

Readme

Windows Azure Toolkit for Social Games

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | Version: | 1.0.0 | | Last updated: | 10/28/2011 | |  |

## Contents

[Overview 2](#_Toc307410667)

[Get Started 3](#_Toc307410668)

[Running the Setup 3](#_Toc307410669)

[Running the Samples 6](#_Toc307410670)

[The Solution 17](#_Toc307410671)

[Solution Structure 18](#_Toc307410672)

[Deployment 22](#_Toc307410673)

[Creating an AppFabric Access Control Service Namespace 23](#_Toc307410674)

[Preparing the Application for Deployment 30](#_Toc307410675)

[Using the Automated Deployment Script 38](#_Toc307410676)

Overview

Building a social game is a tough challenge. From the first iteration developers must plan for and deal with issues like high concurrency, real time interaction, and rapid growth. The Windows Azure Toolkit for Social Games provides you with the tools you need to ramp up your game development quickly on an architecture that will help you face future challenges. The toolkit leverages the power of Windows Azure and game development best practices to handle even the most demanding social games.

This version of the Toolkit consists of a generic game play service API, and two sample games that consume the operations exposed by the game service: “Tic-Tac-Toe” and “Four in a Row”. The game play service is responsible for handling the generic game operations like user authentication, joining multiple players to a game, and persisting the game live state.

This guide will walk you through the steps for running the sample game locally using the Windows Azure emulator and for deploying it to Windows Azure.

## Prerequisites

* 1. The following software is required to run this toolkit:
  + [Microsoft Visual Web Developer 2010 Express or Microsoft Visual Studio 2010](http://www.microsoft.com/express/Web/)
  + [Microsoft .NET Framework 4.0](http://go.microsoft.com/fwlink/?linkid=186916)
  + Internet Information Services 7 , with ASP.NET feature enabled
  + [Windows Azure SDK and Tools for Visual Studio (September 2011) version 1.5](http://go.microsoft.com/fwlink/?LinkID=128752)
  + [Microsoft SQL Server 2008 (Express edition or greater)](http://www.microsoft.com/express/Database/InstallOptions.aspx)
  + [Windows Identity Foundation SDK 4.0](http://www.microsoft.com/download/en/details.aspx?id=4451)
  + [Windows PowerShell 2.0](http://go.microsoft.com/fwlink/?LinkId=177670)
  + An HTML 5 capable browser – for example, Internet Explorer 9

## Learning more about the Windows Azure Platform

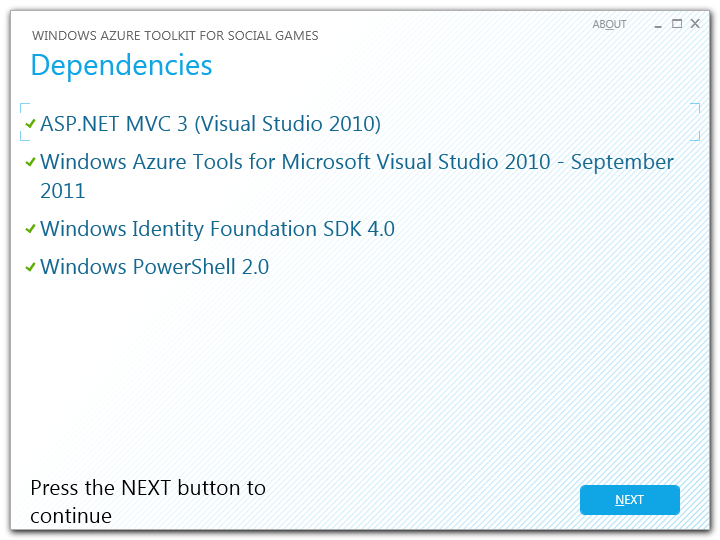
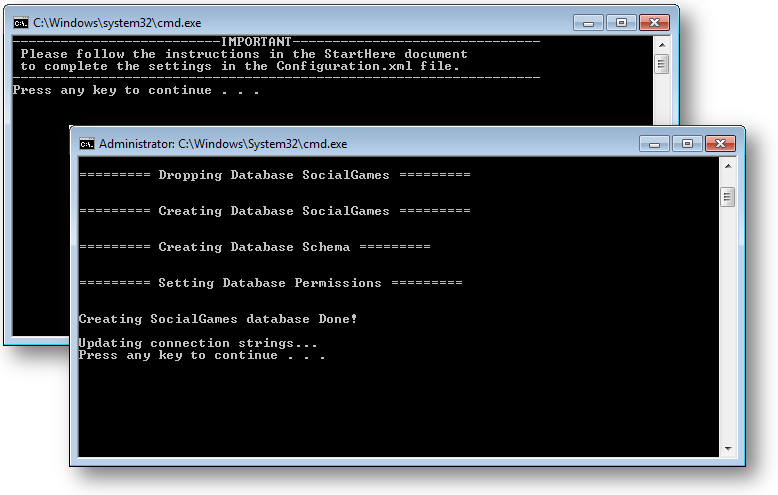
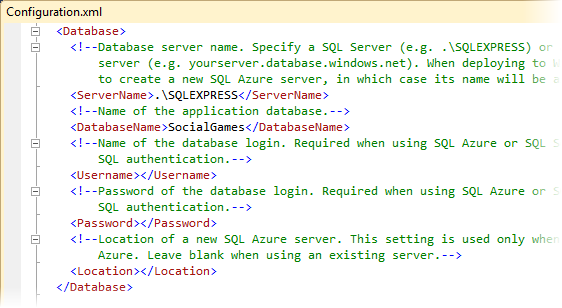
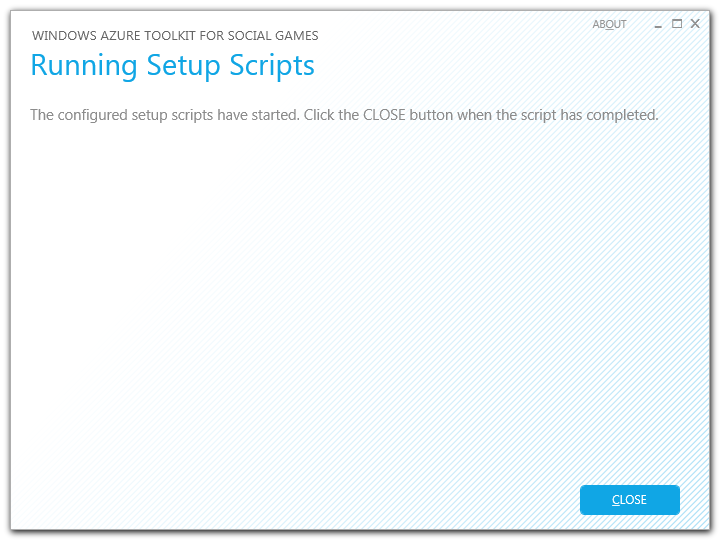
To learn more about the Windows Azure Platform and AppFabric, check these resources:

* + Complete the hands-on Labs in the Windows Azure Platform Training Course online on [MSDN](http://go.microsoft.com/fwlink/?LinkID=207018).
  + Learn how to build applications with the Windows Azure Platform Training Kit you can [Download Here](http://go.microsoft.com/fwlink/?LinkID=130354).

