WorkshopPLUS

Microsoft Azure Service Fabric for Developers

Service Fabric End-to-End Services

Student Lab Manual

V2.0, March 15, 2017

**Conditions and Terms of Use**

Microsoft Confidential

This training package is proprietary and confidential, and is intended only for uses described in the training materials. Content and software is provided to you under a Non-Disclosure Agreement and cannot be distributed. Copying or disclosing all or any portion of the content and/or software included in such packages is strictly prohibited.

The contents of this package are for informational and training purposes only and are provided "as is" without warranty of any kind, whether express or implied, including but not limited to the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

Training package content, including URLs and other Internet Web site references, is subject to change without notice. Because Microsoft must respond to changing market conditions, the content should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication. Unless otherwise noted, the 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.

**Copyright and Trademarks**

© 2015 Microsoft Corporation. All rights reserved.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in 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.

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.

For more information, see Use of Microsoft Copyrighted Content at  
<http://www.microsoft.com/en-us/legal/intellectualproperty/Permissions/default.aspx>

Azure, Microsoft, Internet Explorer, and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Other Microsoft products mentioned herein may be either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. All other trademarks are property of their respective owners.

Contents

[Working with Service Fabric Services 4](#_Toc474835839)

[Objective 4](#_Toc474835840)

[Prerequisites 4](#_Toc474835841)

[Overview of the Lab 4](#_Toc474835842)

[Scenario 4](#_Toc474835843)

[Data Flow 5](#_Toc474835844)

[Task 1 – Create an Azure Storage Account 6](#_Toc474835845)

[Task 2 – Build a Service Fabric Cluster in Azure 9](#_Toc474835846)

[Task 3 – Update code in the Web.Service project 14](#_Toc474835847)

[Task 4 – Update code in the CustomerOrder.Actor project 15](#_Toc474835848)

[Task 5 – Update code in the Inventory.Service project 16](#_Toc474835849)

[Task 6 – Update code in the RestockRequest.Actor project 17](#_Toc474835850)

[Task 7 – Update code in the RestockRequestManager.Service project 17](#_Toc474835851)

[Task 8 – Deploy the application to your Service Fabric Cluster in Azure 18](#_Toc474835852)

# Working with Service Fabric Services

Estimated time to complete this lab

60 minutes

## Objective

After completing this lab, you will be able to:

* Create an end-to-end Service Fabric application
* Understand the concepts of Reliable Services and Reliable Actors and use them together
* Use Visual Studio and Service Fabric Explorer to efficiently develop Service Fabric applications and deploy to the Service Fabric Cluster in Azure or the local development cluster

## Prerequisites

The following is required to complete this hands-on lab:

* Microsoft V[isual Studio 2017 Professional or Enterprise edition](http://www.microsoft.com/visualstudio/)
* [Microsoft Azure SDK for .NET for Visual Studio 2017](http://www.microsoft.com/windowsazure/sdk/)
* Microsoft Azure Service Fabric SDK – 2.4.164
* Microsoft Azure PowerShell
* A Microsoft Azure subscription

Note that the same resource group will be used for all labs.

## Overview of the Lab

The web reference application shows how to build an end-to-end Service Fabric application with multiple types of services, combining reliable services and reliable actors to construct a complete solution.

## Scenario

The context of this scenario is a web-based store with a customer order and inventory management back-end. Logical parts of the management back-end are represented by individual services, allowing loose coupling of functionality and independently-upgradeable components:

