# Running Orleans on Windows Azure

The Orleans server runtime (silos) and application components (grains) can be run on either Windows Server or Windows Azure.  
A different hosting process is used for the different runtime environments.   
This document describes the steps required to set up hosting of Orleans silos and client applications in Azure.

In Azure, one or more worker roles will be used to host the Orleans silos containing the application code running as Orleans grains, and an Azure web role will act as the presentation layer for the application and client to the application grains running in the Orleans silos. Here is an overview of a typical setup for running Orleans applications in Azure.



The [SDK-ROOT]\Samples\AzureWebSample\**AzureWebSample.sln** sample app in the Orleans SDK provides a worked example of how to run a web application with supporting Orleans silo cluster backend in an Azure hosted service, and the details are described below.

## Pre-requisites

The following pre-requisite products should be installed on your development machine for running Orleans in Azure:

1. Visual Studio 2010
2. Windows Azure Tools for Visual Studio 2010 v1.6 (includes Windows Azure SDK)  
   Download from <http://www.microsoft.com/windowsazure/getstarted/>   
   or install through the Web Platform Installer tool <http://www.microsoft.com/web/>
3. If applicable, hot fix KB983301 will be required in order to enable Windows Azure IntelliTrace on 32-bit OS.  
   <http://archive.msdn.microsoft.com/KB983301>

For more info on installing and working with Windows Azure in general, see the Microsoft Developer Network (MSDN) web site: <http://msdn.microsoft.com/en-us/windowsazure/>

## Write Your Application Business Logic as Orleans Grains

The same grain interfaces and implementation can run on both Windows Server and Windows Azure, so no special considerations are needed in order to be able to run your application in a Windows Azure hosting environment.

See the **Orleans Programmer’s Guide** document and sample applications in the Orleans SDK for more info on writing Orleans grains.

## Running Orleans Silos as Azure Worker Role

### Create Azure Worker Role for Orleans Silos

Using the Azure Tools for Visual Studio, create a normal Azure worker role project called ‘**OrleansAzureSilos**’[[1]](#footnote-1)

Create a new project/solution:

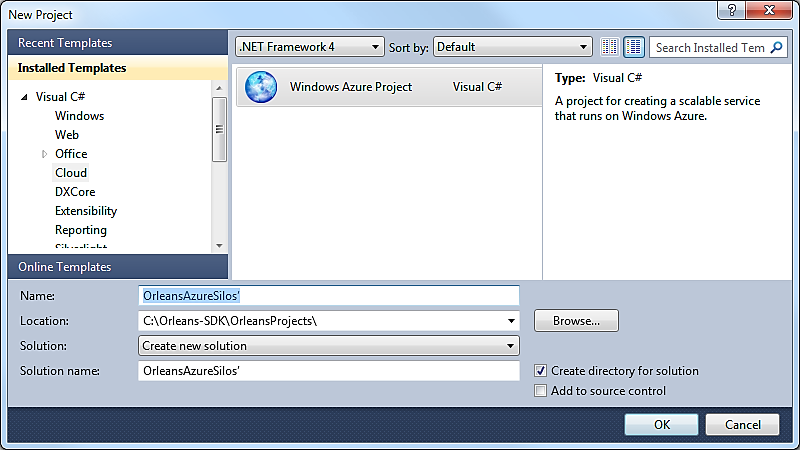


Figure 1- Create new Azure solution.

Add a worker role to the solution:

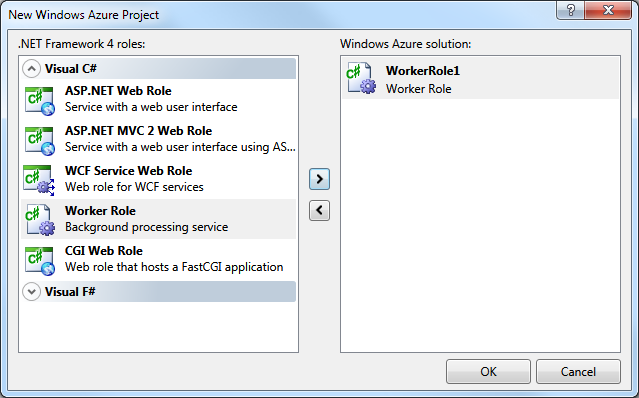


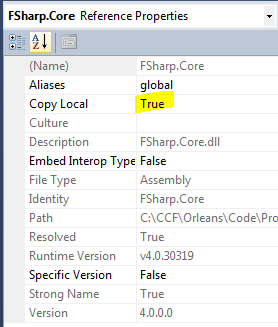
Figure 2 - Inserting an Azure Worker Role

### Add project references for Orleans Silo binaries

Add references to the OrleansAzureSilos project for the required Orleans server library files. Copies of these files can be found in either [SDK-ROOT]\Samples\AzureWebSample\ServerBin or [SDK-ROOT]\Binaries\OrleansServer directories.