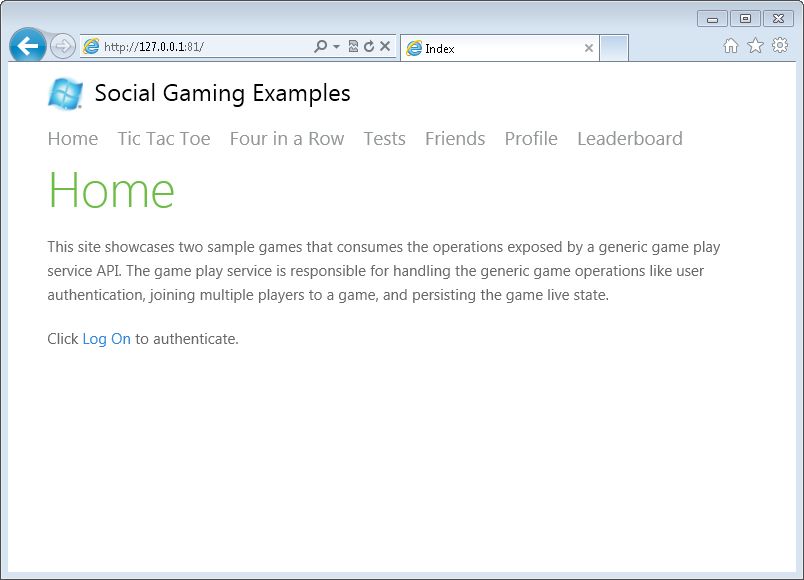
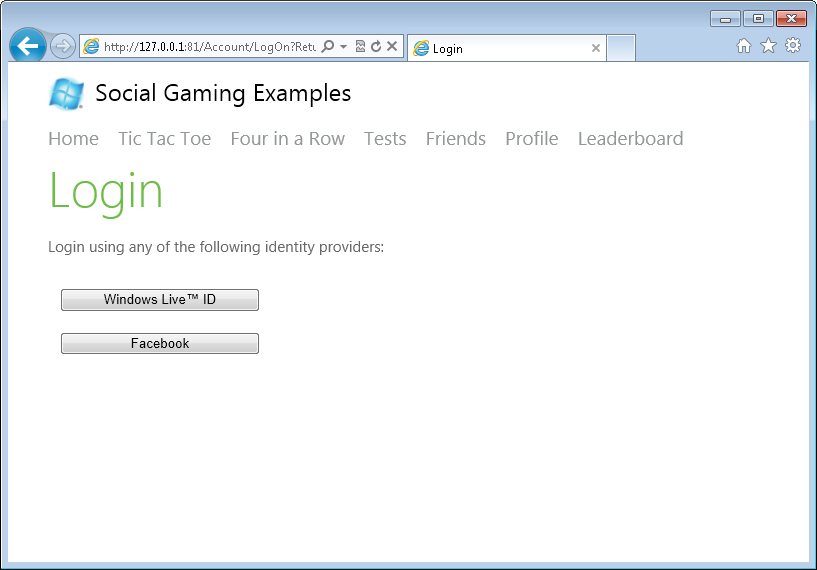
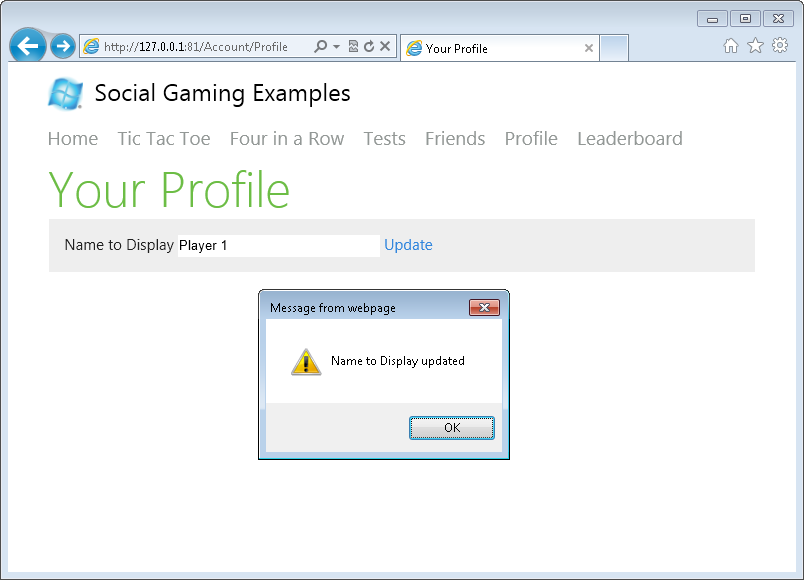
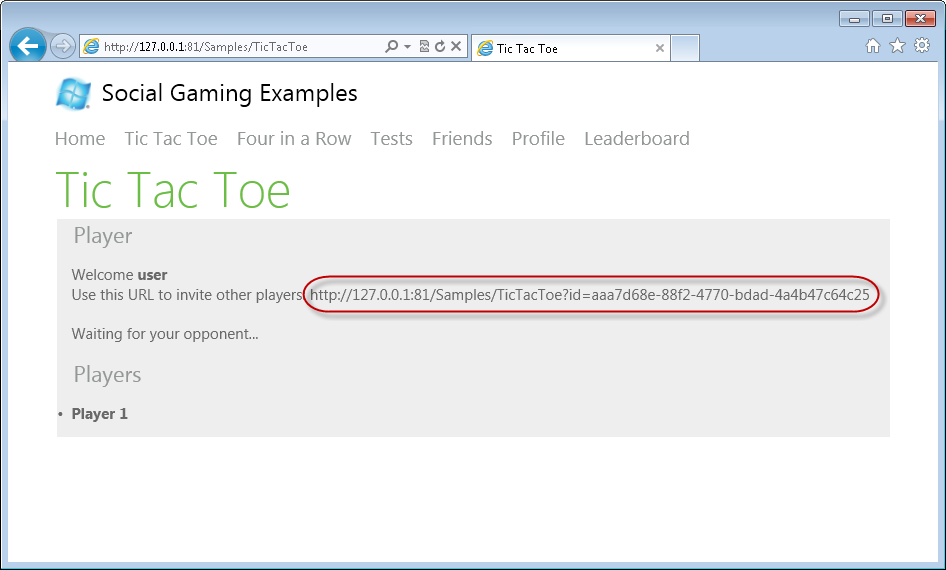
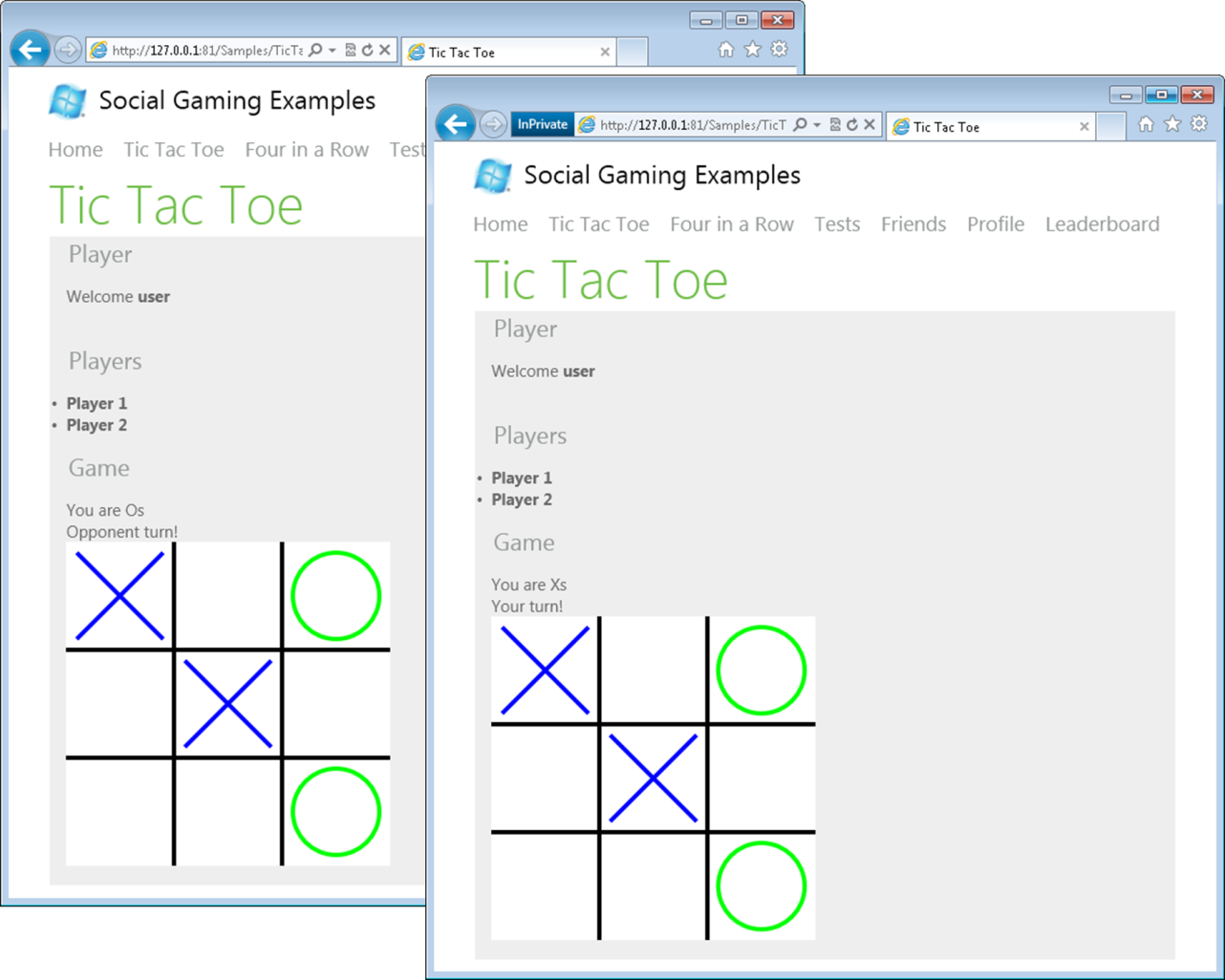
