WorkshopPLUS

Microsoft Azure Service Fabric for Developers

Service Fabric Internals

Student Lab Manual

V1.9, March 15, 2017

Information in this document is subject to change without notice. The example companies, organizations, products, people, and events depicted herein are fictitious. No association with any real company, organization, product, person 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, trademarked, 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.

© 2010 Microsoft Corporation. All rights reserved.

Microsoft, MS-DOS, MS, Windows, Windows NT, MSDN, Active Directory, BizTalk, SQL Server, SharePoint, Outlook, PowerPoint, FrontPage, Visual Basic, Visual C++, Visual J++, Visual InterDev, Visual SourceSafe, Visual C#, Visual J#,  and Visual Studio are either registered trademarks or trademarks of Microsoft Corporation in the U.S.A. and/or other countries.

Other product and company names herein may be the trademarks of their respective owners.

Contents

[Running non-Service Fabric aware application in a Service Fabric Cluster 4](#_Toc476729950)

[Introduction 4](#_Toc476729951)

[Prerequisites 4](#_Toc476729952)

[Overview 4](#_Toc476729953)

[Task 1 – Creating the basic service to run our command line web listener 5](#_Toc476729954)

[Task 2 – Setting up PowerShell to run the deployment script 10](#_Toc476729955)

[Task 3 – Correcting the endpoint security issue 13](#_Toc476729956)

[Task 4 – Exploring our applications behavior 15](#_Toc476729957)

# Running non-Service Fabric aware application in a Service Fabric Cluster

## Introduction

Estimated time to complete this lab

45 minutes

After completing this lab, you will be able to:

* Understand how to package and deploy an non-Service Fabric aware application into a Service Fabric cluster
* Edit the associated manifest files for an application

## 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
* A Microsoft Azure subscription

Please note, the same Resource Group will be used for all Labs.

## Overview

The process of packaging an existing application is based on the following steps:

1. Create the package folder structure
2. Add the application's code and configuration files
3. Edit the service manifest file
4. Edit the application manifest file

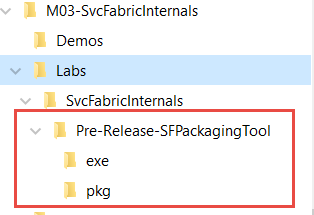
This Lab consists of three tasks:

1. Creating the basic service to run our command line web listener
2. Fixing an endpoint security issue
3. Exploring our applications behavior

## Task 1 – Creating the basic service to run our command line web listener

A Service Fabric application package can be created by collecting all the required files and manually packaging them, however, to make this process easier, a tool has been developed which will provide the package structure and base configuration files for you to work with. The pre-release Service Fabric packing tool is available for download [here](http://aka.ms/servicefabricpacktool), but for convenience, it has been provided in your lab folder. Look under the **.\Labs\SvcFabricInternals\Pre-Release-SFPackagingTool** sub-directory for the ***ServiceFabricAppPackageUtil.exe*** file.

1. Using Windows Explorer, browse to the **.\Labs\SvcFabricInternals\Pre-Release-SFPackagingTool** directory. In this directory, you will see two sub-directories, ‘*exe’* and ‘*pkg’*. In the ‘exe’ folder, you will find the application, **HostedWebServer.exe**, that you will deploy out to the Service Fabric cluster. The ‘pkg’ folder will be empty until you build the package to send out to the cluster. In the root directory, you will find ServiceFabricAppPackageUtil.exe. Note that the HostedWebServer.exe app is a non-Service Fabric aware application.



