**Group Title**

**Topic N: Deploying Windows Azure Applications**

[](http://msdn.microsoft.com/en-us/rampup/bb352986.aspx)

Ascend to new heights in your career

[**http://msdn.microsoft.com/rampup**](http://msdn.microsoft.com/rampup)

**Contents**

[Introduction 3](#_Toc244760813)

[Preparing your application for deployment 3](#_Toc244760814)

[Packaging your application for deployment 4](#_Toc244760815)

[Using the Windows Azure Tools for Visual Studio 4](#_Toc244760816)

[Using the SDK Tools 5](#_Toc244760817)

[Deploying your application 5](#_Toc244760818)

[Additional Resources 8](#_Toc244760819)

## Introduction

Deploying a service to the Windows Azure hosting environment is a relatively straightforward process. The only real requirements are to supply a **package** and a **service configuration file**. The package consists of your service’s binaries and any configuration files and/or resources it needs such as images or localization assemblies, while the service configuration contains the configuration details necessary for your application to access data and communications endpoints. Windows Azure provides several tools to simplify this process, and this article provides a brief overview of how a developer may use these tools to deploy their service to the Windows Azure Hosting Fabric.

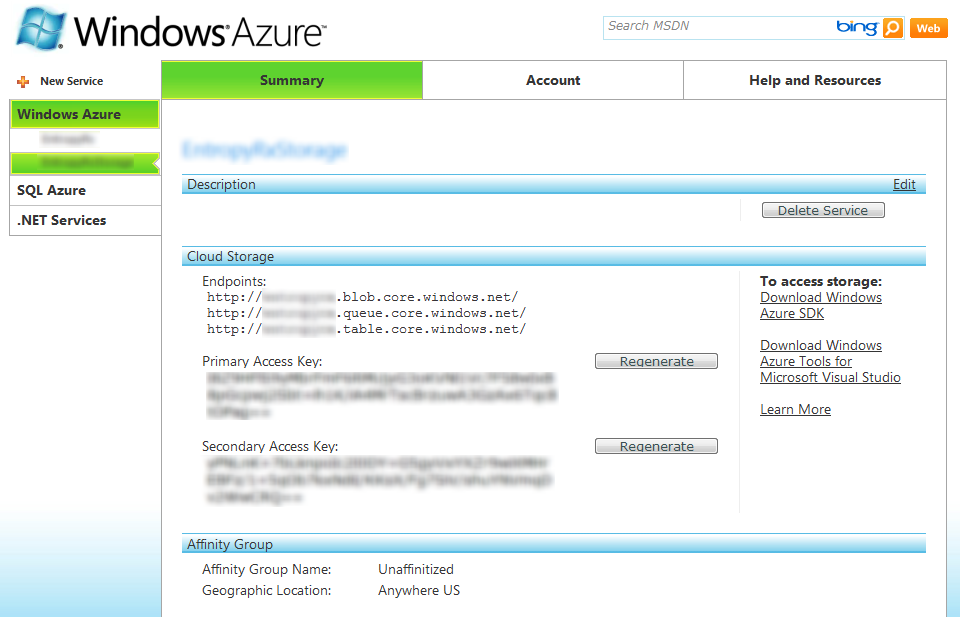
## Preparing your application for deployment

If your application consumes external services such as Web Services or databases, you may have placed settings specific to your development environment in your application’s configuration files. In such cases, it will be necessary to ensure that your application’s settings are updated so that it functions properly after deployment to the cloud. To facilitate the differences in configuration requirements for development, staging and production, Windows Azure provides the **Service Configuration File**. The default name used for this file by tools such as Windows Azure Tools for Visual Studio is “**ServiceConfiguration.csfg**,” but unlike configuration files such as ASP.NET’s web.config, the name of the file has no real significance.

By abstracting away the configuration from the actual runtime components, the service developer can deploy the same unmodified runtime package in several hosting environments by simply supplying different configuration files. While using this approach does require that your service is written to rely upon the configuration services and APIs provided by Windows Azure, this reliance upon configuration helps to reduce the differences between hosting environments (Development, Staging, Production) to just the differences in their configuration and therefore simplifies the troubleshooting process if any issues do arise.