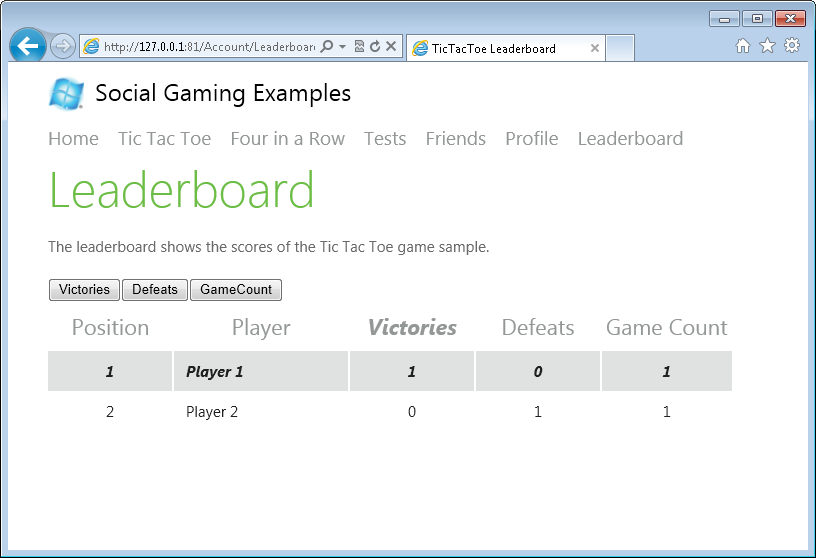
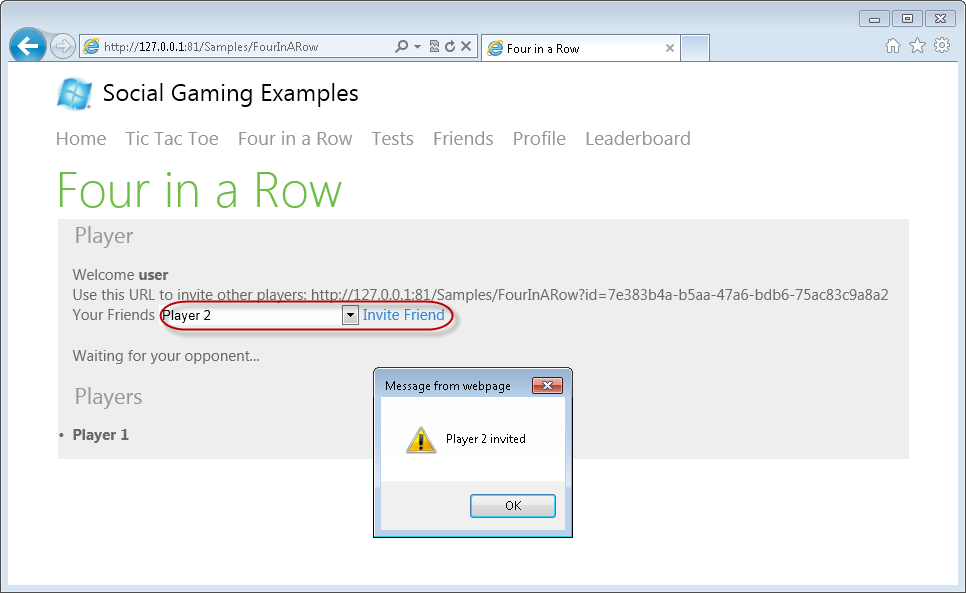
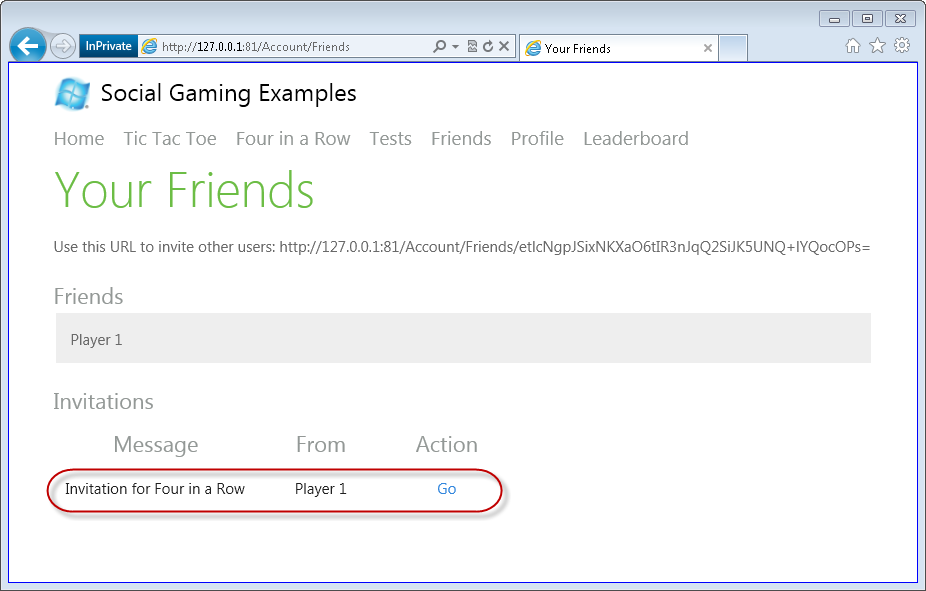
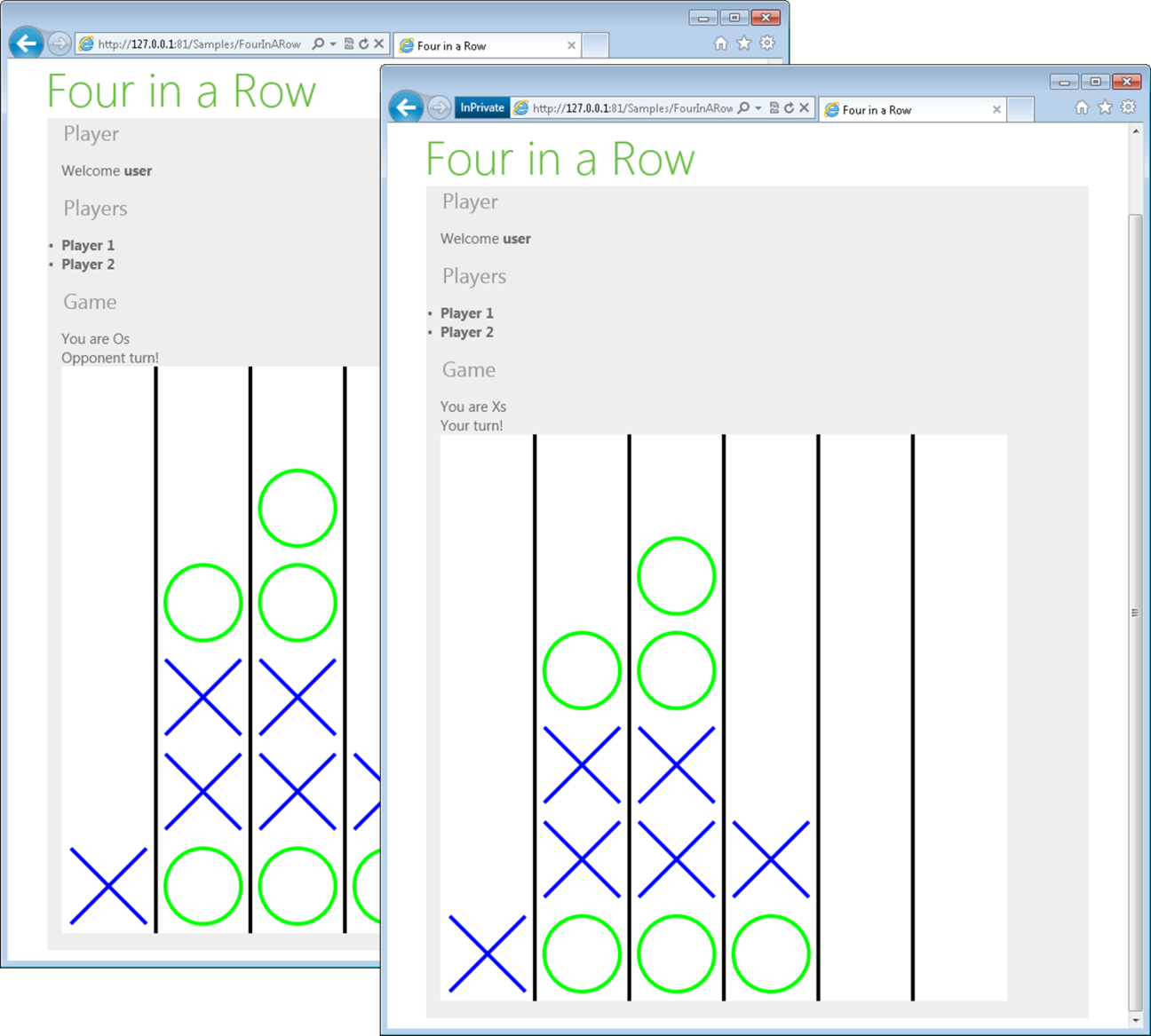
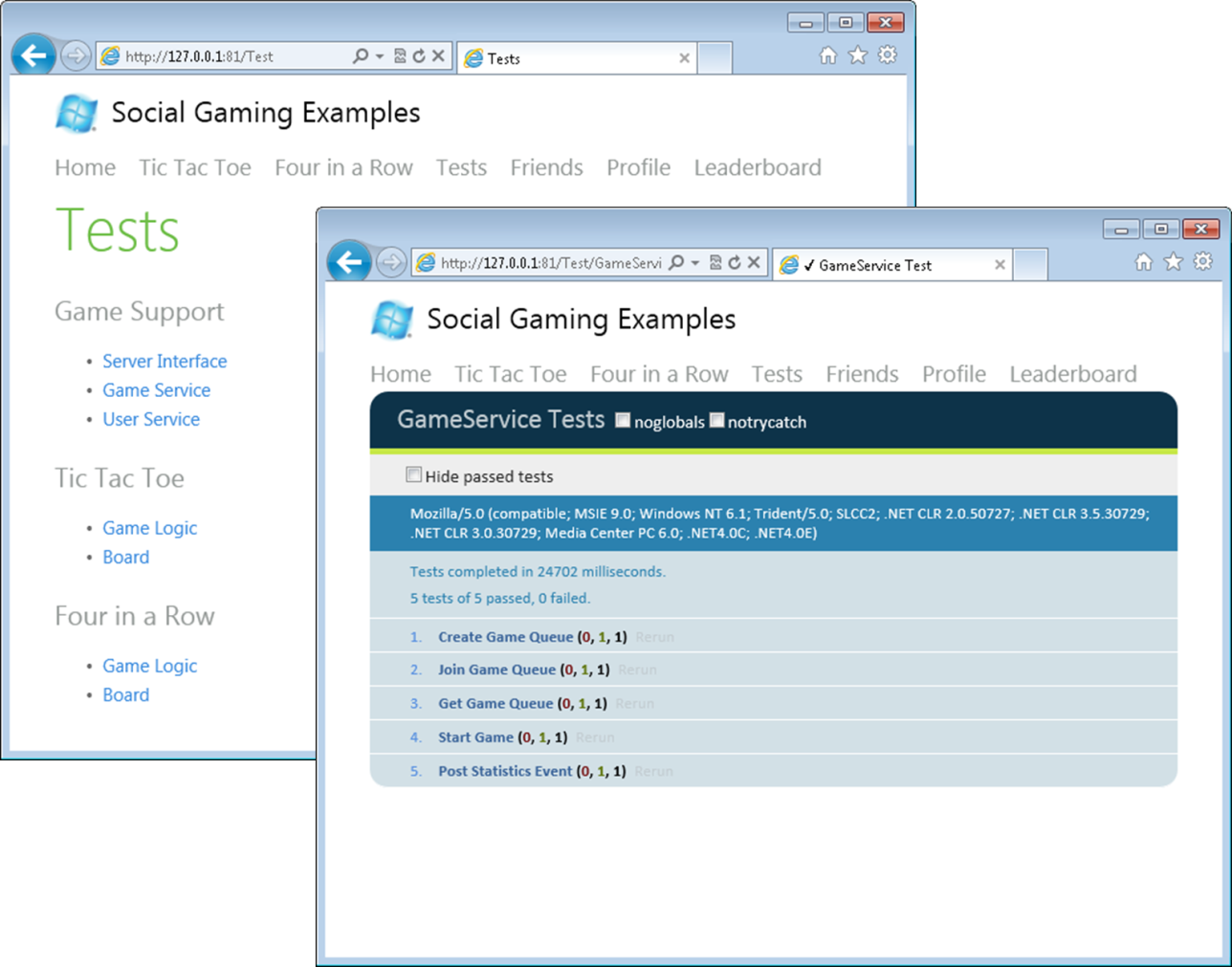
Get Started