1. To create the package directory structure and configuration files for the application, simply open a command prompt and navigate to the directory containing **ServiceFabricAppPackageUtil.exe**, and enter a command line like the following:

Syntax:

ServiceFabricAppPackageUtil.exe /source:<directory where your exe is located> /target:<your package directory> /appname:<application name> /exe:<application name including .exe extension>

Example:

<Path>ServiceFabricAppPackageUtil.exe /source:.\exe /target:.\pkg /appname:HostedWebServer /exe:HostedWebServer.exe

**Note**:

* + Make sure that the **target** directory is at the same level or higher than the ***exe*** directory or the application will crash.
  + If you are writing to a location other than your machines user folders, make sure you run as administrator, otherwise you will get an access denied error
  + Remember the location you specify as “target”. This is where your package will be after a successful run of the utility. You should end up with a set of folders with *ApplicationManifest.xml* at the root, and one or more folders with the names of your services. Within the sub-folder of the package will be the *ServiceManifest.xml* file, and a directory named “**C**” containing the executable code which your service will run. The executable file **HostedWebServer.exe** which you pointed to in the above command is copied here. There is also a config folder.
  + After you have created the directory structure, you can add the application's code and configuration files under the *code* and *config* directories. You can also create additional directories or subdirectories under the **C** or **config** directories. For this lab exercise, you do not need to add any files.
  + Service Fabric does an xcopy of the content of the application root directory, so there is no predefined structure to use other than creating two top directories, code and settings which will be described later.

NOTE:

Make sure that you include all the files/dependencies that the application needs. Service Fabric will copy the content of the application package on all nodes in the cluster where the application's services are going to be deployed. The package should contain all the code that the application needs to run. It is not recommend to assume that the dependencies are already installed. For this simple application, there are no dependencies to install.

1. Open the **ApplicationManifest.xml** file created in the previous step and examine (but don’t modify) the resulting XML. This should be located in the **.\Labs\SvcFabricInternals\Pre-Release-SFPackagingTool\pkg** directory.

In the ServiceManifestImport element, you can specify one or more services that you want to include in the app. Services are referenced with ServiceManifestName, which specifies the name of the directory where the ServiceManifest.xml file is located. In this case, this element should be set to “**HostedWebServer**” with a ServiceManifestVersion of “1.0”. Also check in the DefaultServices element. The *service name* should be set to “**HostedWebServerService**” to match the name of the DefaultService element in the ServiceManifest.xml file. The next element sets up our service to use only one partition, <SingletonPartition />.

1. Open and examine (but don’t modify) the **ServiceManifest.xml.** This should be located under the **.\Labs\SvcFabricInternals\Pre-Release-SFPackagingTool\pkg\HostedWebServer** folder.

The service manifest allows resources that are used by the service to be declared/changed without changing the compiled code. Azure Service Fabric supports configuration of endpoint resources for the service.

Different parts of the file to examine:

**ServiceTypes**

<ServiceTypes>

<StatelessServiceType ServiceTypeName="HostedWebServer" UseImplicitHost="true" />

…(other generated code)

</ServiceTypes>

* You can pick any name that you want for ServiceTypeName. The value is used in the ApplicationManifest.xml file to identify the service.
* You need to specify UseImplicitHost="true". This attribute tells Service Fabric that the service is based on a self-contained app, so all Service Fabric needs to do is to launch it as a process and monitor its health.

**CodePackage**

The CodePackage element specifies the location (and version) of the service's code.

<CodePackage Name="C" Version="1.0">

The Name element is used to specify the name of the directory in the application package that contains the service's code. CodePackage also has the version attribute. This can be used to specify the version of the code--and can also potentially be used to upgrade the service's code by using Service Fabric's application lifecycle management infrastructure.

**Entrypoint**

<EntryPoint>

<ExeHost>

<Program>HostedWebServer.exe</Program>

<WorkingFolder>CodePackage</WorkingFolder>

</ExeHost>

</EntryPoint>

The Entrypoint element in the service manifest file is used to specify how to launch the service. The ExeHost element specifies the executable (and arguments) that should be used to launch the service. You will be modifying the entry point for our package to pass arguments, and send console output to a log. You will do this in a later step. For now, here are the various options you can set in this element:

* Program specifies the name of the executable that should be executed to start the service.
* Arguments specifies the arguments that should be passed to the executable. It can be a list of parameters with arguments.
* WorkingFolder specifies the working directory for the process that is going to be started. You can specify three values:
  + CodeBase specifies that the working directory is going to be set to the code directory in the application package.
  + CodePackage specifies that the working directory is going to be set to the root of the application package (MyServicePkg).
  + Work specifies that files are placed in a sub-directory named work.
* WorkingDirectory is useful to set the correct working directory so that relative paths can be used by either the application or initialization scripts.
* ConsoleRedirection can be used to redirect console output (both stdout and stderr) to a working directory so they can be used to verify that there are no errors during the setup or execution of the application in the Service Fabric cluster.