Although there may be many ways in which a given service’s configuration may change between Development and the runtime hosting fabric, one of the notable differences is the difference between the **Azure Development Storage** **system** and the runtime **Azure Storage service**. In the case of Azure Storage, the mechanism for locating the different storage services (table, blob, and queue) differs: on a developer’s machine the locations are determined using different ports on the development machine, while the runtime services are located using domain name addresses.

If your application is using Azure Storage services, you will need to update your application’s settings in order to access your Azure Storage account rather than your local development storage by modifying the access key and endpoint URLs used to access storage services to match those provided for your Azure Storage account. You can find this information on the summary page for your Azure Storage account as seen below.



If you are using the StorageClient sample from the Windows Azure SDK for your Azure Storage data access, you will need to update the configuration settings used by StorageClient in one of your application’s configuration files (ServiceConfiguration.cscfg or web.config). Set the AccountName setting to the Service/Domain Name you entered; this should be a lower-case string containing the first part of your endpoint’s domain name (e.g., if your table storage endpoint is <http://storagedemo.table.core.windows.net>, it would be ‘storagedemo’), and set the AccountSharedKey setting to be the Primary Access Key as seen on the Azure Storage summary page. An example is shown below:

|  |  |  |
| --- | --- | --- |
| **Storage Service** | **Development Storage** | **Using the Windows Azure storage with the StorageClient sample** |
| **Blob Storage** | <http://127.0.0.1:10000/> | [http://blob.core.windows.net](http://blob.core.windows.net/) |
| **Queue** | <http://127.0.0.1:10001/> | [http://queue.core.windows.net](http://queue.core.windows.net/) |
| **Table Storage** | <http://127.0.0.1:10002/> | [http://table.core.windows.net](http://table.core.windows.net/) |

<?xml version="1.0"?>

<ServiceConfiguration serviceName="SampleApp" xmlns="http://schemas.microsoft.com/ServiceHosting/2008/10/ServiceConfiguration">

<Role name="SampleApp\_WebRole">

<Instances count="1" />

<ConfigurationSettings>

<Setting name="AccountName" value="storagedemo"/>

<Setting name="AccountSharedKey" value="<<account key from the portal>>"/>

<Setting name="TableStorageEndpoint" value="http://table.core.windows.net"/>

<Setting name="BlobStorageEndpoint" value="http://blob.core.windows.net"/>

<Setting name="QueueStorageEndpoint" value="http://queue.core.windows.net"/>

</ConfigurationSettings>

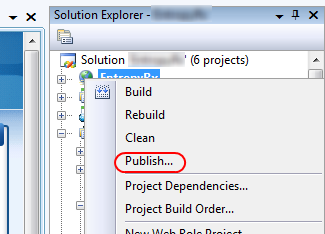
</Role>

</ServiceConfiguration>

## Packaging your application for deployment

### Using the Windows Azure Tools for Visual Studio

The Windows Azure Tools for Visual Studio simplify the process of service development by providing a framework in the Visual Studio development environment for automating most of the steps necessary for developing, testing and deploying your Azure service. After you have reached a state at which you are ready to deploy your service to the cloud, you may then start the service publication process by either clicking the “Publish” item in your Service project’s context menu, or by choosing the “Publish <<Service Name>>” menu item under the Build menu.



### 

### Using the SDK Tools

The **CSPack** command-line tool is provided by the Windows Azure SDK, and is used to package a service for deployment. Based upon the parameters passed to the command line, CSPack can generate a package for deployment to the Development Fabric or to be deployed to the Windows Azure runtime fabric for deployment to the Windows Azure hosting environment via the developer portal.

To generate the package for deployment, compile your service and ensure that all resource files are copied to the folder that contains the compiled assembly. You can then run the CSPack utility to generate the package by supplying the name of your service’s Service Definition file (.csdef file). By default, the CSPack utility will use the names of the roles defined in the .csdef file to locate the folders that contain the binaries and resources, but this location may be manually supplied using the /role command-line switch. By default the package file is named <service-name>.cspkg, but you can specify a different name if you choose by supplying a value to the /out command-line switch.

## Deploying your application

Once you have built and packaged your application, you are ready to deploy it to the Windows Azure Hosting Fabric. To deploy the application, you will have to have created an account for the Azure Service Platform, obtained an access token for Windows Azure, and provisioned your application. These steps are described in detail in a separate article (see: “Setting Up Accounts”) and will not be covered in this article.