To get started with the Windows Azure Toolkit for Social Games, you can run it locally using the Windows Azure compute emulator and your local SQL Server.

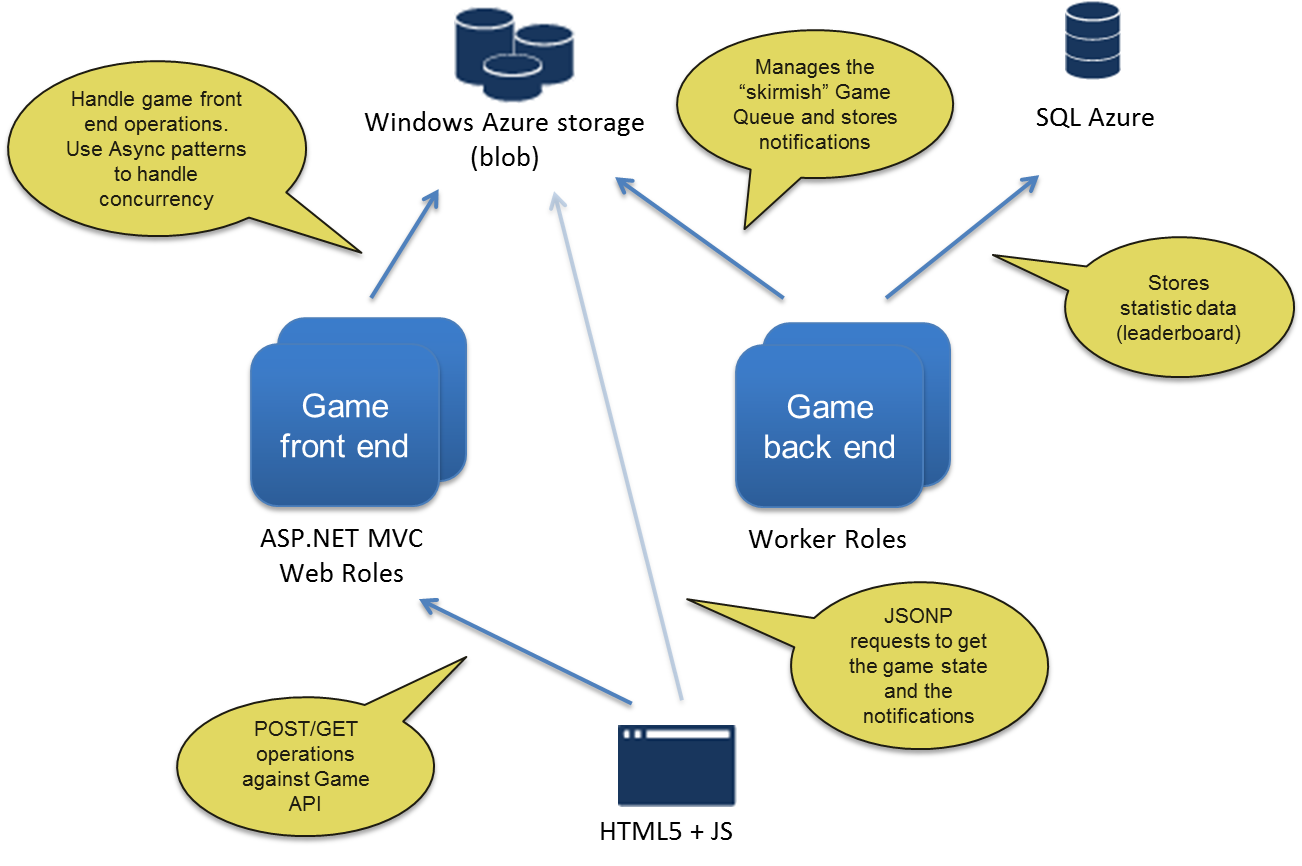
Running the Setup

* 1. Before launching the application, you first need to verify that your computer is properly configured and has all the necessary software to build and use the Toolkit.
  2. Launch **Setup.exe** located in the root folder where you extracted the sample package. Notice that it requires administrator privileges.
  3. When prompted, click **ACCEPT** if you agree to the terms of the license; otherwise, you will be unable to proceed with the installation and use the Toolkit.
  4. The next step involves checking your machine for the prerequisites. If you do not have the necessary configuration or dependencies, the dependency checker will either provide a link to download any required software or launch the Web Platform Installer to install the prerequisite for you. After installing a missing dependency, click **Refresh** to initiate the detection process again.
     1. 
     2. Figure 1
     3. Checking your system for prerequisites
     4. **Note:** By default, you will not be able to proceed with the setup procedure unless the dependency checker has successfully verified every dependency. You can, however, instruct it to ignore a missing dependency by choosing the **SKIP** or the **SKIP ALL** option, but be aware that the sample might not run correctly if you do this. This option is mainly used in cases where a dependency has been superseded by a more recent software package not contemplated in the original dependency scripts.
  5. Once the detection process is complete and you have verified every prerequisite, click **Next** to launch the setup script. The script will prompt you to complete the **Configuration.xml** file, which allows you to customize the database used by the Toolkit, among other settings that you will use when deploying the sample application to Windows Azure. You may leave the file unchanged to use the default local instance of SQL Server Express.
     1. 
     2. Figure 2
     3. Running the setup script
     4. The setup script configures the Toolkit to run locally using the compute emulator. It creates a *SocialGames* database in your local SQL Express server instance (.\SQLExpress), and using Windows Authentication to connect. To use a different database server instance for the Toolkit, open the **Configuration.xml** file, which is located inside the root folder of this package, find the **Database** section, and update the **ServerName** setting to point to the correct SQL Server or SQL Azure instance. Similarly, configure the **DatabaseName** setting to use a different database.
     5. After changing the settings, please run the setup again.
     6. 
     7. Figure 3
     8. Configuring database settings
     9. **Note:** The **UserName** and **Password** settings are not required when using integrated authentication. The **Location** setting is only used when creating a new SQL Azure database server when deploying to Windows Azure.
  6. Once the setup script has completed its execution, click **Close** to exit the dependency checker.
     1. 
     2. Figure 4
     3. Completing the setup

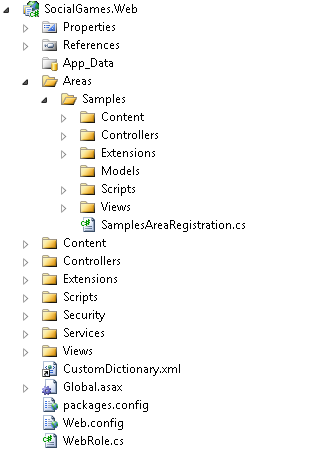
Running the Samples

* 1. Open Visual Studio as administrator from **Start** | **All Programs** | **Microsoft Visual Studio 2010** by right clicking the Microsoft Visual Studio 2010 shortcut and choosing **Run as administrator**.
  2. Using Visual Studio, open the **SocialGames.sln** solution located under the **code** folder of the Toolkit package.
  3. Make sure that the **SocialGames.Cloud** project is selected as the start-up project (shown in **bold**).
     1. 
     2. Figure 5
     3. Solution Explorer showing the SocialGames solution
  4. Press **CTRL+F5** to build and deploy the application to the compute emulator. Your default Web browser should open pointing to <http://127.0.0.1:81/> and show the Social Gaming Examples home page.
     1. **Note:** By default, the application is configured to use port 81, so you should make sure this port is free before running the application.
     2. 
     3. Figure 6
     4. Social Gaming Examples home page
  5. Click **Log On** to authenticate. You will be redirected to the login page.
  6. Sign in using your Windows Live ID or Facebook:
     1. 
     2. Figure 7
     3. Login page
     4. **Note:** This sample uses a pre-configured AppFabric Access Control Service for managing user identity and access control. If you have a Windows Azure account and you want to use your own service, you can do so by updating the settings in the **Web.config** file of the **SocialGames.Web** project and the **ServiceConfiguration.cscfg** file in **SocialGames.Cloud**.
  7. Click **Profile** and change your display name to something else.
     1. 
     2. Figure 8
     3. Profile page
     4. **Note:** For Live ID users, the display name shown by default is empty the first time they authenticate. This is because there is no claim associated with the Live ID user name in the ACS namespace configuration. You should update the display name manually in the Profile page.
  8. Go back to the Tic-Tac-Toe page and copy the invite URL shown on the screen.
     1. 
     2. Figure 9
     3. Invite page
  9. Open a second browser window using InPrivate mode (**Ctrl+Shift+P** in Internet Explorer) and browse to the invite URL. Sign in using a different account. Wait until the board is displayed and then start playing by taking turns in each of the two browser windows.
     1. 
     2. Figure 10
     3. Online multiplayer Tic-Tac-Toe game
     4. **Note:** For Live ID users, the display name shown by default is empty the first time they authenticate. This is because there is no claim associated to the Live ID user name in the ACS namespace configuration. You should update the display name manually in the Profile page.
  10. Once the game finishes, click the **Leaderboard** menu option. A board will be shown, listing the players and their current scores.
      1. 
      2. Figure 11
      3. Leaderboard page
  11. Now, click the **Four in a Row** menu option to play a different game. Make sure to select the player to invite from the combo box and then click **Invite Friend**.
      1. 
      2. Figure 12
      3. Inviting a friend
      4. **Note:** After accepting an invitation, the game sets a friend relation between the players of an invited game.
  12. Switch to the browser window of the invited player and click the **Friends** menu option. The invitation to the game should be displayed in the **Invitations** section. Click **Go** to start the game.
      1. 
      2. Figure 13
      3. Invite notifications
  13. Wait until the board is displayed and start playing by taking turns in each of the two browser windows.
      1. 
      2. Figure 14
      3. Online multiplayer Four in a Row game
  14. The sample web site also provides a set of tests for the different client components of the game. To execute the tests, click the **Tests** menu option and select the test fixture to run.
      1. 
      2. Figure 15
      3. Running the tests