**Endpoints**

<Endpoints>

<Endpoint Name="HostedWebServerTypeEndpoint" Protocol="http" Type="Input" />

</Endpoints>

The Endpoint element specifies the endpoints that the application can listen on. What you see above is the default endpoint, but in the next steps you will add a new endpoint for the Node.js application listens on port 8096. The type ‘input’ specifies that this port is open to the outside world.

1. Modify the **ServiceManifest.xml** to run our executable

The sample executable is a simple web service console app that listens on a port number. The port number is passed in as a parameter upon startup, and the executable reflects a response back the HTTP request that is sent to it. It is a modified version of [this](http://bartdesmet.net/blogs/bart/archive/2007/02/22/httplistener-for-dummies-a-simple-http-request-reflector.aspx) sample app. Because the app requires a port as a parameter, you need to configure the ExeHost element within the EntryPoint element to provide an argument to the app on startup.

Add the following Arguments element:

|  |
| --- |
| <EntryPoint>  <ExeHost>  <Program>HostedWebServer.exe</Program>  **<Arguments>8096</Arguments>**  <WorkingFolder>CodePackage</WorkingFolder>  </ExeHost>  </EntryPoint> |

The code addition above will take care of modifying the endpoint used by our application, but you also need to tell our Service Fabric host to listen on the same endpoint. You do this by **adding** the Endpoint element nested within the Resources element as follow:

|  |
| --- |
| <Resources>  <Endpoints>  <Endpoint Name="HostedWebServerTypeEndpoint" Protocol="http" Type="Input" />  **<Endpoint Name="WebServerTypeEndpoint" Protocol="http" Port="8096" Type="Input" />**  </Endpoints>  </Resources> |

**Logging**

It is very useful to be able to see console logs to find out if the application and configuration scripts show any errors. Console redirection can be configured in the ServiceManifest.xml file using the ConsoleRedirection element.

1. Insert the following to see output from the console in log files.

|  |
| --- |
| <EntryPoint>  <ExeHost>  <Program>HostedWebServer.exe</Program>  <Arguments>8096</Arguments>  <WorkingFolder>CodePackage</WorkingFolder>  **<ConsoleRedirection FileRetentionCount="5" FileMaxSizeInKb="2048"/>**  </ExeHost>  </EntryPoint> |

* FileRetentionCount determines how many files are saved in the working directory. A value of 5, for instance, means that the log files for the previous five executions are stored in the working directory.
* FileMaxSizeInKb specifies the max size of the log files.

Log files are saved in one of the service's working directories. Once you have deployed your application to the cluster, to determine where the files are located, you need to use the Service Fabric Explorer to determine which node that the service is running on and which working directory is being used. The working directory on OneBox will conform to the following pattern:

<Drive>:\SfDevCluster\Data\\_App\Node.#\AppType\_Version\log

You can also find this location in the Service Fabric Explorer. First find the node(s) your app is running on by drilling down on the Applications section, then go to the app instance in that node. The path is under “Disk Location” in the ESSENTIALS tab.

## Task 2 – Setting up PowerShell to run the deployment script

1. Now that you have configured our app, you need to setup our PowerShell environment to run the script to install the application. Start by opening PowerShell ISE (Integrated Script Environment)
2. Remove security requirements (note that you would want to sign PowerShell scripts in production):

Set-ExecutionPolicy unrestricted

1. Change PowerShell’s root directory to the directory containing your package. In the command line PowerShell window, simply type “cd <your\_path>” , ie, where your ‘pkg’ folder is.