* Orleans.dll
* OrleansAzureUtils.dll
* OrleansRuntime.dll
* OrleansRuntimeGrainInterfaces.dll
* OrleansSiloHost.dll

NOTE: All of these references MUST have **Copy Local = True** settings to ensure the necessary library DLL’s get **copied** into the OrleansAzureSilos project output directory.



### Configure Azure Worker Role for Orleans Silos

The Worker Role initialization class is a normal Azure worker role - it needs to inherit from the usual Microsoft.WindowsAzure.ServiceRuntime.RoleEntryPoint base class.

The worker role initialization class needs to create an instance of Orleans.Host.Azure.OrleansAzureSilo class, and call the appropriate Start/Run/Stop functions in the appropriate places:

1. public class WorkerRole : RoleEntryPoint
2. {
3. Orleans.Host.Azure.OrleansAzureSilo silo;
4. public override bool OnStart() {
5. /\* *Do other silo initialization – for example: Azure diagnostics, etc* \*/
6. **silo** = new OrleansAzureSilo();
7. return silo.**Start**(
8. RoleEnvironment.DeploymentId, RoleEnvironment.CurrentRoleInstance);
9. }
10. public override void OnStop() { silo.**Stop**(); }
11. public override void Run() { silo.**Run**(); }
12. }

Then, in the **ServiceDefinition.csdef** file for this role, add some required configuration items used by the Orleans Azure hosting library to the WorkerRole configuration:

* Add a **ConfigurationSettings** declaration named ‘**DataConnectionString**’  
  This is the Azure storage location where Orleans Azure hosting library will place / look for its silo instance table.
* Add a **LocalStorage** declaration named ‘**LocalStoreDirectory**’  
  This is the directory that will be used for any local cache directories.
* Add an **InternalEndpoint** declaration for a TCP endpoint named ‘**OrleansSiloEndpoint**’
* Add an **InternalEndpoint** declaration for a TCP endpoint named ‘**OrleansProxyEndpoint**’

1. <ServiceDefinition ...>
2. <WorkerRole name="OrleansAzureSilos" ...>
3. ...
4. <ConfigurationSettings>
5. <Setting name="DataConnectionString" />
6. </ConfigurationSettings>
7. <LocalResources>
8. <LocalStorage name="LocalStoreDirectory" cleanOnRoleRecycle="false" />
9. </LocalResources>
10. <Endpoints>
11. <InternalEndpoint name="OrleansSiloEndpoint" protocol="tcp" port="11111" />
12. <InternalEndpoint name="OrleansProxyEndpoint" protocol="tcp" port="30000" />
13. </Endpoints>
14. ...
15. </WorkerRole>
16. ...
17. </ServiceDefinition>

In the **ServiceConfiguration.cscfg** file for this role, add some required configuration items used by the Orleans Azure hosting library:

* Add a **ConfigurationSettings** definition for ‘**DataConnectionString**’  
  This will be a normal Azure storage connection – either for the development storage emulator (only valid if running locally), or a full Azure storage account connection string for cloud-storage.  
  In general, this connection string is likely to use the same configuration values as the "Microsoft.WindowsAzure.Plugins.Diagnostics.ConnectionString" diagnostics connection string setting, but is not required to.  
  <http://msdn.microsoft.com/en-us/library/ee758697.aspx>

For example, to use local Developer Storage emulator (for local testing only)

1. <ConfigurationSettings>
2. <Setting name="DataConnectionString" value="UseDevelopmentStorage=true" />
3. </ConfigurationSettings>

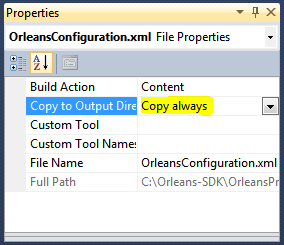
Or using an Azure cloud storage account:

1. <ConfigurationSettings>
2. <Setting name="DataConnectionString" value="DefaultEndpointsProtocol=https;AccountName=MyAccount;AccountKey=MyKey" />
3. </ConfigurationSettings>

### Add your Orleans silo config file to Azure Worker Role for Orleans Silos

Add an **OrleansConfiguration.xml** file into your OrleansAzureSilos worker role project, along with any supporting libraries those grains need. The networking configuration information in OrleansConfiguration.xml will be overridden by values from the Azure role environment / configuration.