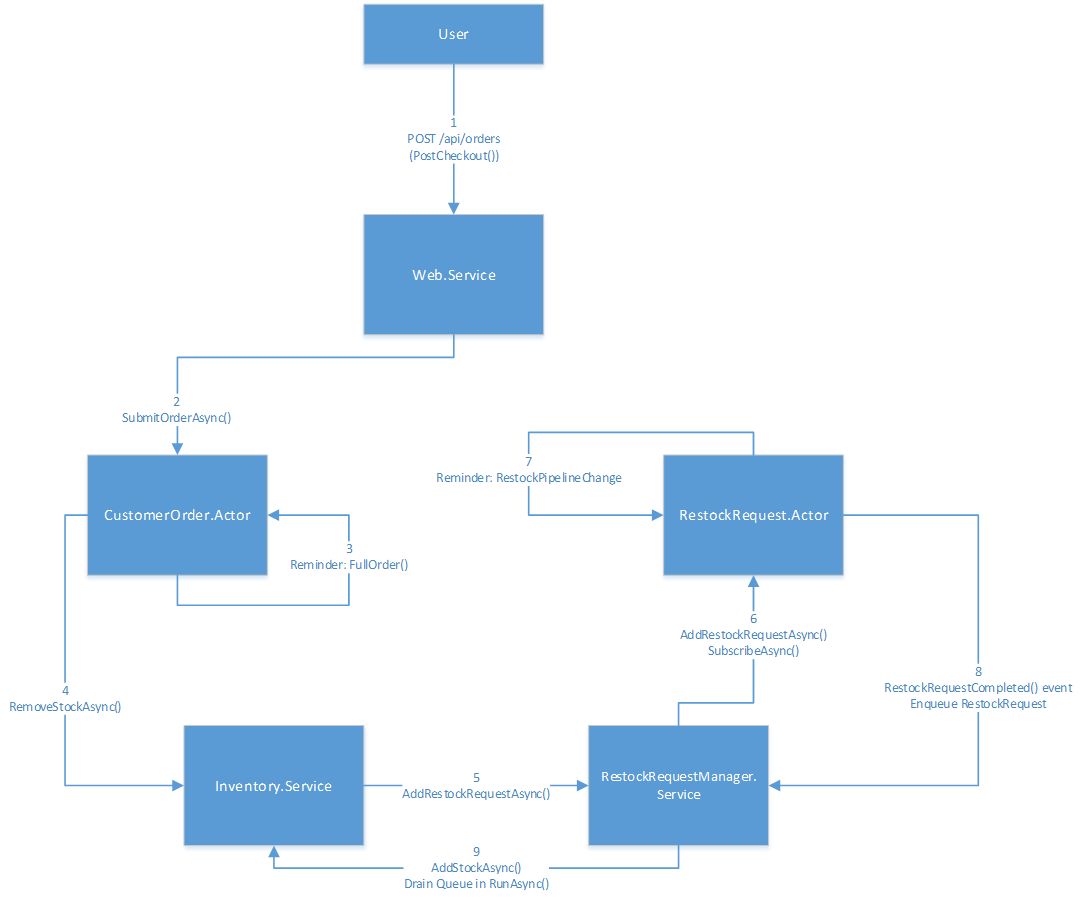
* Customer Order Service
* Inventory Service
* Restocking Service
* Web front-end Service

The customer order and inventory management system tracks user orders, removes items from the inventory to fulfill orders, and requests restocking of inventory items when an item's stock goes below a certain threshold. If a user requests items that are out of stock, the order is placed on back-order until the inventory is replenished, at which point the order is completed.

Using Service Fabric's stateful services, each of these services can maintain its own data, rather than relying a shared monolithic data base. This allows each service to scale independently using Service Fabric's stateful partitioning to meet its unique requirements for data capacity and throughput.

## Data Flow

When a user makes a purchase, data flows the through the system as follows:



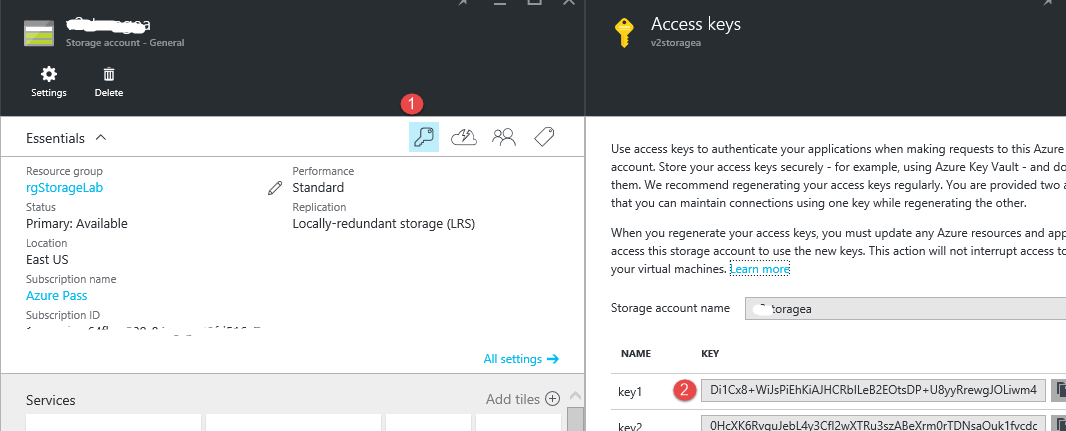
## Task 1 – Create an Azure Storage Account