Note: This directory location needs to be at the same level as the package directory you used in Task 1.

1. You can now run the script to install our application. Paste the following PowerShell script into the script pane area in ISE:

<#Deploy SimpleWebServer.exe locally#>

Connect-ServiceFabricCluster localhost:19000

Write-Host 'Copying application package...'

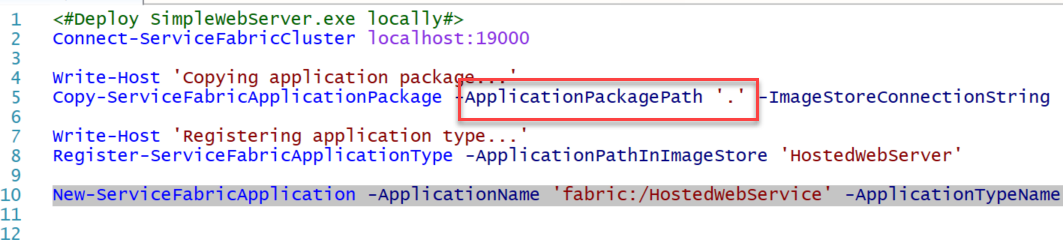
Copy-ServiceFabricApplicationPackage -ApplicationPackagePath '.\<your package directory>' -ImageStoreConnectionString 'file:C:\SfDevCluster\Data\ImageStoreShare' -ApplicationPackagePathInImageStore 'HostedWebServer'

Write-Host 'Registering application type...'

Register-ServiceFabricApplicationType -ApplicationPathInImageStore 'HostedWebServer'

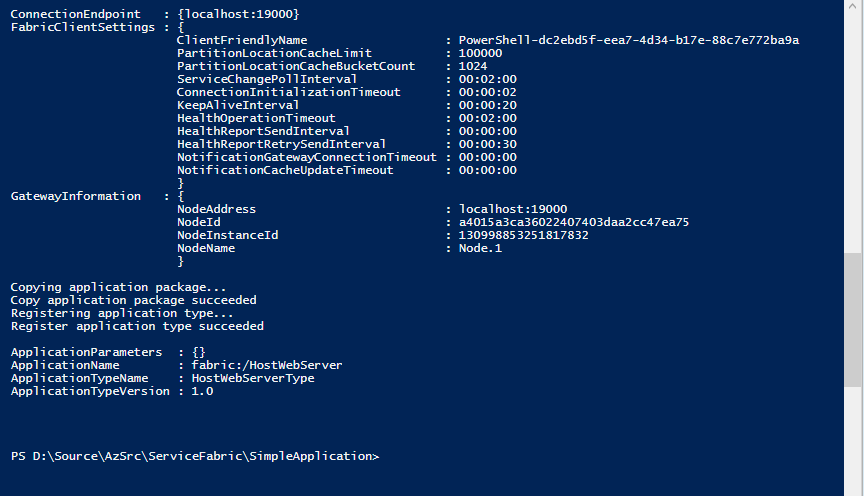
New-ServiceFabricApplication -ApplicationName 'fabric:/HostedWebService' -ApplicationTypeName 'HostedWebServerType' -ApplicationTypeVersion 1.0

You will need to modify the folders referenced in the script for local image store if necessary. If you are using the directory structure as given in the lab exercises, the –ApplicationPackagePath will be ‘.’.