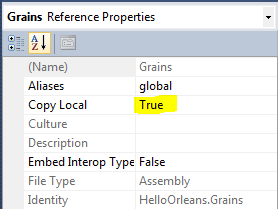
Note: You MUST ensure this config file get **copied** into the OrleansAzureSilos project output directory, to ensure they get picked up by the Azure packaging tools.



### Add your grain binaries to Azure Worker Role for Orleans Silos

Add the grain interfaces DLL, generated client proxy DLL and implementation classes DLL for the grains to he hosted in the Azure silo into the OrleansAzureSilos project, along with any supporting libraries those grains need.

Note: You MUST ensure that all the referenced binaries are **copied** into the OrleansAzureSilos project output directory, to ensure they get picked up by the Azure packaging tools.



## Running Orleans Client as Azure Web Role

The user interface / presentation layer for your application will usually run as a Web Role in Azure.

The **Orleans.Host.Azure.Client.OrleansAzureClient** utility class is the main mechanism for bootstrapping connection to the Orleans silo worker roles from an Azure Web Role. A few additional configuration steps are needed to make the **OrleansAzureClient** utility class work – see below for details.

Note: A current limitation of the OrleansAzureClient utility class is that it requires the Azure Web Role to run in Hosted Web Core (HWC) mode in order to automatically find the bootstrap Orleans client config files.   
This limitation should be removed in a later release of the Orleans SDK, or you can explicitly specify a location to find the Orleans config file in the call to OrleansAzureClient.Initialize if required.  
See this note from the Windows Azure team on the differences between HWC and full IIS hosting modes.  
<http://blogs.msdn.com/b/windowsazure/archive/2010/12/02/new-full-iis-capabilities-differences-from-hosted-web-core.aspx>

### Create Azure Web Role for Orleans Client

Using the Azure Tools for Visual Studio, create a normal Azure web role project.  
Any type of web role can be used as an Orleans client, and there are no specific naming requirements or conventions for this project.

Add a web role to the solution:

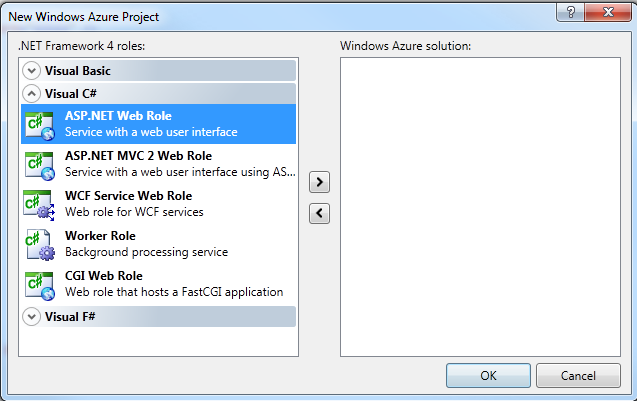


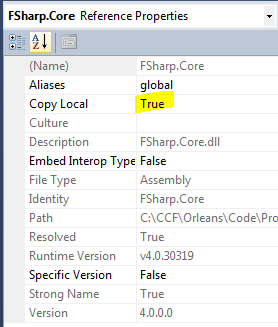
Figure 3 Inserting an Azure Web Role

### Add project references for Orleans Client binaries

Add references to the web role project for the required Orleans client library files. Copies of these files can be found in either [SDK-ROOT]\Samples\References or [SDK-ROOT]\Binaries\OrleansClient directories.

* Orleans.dll
* OrleansAzureUtils.dll
* OrleansRuntimeGrainInterfaces.dll

NOTE: All of these references MUST have **Copy Local = True** settings to ensure the necessary library DLL’s get **copied** into the web role project output directory.



### Configure Azure Web Role to be an Orleans Client

In the **ServiceDefinition.csdef** file for this web role, add some required configuration items used by the Orleans Azure hosting library:

* Add a **ConfigurationSettings** declaration named ‘**DataConnectionString**’  
  This is the Azure storage location where Orleans Azure hosting library will place / look for its silo instance table.
* In addition, a http/s InputEndpoint will also need to be declared, just as for any other Azure web role config.

1. <ServiceDefinition ...>
2. <WebRole name="MyWebRole" ...>
3. ...
4. <ConfigurationSettings>
5. <Setting name="DataConnectionString" />
6. </ConfigurationSettings>
7. <Endpoints>
8. <InputEndpoint name="Endpoint1" protocol="http" port="80" />
9. </Endpoints>
10. ...
11. </WebRole>
12. ...
13. </ServiceDefinition>

In order to ensure Hosted Web Core (HWC) hosting is used for this Web Role, **remove** the <Sites> element listed for this web role.