Azure Storage can take on many forms for storing blobs, non-relational table data and queue data. In the case of this application, we will be using both blob and queue storage. For this, we need to create a new Azure Storage account.

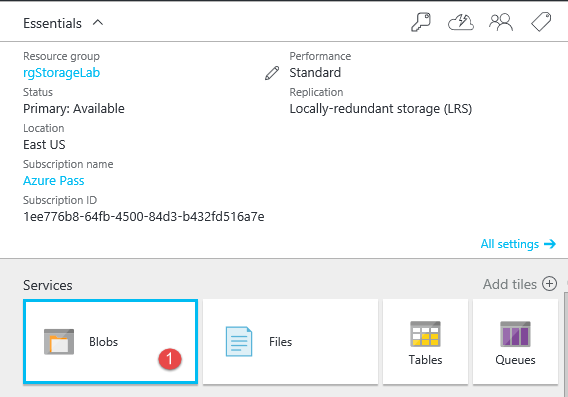
1. Log in to the Azure Portal at <https://portal.azure.com>.
2. Click the +**New** button from the menu on the left hand side of the portal and then select  **Storage** | **Storage Account. NOTE: If you already have a storage account you can use, skip to Task 2.**
3. Enter/select the following selections below.

|  |  |
| --- | --- |
| 1. Enter a unique name (in all of Azure) for your storage account. 2. Leave *Resource manager* selected as the Deployment model 3. Select *General purpose* as the Account kind so we have full access to blobs, tables, queues etc 4. Leave *Standard* selected for Performance 5. Leave *LRS* selected as the Replication type 6. Leave storage data unencrypted for this lab 7. Choose your subscription 8. Use the resource group that you created in previous lab exercises (the name in the screenshot is just a sample)*.* 9. Select your Location (region) 10. Click the **Create** button |  |

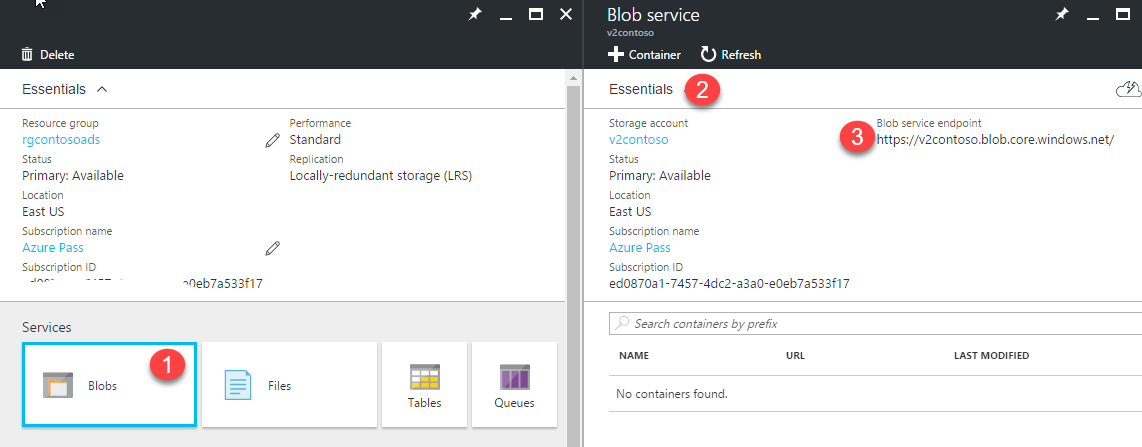
1. You will need your storage account name, key1 and your storage blob key in upcoming steps. Once your storage account has been created, you can click on your storage account name in the resource group, select the Access keys menu item storage account blade to get this information. Copy *key1* and the storage account name to Notepad.



