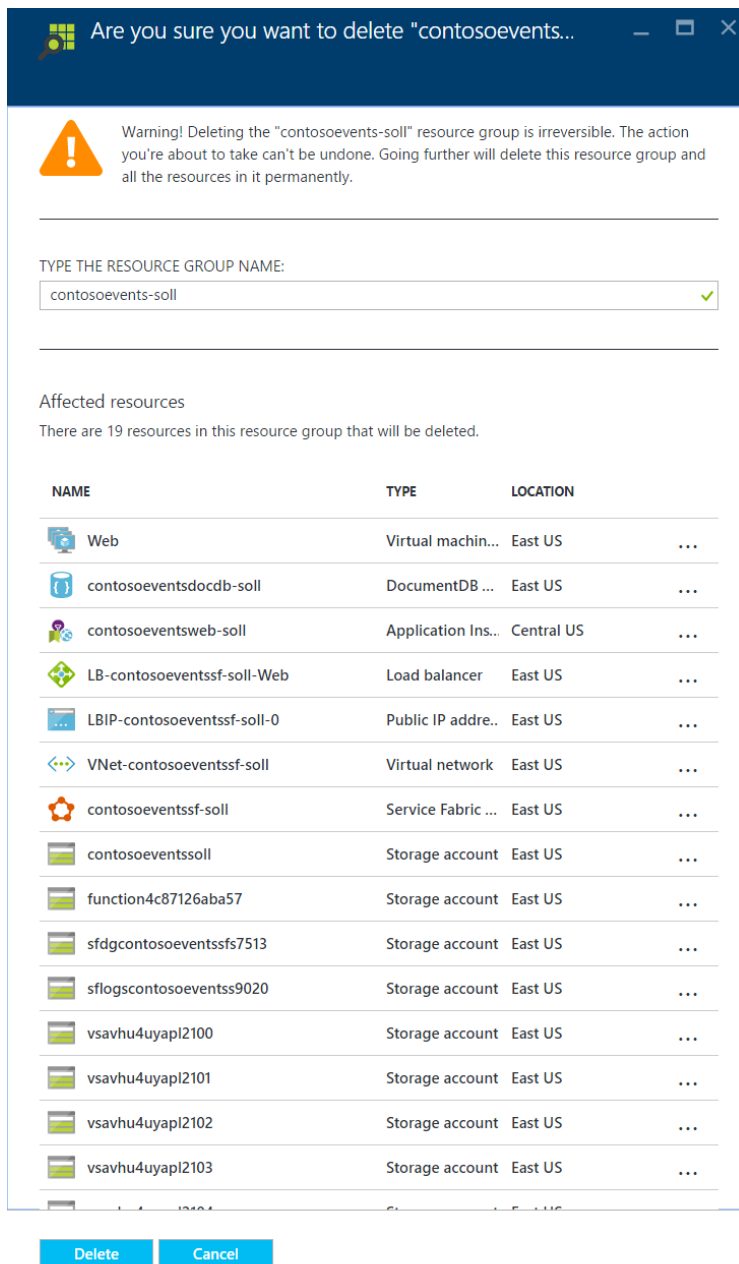
In this exercise, attendees will de-provision any Azure resources that were created in support of the lab.

Tasks to complete

1. Go to the Azure Portal
2. Find the first Resource Group you created for this exercise.



17. Select the Resource Group to delete.
4. Click the Delete menu from the resource blade.
5. In the delete confirmation blade, type the Resource Group name
6. click Delete. You will be able to see all of the resources allocated to the group before you confirm.



Exit criteria

- After all of the deletion tasks are complete, the resources will no longer be listed in the Azure Portal or the Management Portal. This may take about 10 minutes. You can optionally wait to see a confirmation. Otherwise, you are done!