1. Select the script (all lines) and press **F8** or right click on the script and choose “run selection”

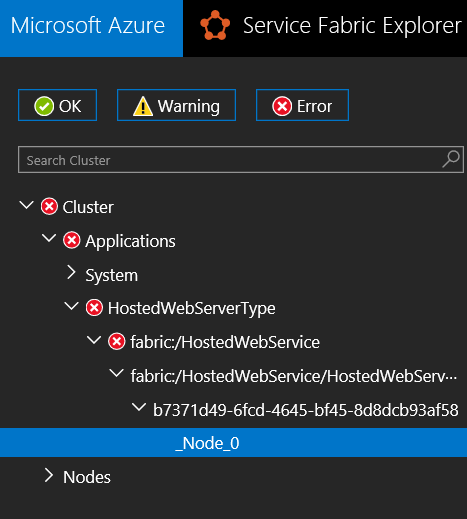
You should see output similar to the following:



1. Connect to the service in a browser “<http://localhost:8096/>”

You will see and error saying that the “page can’t be displayed”. This is an expected outcome.

1. To learn how to diagnose problems that you may have, let’s examine the error logs to see what is going on. First check in Service Fabric Explorer (<http://localhost:19080/Explorer>) to see what node you are running on:



1. Then open the directory containing the logs for that node. It will look something like this **“C:\SfDevCluster\Data\\_App\Node.0\HostedWebServerType\_App1\log”**.
2. Open a .err log containing data (file size greater than 0).

The HttpListener application requires admin permissions.

## Task 3 – Correcting the endpoint security issue

You have these issues because in the deployed application, there is an HttpListener C# class which uses Http.sys under the covers. This kernel mode component puts ACL security restrictions on its endpoints, so the endpoints can only be accessed by an Administrator by default. One way to deal with this issue would be to configure the endpoints to remove the security restrictions. You can reconfigure the endpoint security using a script provided in your lab tools folder called **SetupEndpoints.cmd**. This script will re-ACL the endpoints for ports 8096 – 8099, but you could configure it for any port or range. You need to run this script as an administrator before the application is deployed.

**Running the script**

1. To get the script installed on the cluster nodes, you can configure a new exe host which runs at setup time, the <SetupEntrypoint> of the **ServiceManifest.xml**. The code just below this is an example of what our updated ServiceManifest.xml will look like.

**SetupEntrypoint**

<SetupEntryPoint>

<ExeHost>

<Program>scripts\HostedWebServiceSetup.cmd</Program>

</ExeHost>

</SetupEntryPoint>

The <SetupEntrypoint> element is used to specify any executable or batch file that should be executed before the service's code is launched. It is an optional element, so it does not need to be included if there is no initialization/setup required. The <SetupEntryPoint> element is executed every time the service is restarted.

There is only one <SetupEntryPoint>, so setup/config scripts need to be bundled in a single batch file if the application's setup/config requires multiple scripts. Like the Entrypoint element, <SetupEntryPoint> can execute any type of file--executable files, batch files, and PowerShell cmdlets. In the example above, the <SetupEntryPoint> is based on a batch file **HostedWebServer.cmd** that is located in the *scripts* subdirectory of the CodePackage directory (assuming the WorkingDirectory element is set to CodePackage).

1. In the **.\pkg\HostedWebServer\C** folder, there is a command file named **SetupEndpoints.cmd** that will be used to setup the ACL access.

Modify your ServiceManifest.xml file as follows to define the setup entry point.

|  |
| --- |
| <CodePackage Name="C" Version="1.0">  **<SetupEntryPoint>**  **<ExeHost>**  **<Program>SetupEndpoints.cmd</Program>**  **<ConsoleRedirection FileRetentionCount="5" FileMaxSizeInKb="2048"/>**  **</ExeHost>**  **</SetupEntryPoint>**  <EntryPoint>  <ExeHost> |

Notice that you have configured the startup script to redirect console output to logs just as you did in your main application.

1. Next, you need to ensure that our setup script runs with administrator privileges, but you don’t necessarily need the whole application to run in this mode.

Following the advice in this [article](https://azure.microsoft.com/en-us/documentation/articles/service-fabric-application-runas-security/), you can modify the app to run to run just the startup script with admin permissions.