1. You will also need to note the blob endpoint setting. To get this endpoint, click on the **Blobs** icon in the storage account blade.



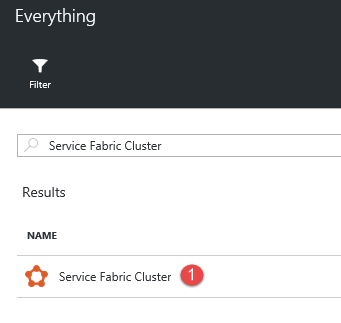
1. In the blob service blade, click on the **Essentials** tab to expand the blob service blade to show the blob service endpoint. Copy this endpoint information to Notepad.



## Task 2 – Build a Service Fabric Cluster in Azure

If you have already created a Service Fabric cluster in Azure in previous lab exercise, please skip to Task 3.

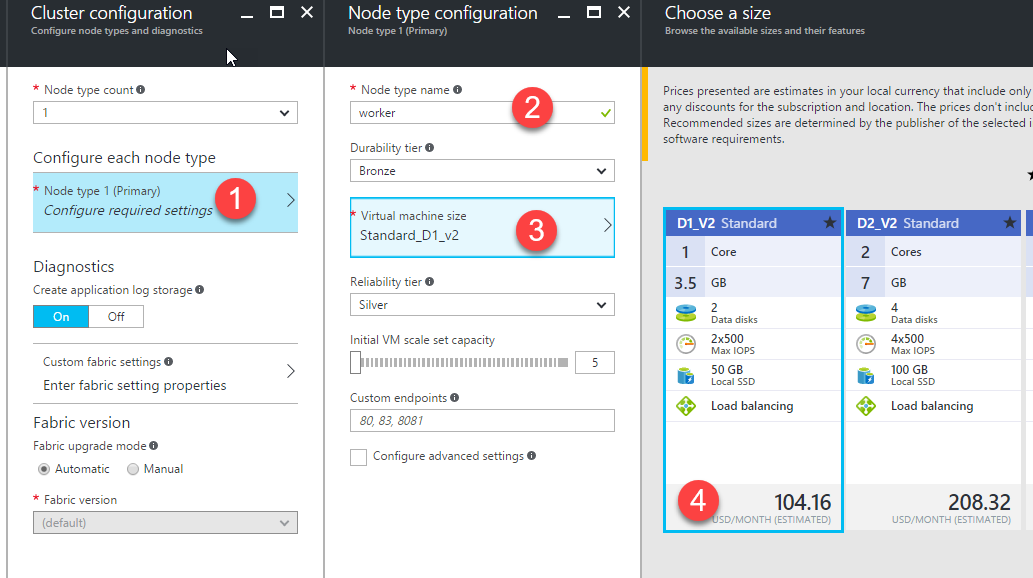
1. Log in to the Azure portal <https://portal.azure.com>.
2. In previous labs, you had already created a resource group, find and open this resource group blade. It is best to find a resource group that does not already have a Service Fabric cluster in it.
3. Within your resource group blade, click on the **Add** button on the toolbar.
4. In the edit field of the *Everything* blade, enter ‘*Service Fabric Cluster’* and select the **Service Fabric Cluster** icon.