The Solution

* 1. The design of the Toolkit is organized in a way that facilitates other developers to identify the reusable components and build their own social games.
  2. **High Level Architecture**
  3. 
  4. Figure 16
  5. High level architecture

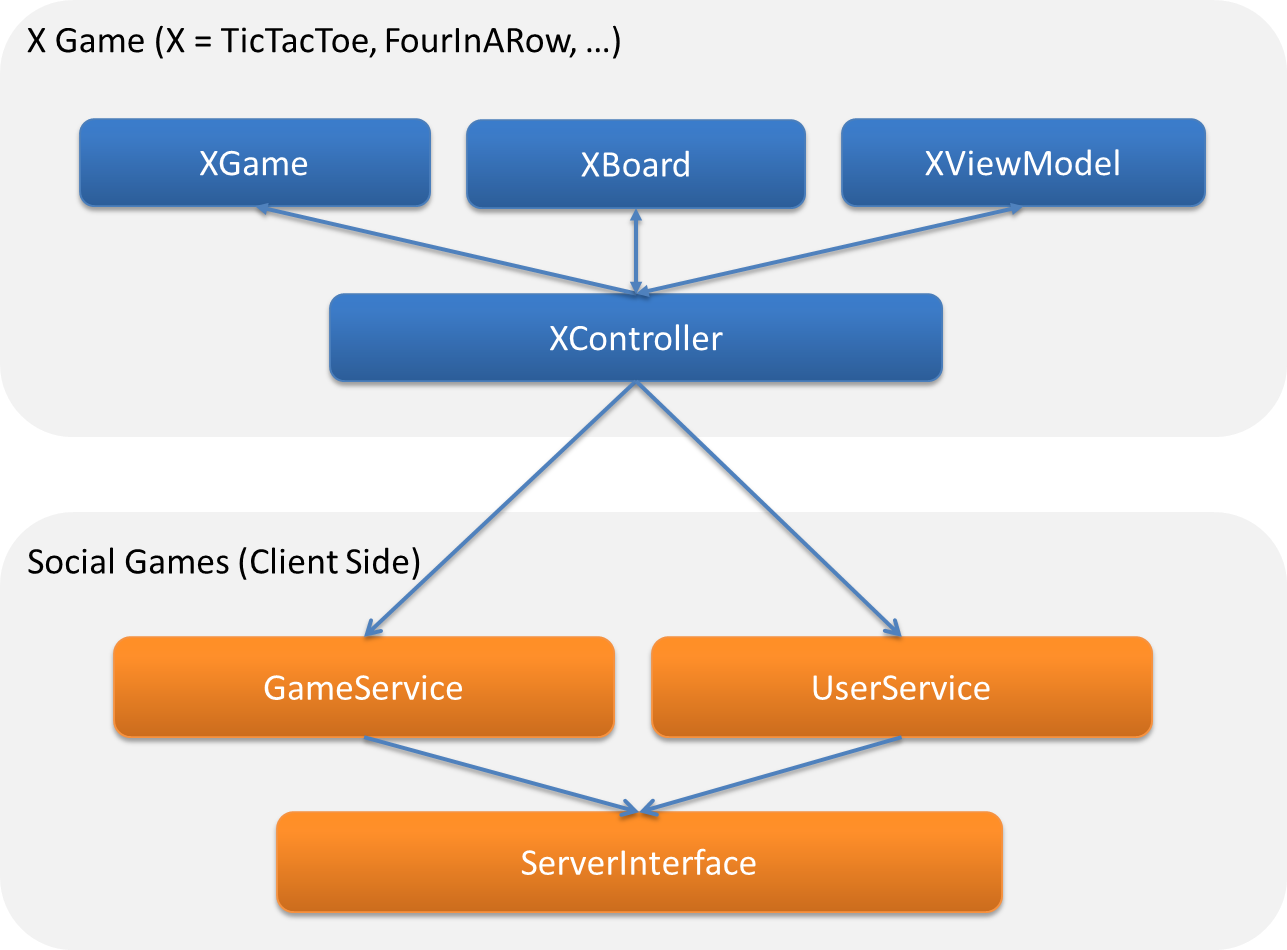
Solution Structure

* 1. Open Visual Studio from **Start** | **All Programs** | **Microsoft Visual Studio 2010**.
  2. Open the **SocialGames.sln** solution located under the **code** folder of the Toolkit package. The solution includes the following projects:
     + **SocialGames.Core**: This project contains the model of the Social Game API. Here is where the all the game entities and the repositories are located.
     + **SocialGames.Worker:** This project contains all the processes that perform background tasks, like storing invitations, collecting game statistics, etc.
     + **SocialGames.Web:** This is an ASP.NET MVC project that hosts the Game Service API endpoints, as well as the “Tic-Tac-Toe” and “Four in a Row” game samples shipped with the Toolkit.
     + **SocialGames.Cloud:** This is the Windows Azure project associated to the role projects in the solution (**SocialGames.Web** and **SocialGames.Worker** projects). The local configuration is already in place to run the services in the Windows Azure compute emulator.
     1. 
     2. Figure 17
     3. SocialGames solution
  3. Let us take a look in more detail to the **SocialGames.Web** project. In Solution Explorer, expand the **SocialGames.Web** project node.
     1. This project contains, at the root level, all common and reusable components to build social games and it uses an MVC area for all code related to a specific game implementation. In this case, the code specific to the “Tic-Tac-Toe” and the “Four in a Row” games is placed in that MVC area.
     2. 
     3. Figure 18
     4. SocialGames.Web project
  4. Expand the **Views** folder at the root level of the web project. Do the same for the **Views** folder located inside the **Samples** MVC area.
     + The **common views** are for the features of the Social Game API that are common to all the games, such as authentication, support for managing friends and user profiles, and a simple leaderboard.
     + The **game specific views** render the game play flow for different types of game.

|  |  |
| --- | --- |
| Common views | Game specific views |
|  |  |

* 1. Expand the **game** subfolder inside the **Script** folder at the root level. Do the same for the **game** folder inside the **Script** folder at the **Samples** MVC area.
     + The **common JavaScript game files** interact with the service operations provided by the Game Service API. They take care of handling the communication through HTTP requests and sending the appropriate response to the game specific components.
     + The **game specific JavaScript files** manage the behavior of a particular game. For example, they know how to render the game board, execute the game rules, etc.

|  |  |
| --- | --- |
| Common JavaScript game files | Game specific JavaScript files |
|  |  |

* + 1. Below is a diagram illustrating the interaction between the common and game specific JavaScript components:
    2. 
    3. Figure 19
    4. JavaScript components diagram

Deployment

This section provides guidance on setting up the Windows Azure Toolkit for Social Games for deployment to the Windows Azure Platform.