User permissions are controlled and defined at the app level using the application manifest. Open the **ApplicationManifest.xml** file and add the following to the bottom of the file to define an admin user on the hosting “machine”:

|  |
| --- |
| </DefaultServices>  **<Principals>**  **<Users>**  **<User Name="SetupAdminUser">**  **<MemberOf>**  **<SystemGroup Name="Administrators" />**  **</MemberOf>**  **</User>**  **</Users>**  **</Principals>**  </ApplicationManifest> |

1. You need to assign this user to run our setup script. To do this, modify the ServiceManifestImport section of the **ApplicationManifest.xml** file:

|  |
| --- |
| <ServiceManifestImport>  <ServiceManifestRef ServiceManifestName="HostedWebServer"  ServiceManifestVersion="1.0" />  **<ConfigOverrides />**  **<Policies>**  **<RunAsPolicy CodePackageRef="C" UserRef="SetupAdminUser"**  **EntryPointType="Setup" />**  **</Policies>**  </ServiceManifestImport> |

Notice that the UserRef attribute refers to the user you defined in the Principals section. Note also that the CodePackageRef needs to match the CodePackage Name in ServiceManifest.xml.

1. Now you can test the app again. Before redeploying, you need to delete and unregister the old application. You can do that with the following PowerShell lines which can be pasted into the ISE, selected and run, just as you ran the main script in the previous section.

Remove-ServiceFabricApplication fabric:/HostedWebService

Unregister-ServiceFabricApplicationType HostedWebServerType 1.0

1. After you run this, you will be prompted twice for confirmation. Go ahead and hit “yes” for both.
2. Next run the main installation script as before.

## Task 4 – Exploring our applications behavior

1. You can now go to <http://localhost:8096/> to test your application. If all went well, you should see the http request reflected back to you.
2. In Server Fabric Explorer (<http://localhost:19080/Explorer/index.html#/tab/essentials>), find the node running the exe and disable it. Refresh the browser continuously for about 10 seconds (until you see the node switch to disabled).

Notice that the browser continues to display the machine name without error.

1. Refresh the Applications branch of Service Explorer and check to see that the service has seamlessly transitioned nodes. This may take some time, and there may be delay while the service is transitioned.
2. Change the instance count of the application in **ApplicationManifest.xml.**

|  |
| --- |
| <Service Name="HostedWebServerService">  <StatelessService ServiceTypeName="HostedWebServer" **InstanceCount="4"**>  <SingletonPartition /> |

Note that changing this value to -1 would cause an instance to be spun up on every node.

1. Delete and reload the application instance as before
2. Return to Visual Studio or the Service Fabric Explorer - What nodes is the app running on now?
3. Make a few more requests to <http://localhost:8096/>
4. Check the log files for each machine. Notice that three of the machines have error logs stating:

|  |
| --- |
| Unhandled Exception: System.Net.HttpListenerException: Failed to listen on prefix 'http://+:8096/' because it conflicts with an existing registration on the machine.  at System.Net.HttpListener.AddAllPrefixes()  at System.Net.HttpListener.Start()  at HostedWebServer.Program.Main(String[] args) |

This is because you are running in a OneBox environment and all instances share the same physical host. If you were running on Azure, all instances would be running on separate physical hosts.

1. When you have identified the instance servicing the request, go to Service Fabric Explorer and disable that instance. You can determine the instance servicing the request by looking in the log files and finding the instance with no error logs.
2. Try connecting to <http://localhost:8096/> again. Notice that another machine in the cluster has seamlessly picked up the workload, much faster this time.

On OneBox, the effect of distributing this application has given us better redundancy, but not scalability. All of the machines are in an error state (port in use) except the machine which is servicing requests.

In Azure, you have the option of putting the collection of machines behind the Azure Load Balancer. This would give us the redundancy, plus scale out advantages for our stateless service.

If you were creating a stateless hosted exe service that was wholly consumed by other services within our application, you could query the Service Fabric for the existing instances of the exe service, and implement our own load balancing logic.

1. Now disable another machine in the cluster, leaving only three enabled.
2. Check the state of your application in Service Fabric Explorer. What is the state of the application?