To deploy your service’s package, simply navigate to the Azure Project Management portal and select your Windows Azure project. You are then presented with the deployment and configuration interface for your staging and production environments. You may then upload your service’s package and configuration to the staging environment by clicking the Deploy button.

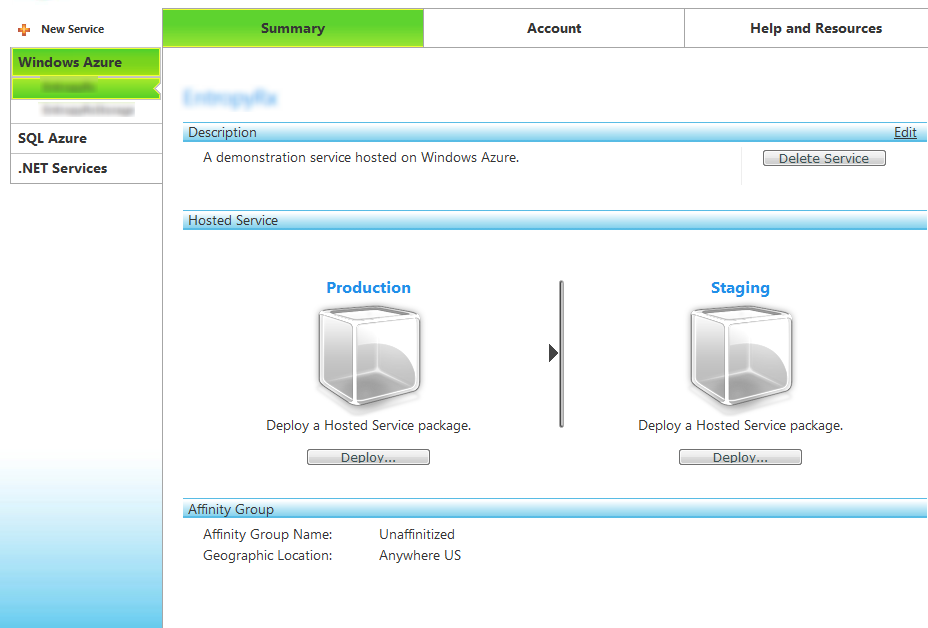


Figure : The Windows Azure Project Deployment Interface

Once you click the Deploy button, you are presented with a new interface for uploading your service’s package and configuration file for the target environment as well as a label for this deployment. The package and configuration file will come from the machine on which you generated the package and the location of these files will depend upon the tools and configuration parameters you use to generate your package. Once you have uploaded the package and configuration file and provided a label, you may then click the Deploy button to return to the deployment interface. Once your service has been allocated, you may then click the Run button to start your service. (Note: it may take some time for the service to reach the “Allocated” state).