In the **ServiceConfiguration.cscfg** file for this role, add some required configuration items used by the Orleans Azure hosting library:

* Add a **ConfigurationSettings** definition for ‘**DataConnectionString**’.  
  This will be a normal Azure storage connection – either for the development storage emulator (if running locally), or a full Azure storage account connection string for cloud-storage.  
  This setting MUST match the DataConnectionString value used by the OrleansSiloWorker role in order for the client to discover and bootstrap the connection to the Orleans silos.  
  <http://msdn.microsoft.com/en-us/library/ee758697.aspx>

For example, to use local Developer Storage emulator (for local testing only)

1. <ServiceConfiguration ...>
2. <Role name="OrleansWebClient" ...>
3. ...
4. <ConfigurationSettings>
5. <Setting name="DataConnectionString" value="UseDevelopmentStorage=true" />
6. </ConfigurationSettings>
7. ...
8. </Role>
9. ...
10. </ServiceConfiguration>

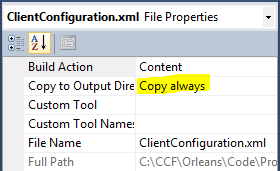
Or using an Azure cloud storage account:

1. <ServiceConfiguration ...>
2. <Role name="OrleansWebClient" ...>
3. ...
4. <ConfigurationSettings>
5. <Setting name="DataConnectionString" value="DefaultEndpointsProtocol=https;AccountName=MyAccount;AccountKey=MyKey" />
6. </ConfigurationSettings>
7. ...
8. </Role>
9. ...
10. </ServiceConfiguration>

### Add your Orleans client config file to Azure Web Role for Orleans Client

Add a **ClientConfiguration.xml** file into this project. The networking configuration information in ClientConfiguration.xml will be overridden by values from the Azure role environment / configuration.

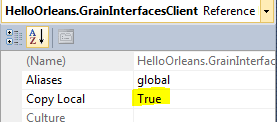
Note: You MUST ensure this config file get **copied** into the web role project output directory, to ensure they get picked up by the Azure packaging tools.



### Add your grain interface binaries to Azure Web Role for Orleans Client

Add the grain interfaces DLL and generated client proxy DLL for the application grains into this web role project.   
Access to the DLL containing the grain implementation classes should not be required by the client web role.

Note: You MUST ensure that all the referenced binaries for grain interfaces and the generated proxy / factory libraries are **copied** into the web role project output directory, to ensure they get picked up by the Azure packaging tools.



The grain implementation DLL’s should not be required by the client and so should not be referenced by the web role.

### Initialize Client Connection to Orleans Silos

It is recommended to bootstrap and initialize the client connection to the Orleans silo worker roles, to ensure a connection is set up before use – either in the Page\_Load method for each .aspx page, or in Global.asax initialization methods.

1. protected void Page\_Load(object sender, EventArgs e)
2. {
3. if (Page.IsPostBack)
4. {
5. if (!OrleansAzureClient.IsInitialized)
6. {
7. OrleansAzureClient.Initialize();
8. }
9. }
10. }

Repeated calls to OrleansAzureClient.Initialize() will return and do nothing if the Orleans client connection is already set up.

An additional variant of OrleansAzureClient.Initialize(System.IO.FileInfo) allows a base client config file location to be specified explicitly. The internal endpoint addresses of the Orleans silo nodes will still be determined dynamically from the Orleans Silo instance table each silo node registers with.

See the Orleans API docs for details of the various Initialize methods available.

## Deploying to Azure

The normal Azure deployment tools are used to deploy the application to Windows Azure – either into the local Azure Compute Emulator for local development / test (Press ‘F5’ to run), or into the Azure cloud hosting environment.  
<http://msdn.microsoft.com/en-us/library/gg433055.aspx>

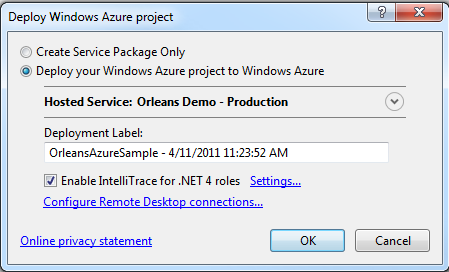


Figure 4 Deploying to Azure cloud from Visual Studio

For more info on working with Windows Azure, see the Windows Azure Developer section of the Microsoft Developer Network (MSDN) web site:  
<http://msdn.microsoft.com/en-us/windowsazure/>

1. If you don’t name the worker role for your silos with the recommended name, then any client web roles you use will need to specify the new name explicitly any place they initialize a connection to the silos. If you use the default name ‘OrleansAzureSilos’ then the OrleansAzureClient.Initialize() calls will be simplified by using the default name. [↑](#footnote-ref-1)