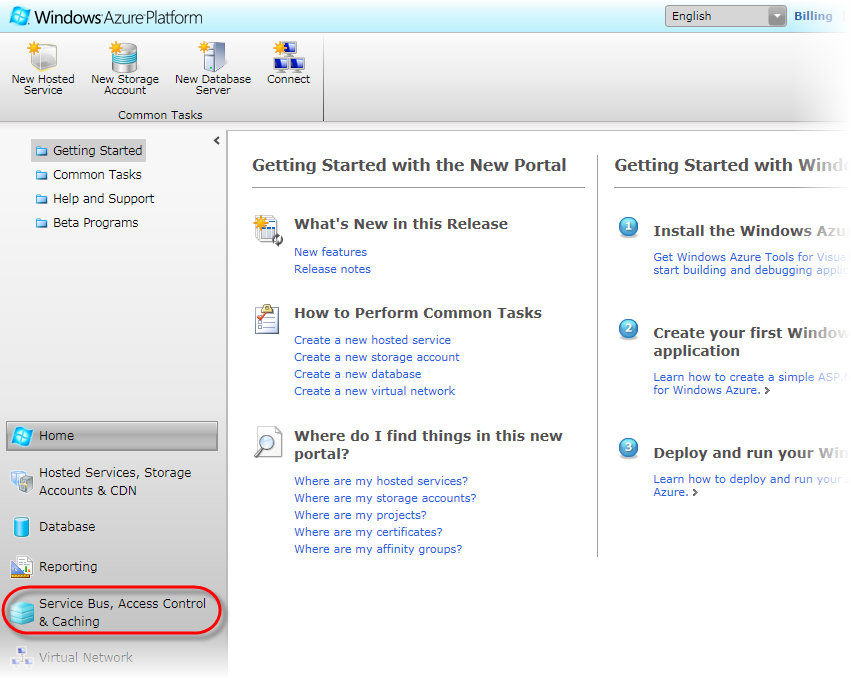
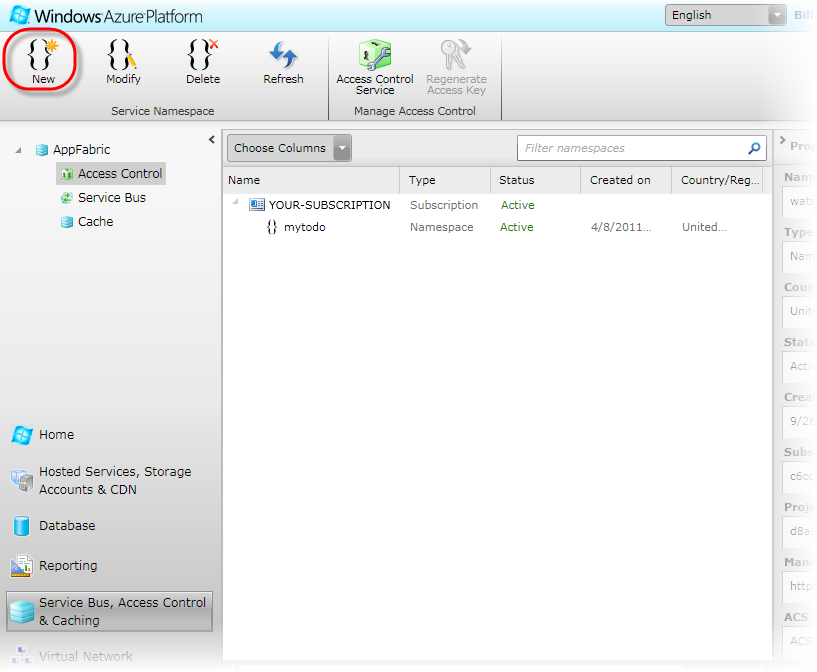
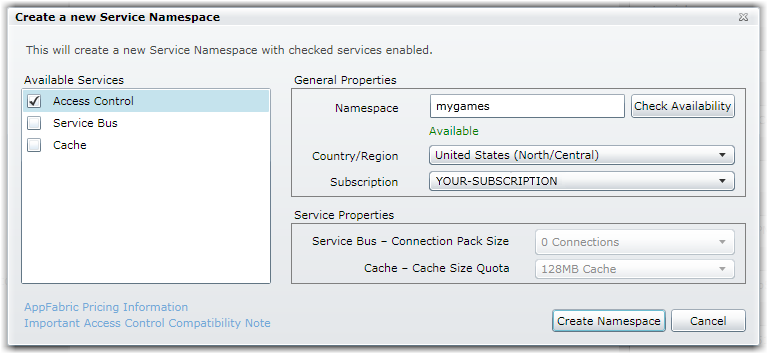
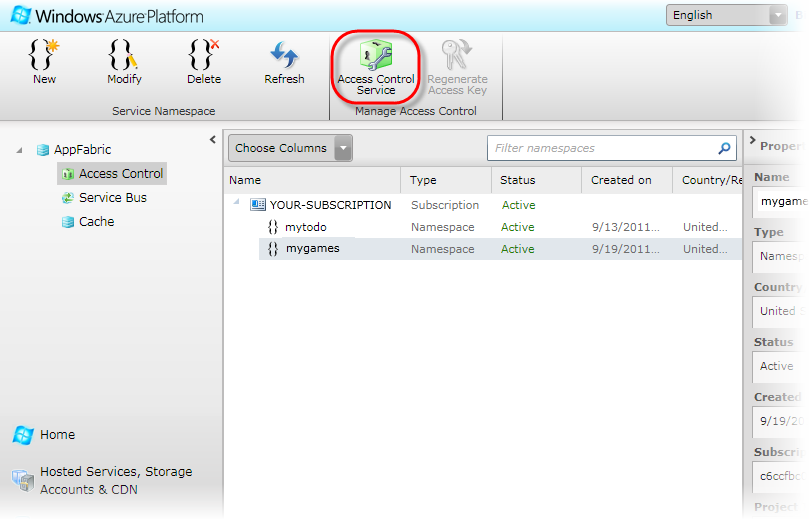
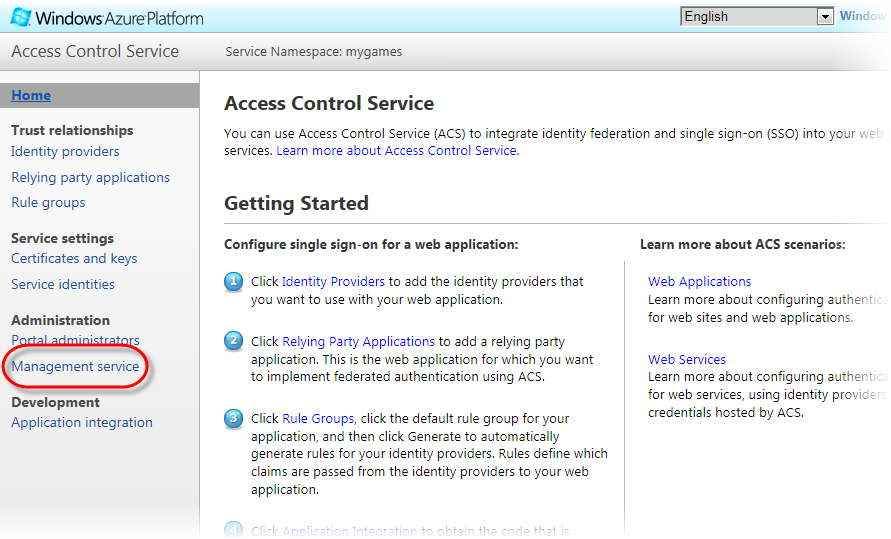
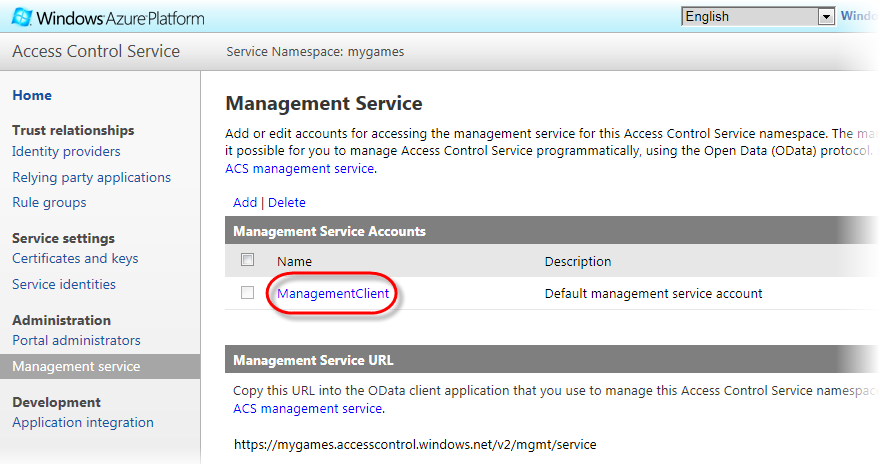
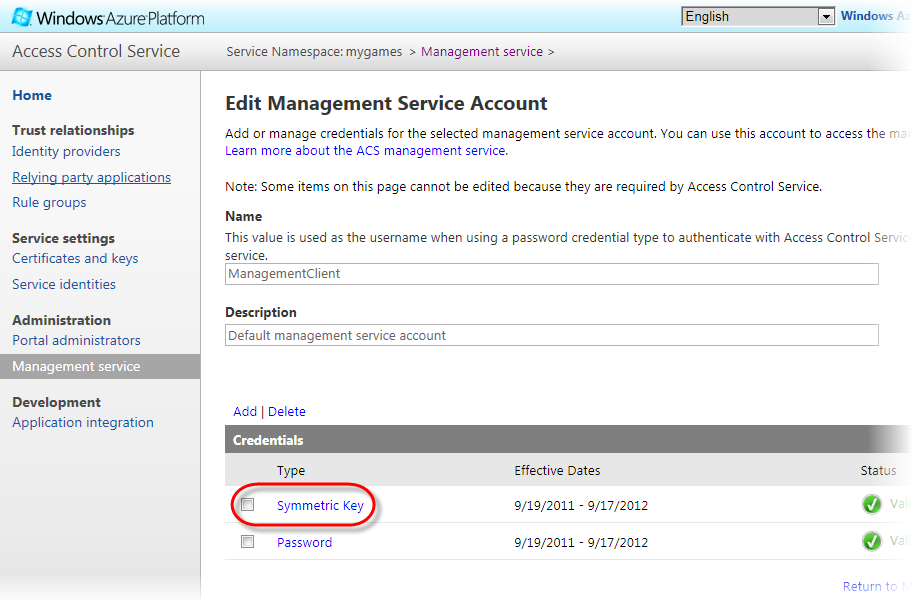
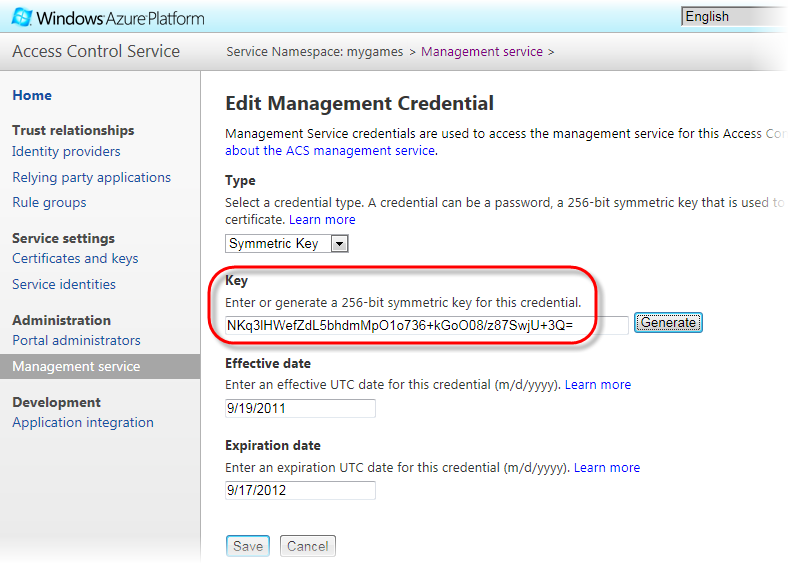
Altogether, deploying the Toolkit to Windows Azure involves the following steps:

* + Creating an AppFabric Access Control Service (ACS) namespace
  + Creating a Windows Azure Storage account
  + Creating a SQL Azure server
  + Creating a Windows Azure hosted service to deploy the sample application
  + Editing an XML file to provide the following configuration settings:
    - SQL Azure server and database settings
    - Windows Azure Storage account name and key
    - Hosted service name
    - AppFabric Access Control Service namespace and management key
    - AppFabric Access Control Service (ACS) configuration values (namespace, which identity providers to use, etc.)
  + Deploying the service package to the hosted service
  1. Although you can carry out all the steps to deploy the Toolkit manually, the package includes a deployment script that takes advantage of the [Windows Azure Platform PowerShell Cmdlets](http://wappowershell.codeplex.com) to fully automate the deployment of the Toolkit to Windows Azure.
  2. You only need to specify a few configuration settings in the **Configuration.xml** file and then execute a script that builds the service package, creates the hosted service and storage accounts, as well as a SQL Azure server to host the application database.
  3. **Note:** For more information about deploying applications to Windows Azure, refer to the following MSDN article: <http://msdn.microsoft.com/en-us/library/gg651132.aspx#deploy>

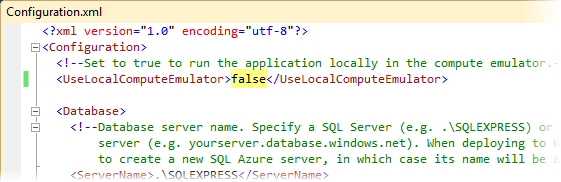
Creating an AppFabric Access Control Service Namespace

* 1. The Toolkit takes advantage of the AppFabric Access Control Service to enable authentication using one of several configured identity providers, for instance Windows Live ID and Facebook.
  2. In this task, you configure an AppFabric Access Control Service namespace that the application uses to enable this support.
  3. Start by browsing to the Windows Azure Management Portal at [http://windows.azure.com](http://windows.azure.com/) and sign in with any valid Windows Azure account.

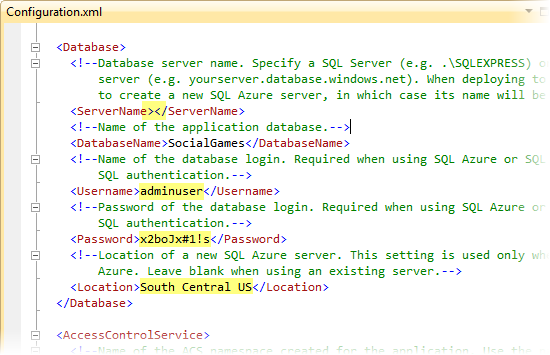
**Note:** The Windows Azure Management Portal is where you create and configure your hosted services, storage accounts, SQL Azure databases, and other offerings from the Windows Azure platform.

* 1. Next, create a new **Service Namespace**. In the home page of the Windows Azure Management Portal, select **Service Bus, Access Control and Caching** in the lower half of the navigation pane.
     1. 
     2. Figure 20
     3. Windows Azure Management Portal
  2. Now, under the **AppFabric** subarea in the upper half of the navigation pane, select **Access Control** and then click **New** on the ribbon.
     1. 
     2. Figure 21
     3. Managing AppFabric Access Control Service namespaces
  3. In the **Create a new Service Namespace** dialog, choose a **Namespace** name and a **Country/Region** for the new namespace. Then, verify that the chosen namespace is available by clicking **Check Availability** and that **Access Control** is selected under **Available Services**. Make a note of the name you have chosen and then click **Create Namespace**.
     1. 
     2. Figure 22
     3. Creating a new AppFabric Access Control Service namespace
     4. **Note:** Service names must be globally unique as they are in the cloud and accessible by whomever you decide to grant access.
  4. Locate the new entry in the list of configured service namespaces and then wait for its **Status** column to show the namespace as *Active*.
  5. On the ribbon, click **Access Control Service** to browse to the ACS Management site.
     1. 
     2. Figure 23
     3. Navigating to the AppFabric Access Control Service Management site
  6. In the ACS Management site, under **Administration** in the navigation pane, select **Management Service**.
     1. 
     2. Figure 24
     3. Administering ACS management service accounts
  7. In the **Management Service Accounts** section, select the **ManagementClient** service account.
     1. 
     2. Figure 25
     3. Retrieving the properties of the default management service account
  8. In the **Edit Management Service Account** page, under **Credentials**, select **Symmetric Key**.
     1. 
     2. Figure 26
     3. Accessing the service management credentials
  9. In the **Edit Management Credential** page, make note of the value displayed for **Key**. You will need this value later, when you configure the application.
     1. 
     2. Figure 27
     3. Retrieving the management key

Preparing the Application for Deployment

* 1. Next, you need to configure the application to operate in the Windows Azure environment. This section describes every setting that you need to provide in order to deploy the application.
  2. In Visual Studio, open the **Configuration.xml** file located inside the root folder of this package.
  3. Near the top of the file, locate the **UseLocalComputeEmulator** setting and ensure that its value is set to *false* to indicate to the setup script that you will deploy the application to Windows Azure.
     1. 
     2. Figure 28
     3. Configuring the administrator secret value
  4. The Toolkit stores the leaderboard in a relational database. To configure the database server to use, locate the **Database** section in the configuration file and enter the settings required to connect to the server that will host the application’s database.
     1. The table below summarizes each setting in this section.

|  |  |  |
| --- | --- | --- |
| Setting | Description | Quick Configuration |
| **ServerName** | Database server name. Specify an existing SQL Azure server or leave blank to create a new server.  If you specify a server, use its full name (e.g. *servername.database.windows.net*) | *<leave blank>* |
| **DatabaseName** | Name of the application database. Default value is *SocialGames*. | *SocialGames* |
| **Username** | User name of the database server login. | *Adminuser* |
| **Password** | Password of the database login. | *<your-password>* |
| **Location** | Location of a new SQL Azure server. This setting is used only when deploying to Windows Azure and creating a new SQL Azure server. Specify one of the available locations: East Asia, North Central US, North Europe, South Central US, Southeast Asia, or West Europe.  Leave blank when using an existing server.  Use the same location for the hosted service, the storage account, and the SQLAzure server. | *<your-location>* |

* + 1. **Note:** The **Quick Configuration** column in this table shows the values that you need to enter when using the automated scripts to create a new SQL Azure server in the chosen location and with the specified credentials.
    2. 
    3. Figure 29
    4. Creating a new SQL Azure server in the South Central US data center with the specified credentials
  1. Authentication in the application is handled by the AppFabric Access Control Service. To configure it, find the **AccessControlService** section and enter the settings for the service namespace that you created previously. Additionally, you can enable Windows Live ID and Facebook as identity providers that the application can use by setting the value of the corresponding **UseXXXProvider** element to *true*. By default, only Windows Live ID is enabled. You need to enable at least one identity provider.
     1. The table below describes each setting in this section.

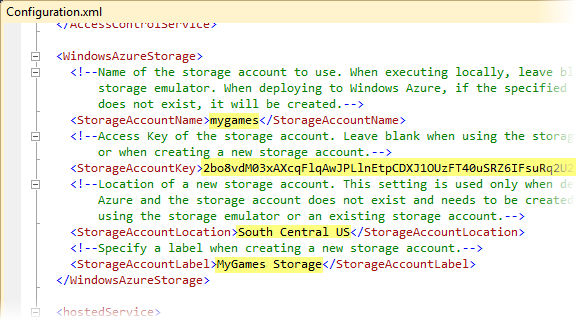
|  |  |  |
| --- | --- | --- |
| Setting | Description | Quick Configuration |
| **Namespace** | Name of the ACS namespace created for the application.  Enter the namespace only, not the full URL (e.g. *mygames*). | *{your-namespace}* |
| **ManagementKey** | Key for the management service account of the ACS namespace. | *{your-management-key}* |
| **RelyingPartyRealm** | Relying party realm.  Identifies the relying party to the identity provider.  When deploying to Windows Azure, use the HTTP endpoint of your hosted service. | *http://{your-hosted-service}.cloudapp.net/* |
| **UseWindowsLiveIdentityProvider** | Set to *true* to enable Windows Live ID as an identity provider. | *true* |
| **UseFacebookIdentityProvider** | Set to *true* to enable Facebook as an identity provider.  If Facebook is enabled as an identity provider, specify the Facebook application name, application ID and secret (see <http://msdn.microsoft.com/en-us/library/gg185919.aspx>). | *false* |
| **FacebookApplicationName** | Facebook application name. Required if Facebook is enabled as an identity provider. | *<leave blank>* |
| **FacebookApplicationId** | Facebook application ID. Required if Facebook is enabled as an identity provider. | *<leave blank>* |
| **FacebookSecret** | Facebook application secret. Required if Facebook is enabled as an identity provider. | *<leave blank>* |

* + 1. **Note:** The **Quick Configuration** column in this table shows the values that you need to enter when deploying the application to Windows Azure using Windows Live ID as the identity provider. The values for the ACS namespace and management key are your own, which you obtained previously. For the relying party realm, you need to specify the URL of the hosted service where you plan to deploy the application (e.g *http://mygames.cloudapp.net/*).
    2. For more information, see the **Creating an AppFabric Access Control Service Namespace** section.