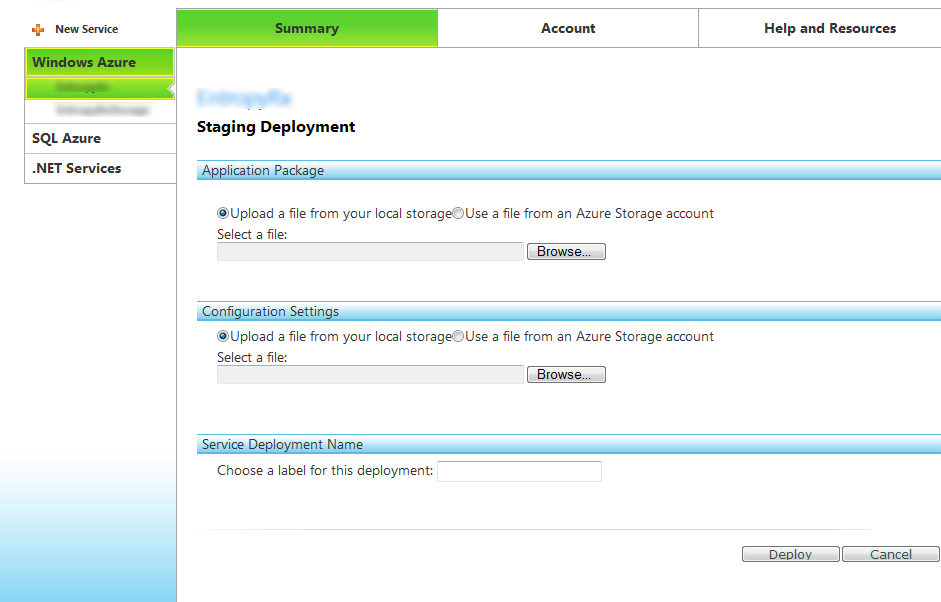


Figure : The interface for uploading your package

Once you have deployed your application to the staging environment, you are then provided with the ability to test the functionality of your service while it is hosted in the Azure Hosting Fabric but prior to a full production deployment. After you have deployed the service, the Azure platform generates a URL containing a unique domain name registration for your application that can then be used to access the staging environment from the Internet. Be aware that it may take some time for the domain name registration to propagate.

After you have tested your service and have verified that it is working properly, you may then swap the Staging and Production environments from the management interface by clicking the blue “swap environments” button between the Production and Staging areas of the management interface. When you perform this step, the package and configuration from the Staging environment is moved to the Production environment and the previous production environment is then moved to the Staging environment (it should be noted that any configuration changes necessary to run under the Production environment should be made prior to this swap). If you determine that the application is not functioning properly under the Production environment, you can restore the previous Production package and configuration by clicking the swap button again. If you are satisfied that the Production environment, you may then resume your service development activities. In the future, you will be able to replace the contents of the Staging environment with a new package and configuration file by clicking the Upgrade button, which will then present you with the deployment interface.

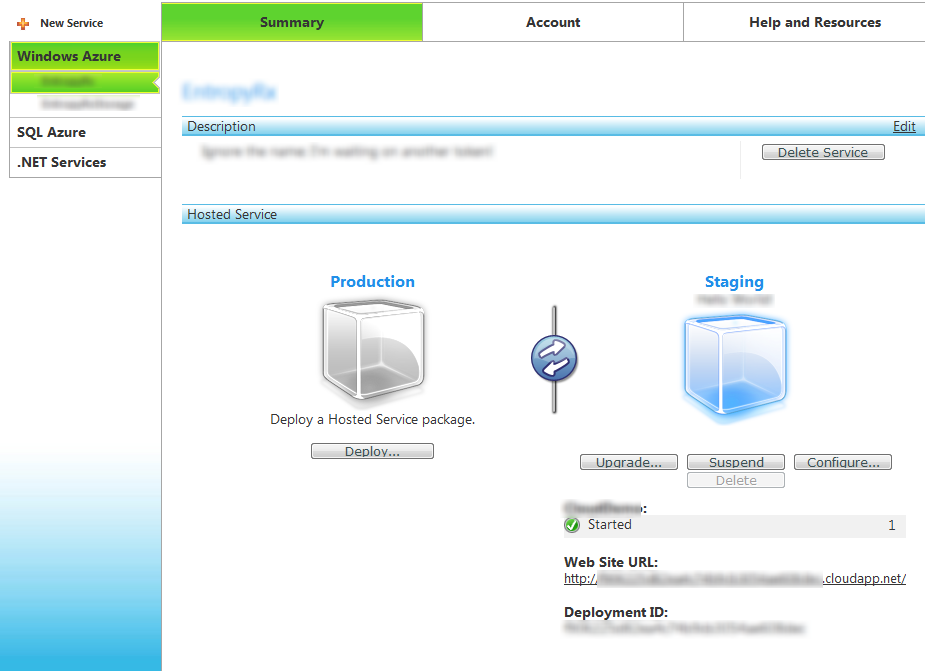


Figure : A service deployed to the Staging environment

## 

## Additional Resources

Additional resources for learning about deploying Windows Azure applications are listed below:

|  |  |
| --- | --- |
| Name | Location |
| Windows Azure Developers Portal | <http://www.microsoft.com/azure/windowsazurefordevelopers/default.aspx> |
| Windows Azure Tools for Visual Studio | <http://go.microsoft.com/fwlink/?LinkID=128752> |
| Windows Azure SDK | <http://go.microsoft.com/fwlink/?LinkId=130232> |
| CSPack reference | <http://msdn.microsoft.com/en-us/library/dd179470.aspx> |