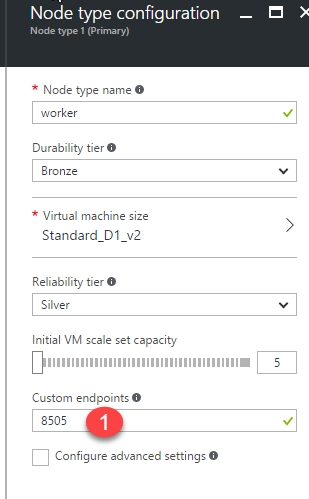
1. Select the **Create** button.
2. For the basic settings, enter the following information:

|  |  |
| --- | --- |
| Ee   1. Enter a unique cluster name (within the Azure region). 2. Choose Windows as the operating system 3. Enter the username you would use to RDP into the machine. 4. Enter your RDP password. 5. Select your Azure subscription name 6. Select a resource group, in this case, the one you have been using for labs up to this point. 7. Select your region.   Select the **OK** button. | C:\Users\larrywa\AppData\Local\Temp\SNAGHTML4454810.PNG |

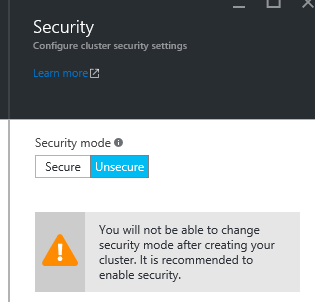
1. For the Cluster Configuration, set the *Node type count* to **1** and then click on the *Node type 1 (Primary*) link. You can give the node type name any name you like and choose any machine size you like but leave the rest of the settings as they are. Click the **Select** button in the *Choose a size* blade once you select your machine size.



1. While still in the *Node type configuration* blade, remember that the Web.Service project requires that port 8505 be opened. Type **8505** into the *Custom endpoints* field and then select the **OK** button.



1. Select the **Ok** button on the *Cluster configuration* blade.
2. On the *Security* blade, select the **Unsecure** *Security mode*. Normally, we would have a secure cluster but for the sake of brevity for the lab, we will use unsecure. Select the **OK** button.



1. Select the **Ok** button in the *Summary* blade. Creating the cluster should take about 20 minutes.

## Task 3 – Update code in the Web.Service project

1. In your **.\Labs\Begin** folder, you will see a folder named ‘**WebRefStudents’**. Copy this entire folder to your root drive level, otherwise, some of the NuGet Packages will fail to download because the path exceeds 248 characters.
2. Open Visual Studio as an Administrator and open the **.\WebRefStudents\ReferenceApp\WebReferenceApp.sln** solution. If you are prompted to restore the required NuGet packages, accept Yes.
3. Rebuild the solution.
4. The **Web.Service** project is a stateless front-end service that hosts the web UI and HTTP API for interacting with the store. Open the **Controllers\InventoryController.cs** and in the **CreateInventoryItem** method, replace:  
     
   IInventoryService inventoryServiceClient = null;

With

IInventoryService inventoryServiceClient = ServiceProxy.Create<IInventoryService>(builder.ToUri(), i.Id.GetPartitionKey());

This code creates the proxy to talk with and allow discovery of partition endpoints on the inventory service.

1. Open **Controllers\OrderControllers.cs** and update the *CustomerOrderServiceName* const class variable to:  
     
   private const string CustomerOrderServiceName = "CustomerOrderActorService";

This code lets OrdersController know the name of the actor that is associated with the order.

1. Open **Controllers\StoreController.cs** and update the *InventoryServiceName* const class variable to:  
     
   public const string InventoryServiceName = "InventoryService";

If you look in to the Task method, you can see that this is where partitions get queried.

1. Still within the Web.Service project, browse to and open the **PackageRoot\ServiceManifest.xml** file. Notice in this file, the http endpoint and the port number **8505** that we will have to have open on the nodes to communicate with this service.

## Task 4 – Update code in the CustomerOrder.Actor project

The CustomerOrder.Actor project is an actor-based service that handles customer orders. A stateful actor is activated for each new order that's placed. The actor represents the lifetime of the customer order, from placement to fulfillment.

1. In the **CustomerOrder.Actor** project, open **CustomerOrderActor.cs** and update the *InventoryServiceName* const class variable to:  
     
   private const string InventoryServiceName = "InventoryService";

You should also quickly review the code in the methods OnActivateAsync(), FullfillOrderAsync(), RetrieveReminderAsync(), GetStatusAsync() and SubmitOrderAsync(). These methods are well commented to gain an understand of the operation of each method.