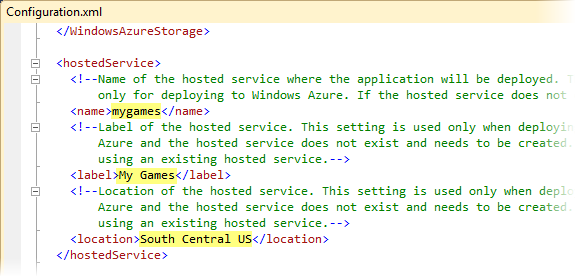
**Note:** To use Facebook as an identity provider, you need to provide a Facebook application name, Id and secret. For detailed instructions on how to do this, see this article: <http://msdn.microsoft.com/en-us/library/gg185919.aspx>**.**

* + 1. 
    2. Figure 30
    3. Configuring AppFabric Access Control Service settings for your namespace
  1. The toolkit uses blob storage to store user profiles, to persist game live state, and to store invite notifications among other things. To configure the storage account, locate the **WindowsAzureStorage** section in the configuration file and enter the settings for the storage account to use. Note that if the specified account does not exist, it will be created by the deployment script.
     1. **Note:** Storage account names are essentially DNS names and thus shared with other subscriptions in the Windows Azure environment so choose a name that is unlikely to collide with other storage account names in existence.
     2. The table below summarizes each setting in this section.

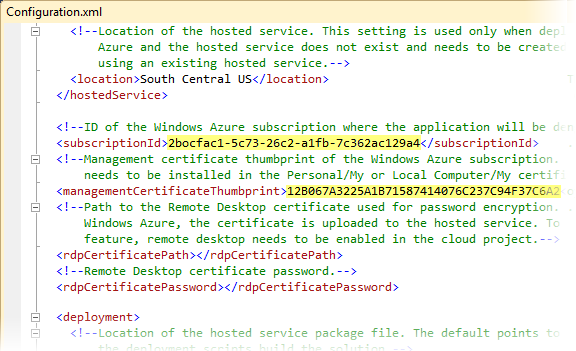
|  |  |  |
| --- | --- | --- |
| Setting | Description | Quick Configuration |
| **StorageAccountName** | Name of the storage account to use. When deploying to Windows Azure, if the specified storage account does not exist, it will be created. | *{your-account-name}* |
| **StorageAccountKey** | Access key of the storage account. Leave blank when creating a new storage account. | *{your-account-key}* |
| **StorageAccountLocation** | Location of a new storage account. Specify one of the available locations: East Asia, North Central US, North Europe, South Central US, Southeast Asia, or West Europe.  Leave blank when using an existing account.  Use the same location for the hosted service, the storage account, and the SQLAzure server. | *{your- account-location}* |
| **StorageAccountLabel** | Label for a new storage account.  Leave blank when using an existing account. | *{your- account-label}* |

* + 1. 
    2. Figure 31
    3. Configuring the storage account settings
  1. To specify where the application is to be deployed, locate the **hostedService** section in the configuration file and enter the settings for the hosted service to use. Note that if the specified hosted service does not exist, it will be created by the deployment script.
     1. **Note:** Hosted service names are essentially DNS names and thus shared with other subscriptions in the Windows Azure environment so choose a name that is unlikely to collide with other hosted service names in existence.
     2. The table below summarizes each setting in this section.

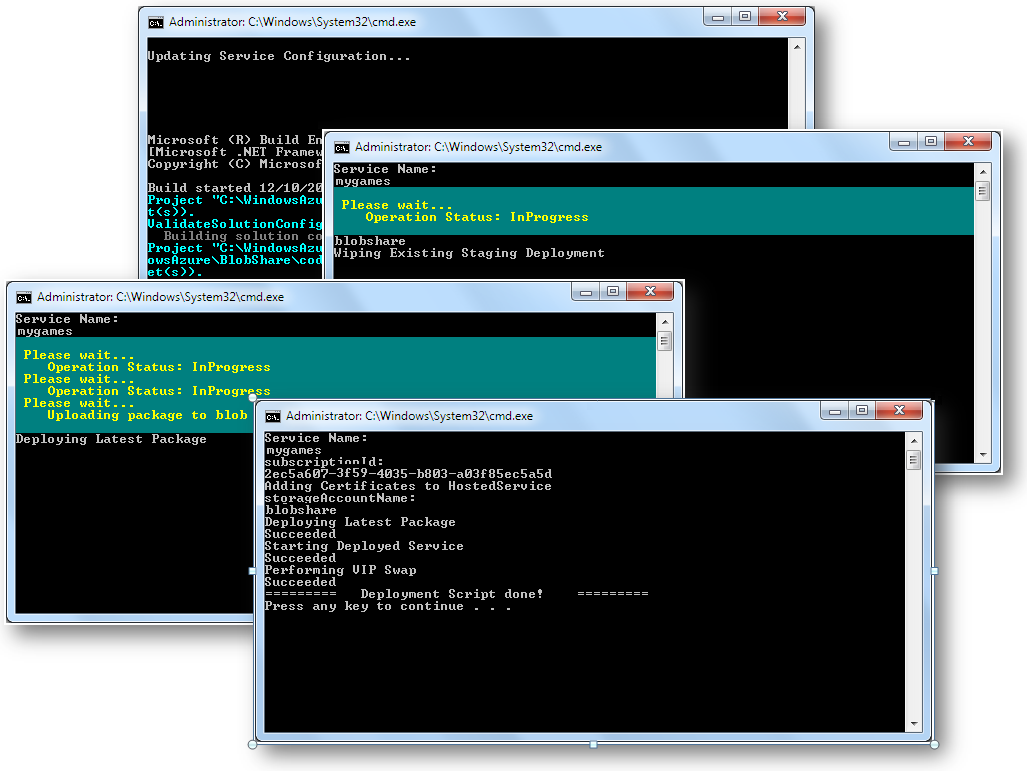
|  |  |  |
| --- | --- | --- |
| Setting | Description | Quick Configuration |
| **name** | Name of the hosted service where the application will be deployed. If the hosted service does not exist, it will be created. | *{your-hosted-service}* |
| **label** | Label of the hosted service. This setting is used only when the hosted service does not exist and needs to be created.  Leave blank when using an existing hosted service. | *{your-service-label}* |
| **location** | Location of a new hosted service. This setting is used only when the hosted service does not exist and needs to be created.  Specify one of the available locations: East Asia, North Central US, North Europe, South Central US, Southeast Asia, or West Europe.  Leave blank when using an existing hosted service.  Use the same location for the hosted service, the storage account, and the SQLAzure server. | *{your- service-location}* |

* + 1. 
    2. Figure 32
    3. Configuring the hosted service settings
  1. The [Windows Azure Platform PowerShell Cmdlets](http://wappowershell.codeplex.com) used by the automated deployment script need to access the Service Management API to execute some of the required tasks. To authenticate with the management API, they need a subscription ID and a management certificate for the account where the application is to be deployed.

Locate the **subscriptionId** setting in the configuration file and enter the ID of your Windows Azure subscription. Next, locate the **managementCertificateThumbprint** setting and specify the thumbprint of the management certificate to use.

* + 1. **Note:** The management certificate must already be uploaded to your subscription and also needs to be installed in the machine where the deployment script is to be executed.
    2. To upload the certificate, go to the [Windows Azure Management Portal](http://windows.azure.com/), select the **Hosted Services, Storage Accounts & CDN** option in the navigation pane, and then the **Management Certificates** subarea. Select **Add Certificate** on the toolbar and upload the \*.cer file with the **public** key of your management certificate.
    3. To install the certificate locally, double-click the \*.pfx file with the **private** key of your management certificate and follow the wizard to install in your certificate store.
    4. 
    5. Figure 33
    6. Configuring authentication with the Service Management API

Using the Automated Deployment Script

* 1. After you complete the previous steps and enter all the necessary information, you may now use the automated script to deploy the application to Windows Azure.
  2. The script runs unattended to create all the resources required by the application, including the hosted service, the storage account, the SQL Azure server, and it also configures the AppFabric Access Control Service by creating a relying party and its rules. By default, the script deploys the application to the staging slot of the selected hosted service and then performs a VIP swap to place the new deployment into production.
  3. If you prefer to carry out these tasks manually, you will need to create all the necessary elements manually, build the service package and then deploy it to the hosted service.
  4. To deploy the application, browse to the **setup** folder in this package and double-click **deploy.cmd** to launch the script.
     1. 
     2. Figure 34
     3. Deploying the Toolkit using the automated script
     4. **Note:** There is a known issue where the script will prematurely exit before the service completely starts and goes into the Ready state. If this occurs, you may need to wait several minutes before the service becomes available and you can access it successfully.
  5. Once the deployment is complete, open a browser window and navigate to the URL of your hosted service (e.g. *http://{yourhostedservice}.cloudapp.net*) to verify that the sample is running successfully. You may sign in and use the Toolkit.