1. Open the **CustomerOrderReminderNames.cs** file and update the *FulfillOrderReminder* const class variable to:  
     
   public const string FulfillOrderReminder = "FulfillOrderReminder";
2. Still within the CustomerOrder.Actor project, browse to and open the **PackageRoot\ServiceManifest.xml** file. Notice in this file we only have internal endpoints and that we also have a ‘replicator’ endpoint which is used, since this actor service is stateful and will have partitions.

## Task 5 – Update code in the Inventory.Service project

The Inventory.Service project is a stateful service that maintains the store's inventory. This service is partitioned, where each partition of the service holds a subset of the store's entire inventory. With a large number of partitions, this service can scale out to meet data capacity and inventory request throughput. This service provides most of the functionality for the scenario.

1. In the **Inventory.Service** project, open **InventoryService.cs** file. There are several class variables at the top of the class that need to be updated to:

internal const string InventoryServiceType = "InventoryServiceType";

private const string InventoryItemDictionaryName = "inventoryItems";

private const string ActorMessageDictionaryName = "incomingMessages";

private const string RestockRequestManagerServiceName = "RestockRequestManager";

private const string RequestHistoryDictionaryName = "RequestHistory";

private const string BackupCountDictionaryName = "BackupCountingDictionary";

1. Edit the **Inventory.Service\PackageRoot\Config\Settings.xml** file. This file contains the connection details for your Azure storage account. Use the information from the Azure storage account you saved at the beginning of this lab.

<Parameter Name="BackupAccountName" Value="Your Storage Account Name" />

<Parameter Name="PrimaryKeyForBackupTestAccount" Value="Your-Storage Acct Primary Key" />

<Parameter Name="BlobServiceEndpointAddress" Value="EndPoint For Blob Service" />

1. Still within the Inventory.Service project, browse to and open the **PackageRoot\ServiceManifest.xml** file. Notice in this file we only have the standard internal endpoints and that we also have a ‘replicator’ endpoint which is used since this actor service is stateful and will have partitions.

## Task 6 – Update code in the RestockRequest.Actor project

The **RestockRequest.Actor** project is a stateful actor that manages the lifetime of a restock request from the Inventory service. Each time the inventory runs low on stock for an item, makes a request to refill the inventory. The restock request actor itself would send a request to a supplier for more items, however this is simply simulated within the actor.

1. Open the file **ServiceHost.cs** and add the following code inside of the try block:

Try

{ ActorRuntime.RegisterActorAsync<RestockRequestActor>().GetAwaiter().GetResult();

Thread.Sleep(Timeout.Infinite);

}

1. Still within the RestockRequest.Actor project, browse to and open the **PackageRoot\ServiceManifest.xml** file. Notice in this file shows that we only have the standard internal endpoints and that we also have a ‘replicator’ endpoint which is used since this actor service is stateful and will have partitions.

## Task 7 – Update code in the RestockRequestManager.Service project

This is a stateful service that manages requests from the inventory service for item restocking. It logs restock requests made by the Inventory Service and activates a RestockRequest actor to fulfill the request. It then receives notifications from the actors when a restock request has been fulfilled. These notifications are placed in a ReliableQueue as they come in. The notifications are periodically de-queued and sent back to the Inventory Service.

1. In the RestockRequestManager.Service project, open the file **RestockRequestManagerService.cs** and update the class level variables to the following values:

private const string ItemIdToActorIdMapName = "actorIdToMapName";

private const string CompletedRequestsQueueName = "completedRequests";

private const string InventoryServiceName = "InventoryService";

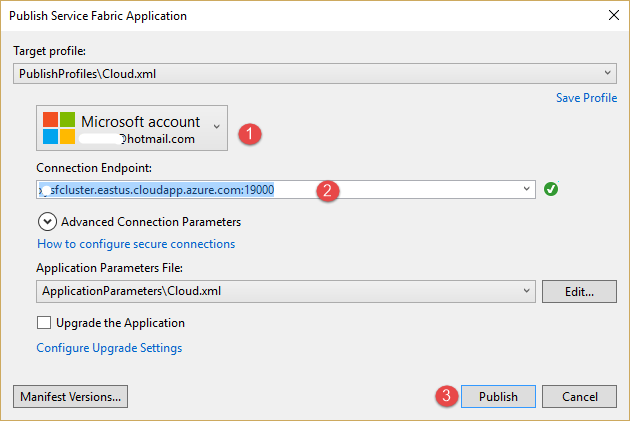
A couple of things to note about this class:

* Notice service inherits from the class StatefulService
* Notice the service inherits from these interfaces IRestockRequestManager, IRestockRequestEvents

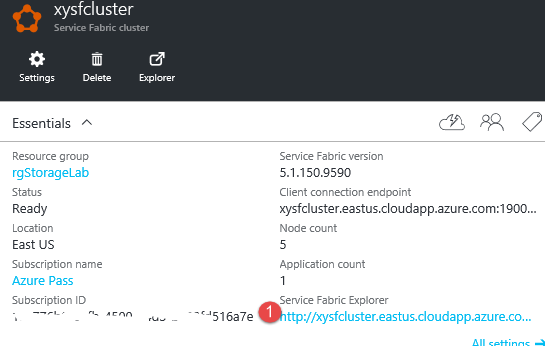
1. Still within the RestockRequestManager.Service project, browse to and open the **PackageRoot\ServiceManifest.xml** file. Notice in this file shows that we only have the standard internal endpoints and that we also have a ‘replicator’ endpoint which is used since this actor service is stateful and will have partitions.

## Task 8 – Deploy the application to your Service Fabric Cluster in Azure

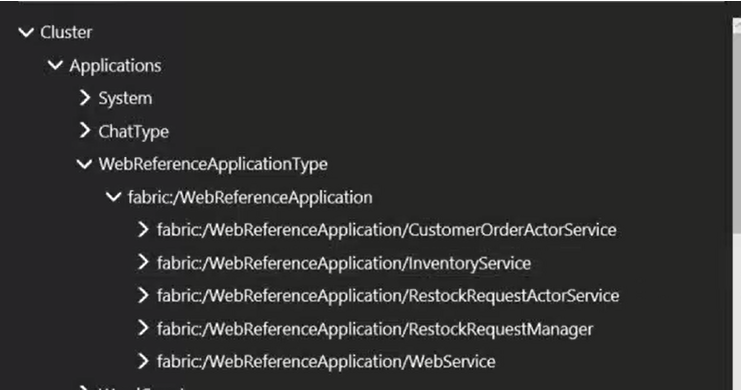
1. The application can be deployed to Azure by right-clicking the **WebReferenceApplication** project in Visual Studio and selecting "**Publish**".
2. In the *Publish Service Fabric Application* dialog box, make sure you have logged in to your subscription and then (if the cluster has completed the creation process), you can select your cluster from the **Connection Endpoint** dropdown. Select the **Publish** button.



1. After Deployment is completed, open up the Service Fabric Explorer from the Azure Portal. You can get to the explorer by going into your resource group, selecting your cluster name and then on the cluster blade, select the **Service Fabric Explorer** link.



1. Navigate to the Cluster, locate the WebReferenceApplicationType, locate the **WebReferenceApplication**, and see all the Services that got deployed.



1. You will be able to get to the following pages using the respective URLs. Please replace the values for the cluster name and if you used a different region than eastus, please use that region.

Go to http://<YourServiceFabricClusterName.eastus.cloudapp.azure.com>:8505/fabrikam/admin.html - a very basic admin portal where you can add items into the inventory. When you first launch the application, the inventory is empty. Copy one line of the sample data into the textbox and press the “**Create Inventory**” button to create that inventory (or make up your own in the provided format). (NOTE that the first line you see on the screenshot below, is the header line like you’d see in a table, don’t paste that in to the textbox).



1. Go to http://<YourServiceFabricClusterName.westus.cloudapp.azure.com>:8505/fabrikam/ to see the basic store front-end. This shows the current inventory and your shopping cart where you need to add and purchase items to see the flow of data through the system. Make sure you have the ‘/’ at the end of the address!!

