Procedure for Migrating App Server Database Schema Changes to Production Server:

This document describes how the Entity Framework Code-First Migration works.

Ref: <http://msdn.microsoft.com/en-us/data/jj591621.aspx>

Ref: <http://www.asp.net/mvc/tutorials/getting-started-with-ef-using-mvc/migrations-and-deployment-with-the-entity-framework-in-an-asp-net-mvc-application>

Ref: <http://msdn.microsoft.com/en-us/data/jj554735.aspx>

This is based on code-first approach of the Entity Framework:

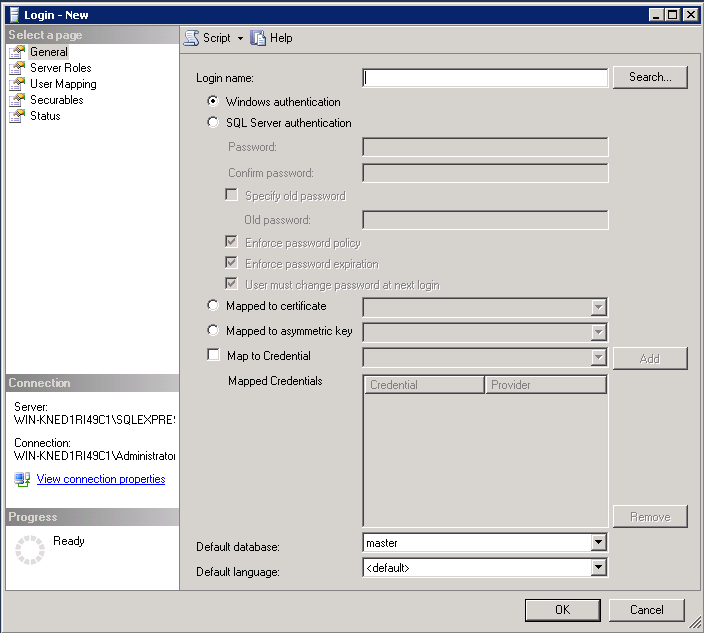
**Setting the Permissions on the remote SQL server instance:**

Proper permissions need to be set on the remote SQL server instance so that the application migration code can create and update the database.

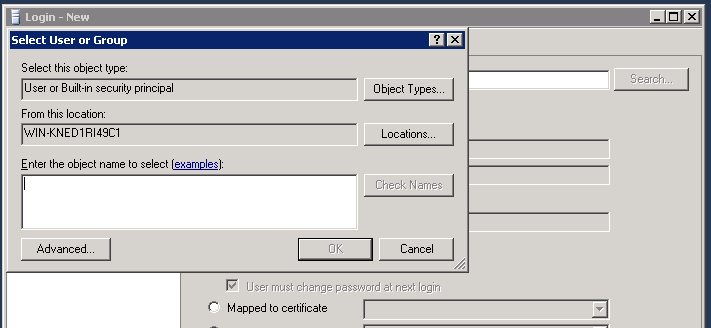
Ref: Refer to the section Method A: Server wide privileges at the following link for details:

<http://www.codeproject.com/Articles/674930/Configuring-IIS-ASP-NET-and-SQL-Server>

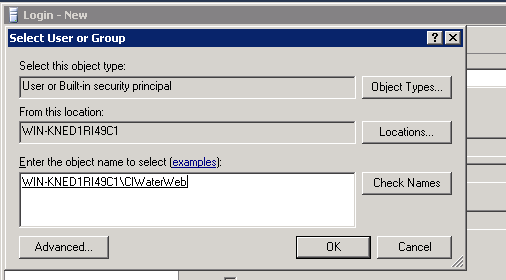
1. The account (Identity) under which the application pool for the web application is running needs to have the proper permission to the SQL server instance. In our case the CIWaterNetService web application is running under CIWaterNetService application pool which has an account (Identity) of CIWaterWeb. That means CIWaterWeb needs to have necessary permission on SQL Server instance.
2. Run SQL Server Management Studio and connect to the SQL server instance. Expand the Security section and right-click on **Logins**. Select **New Login** from the context menu. This should display the following window.



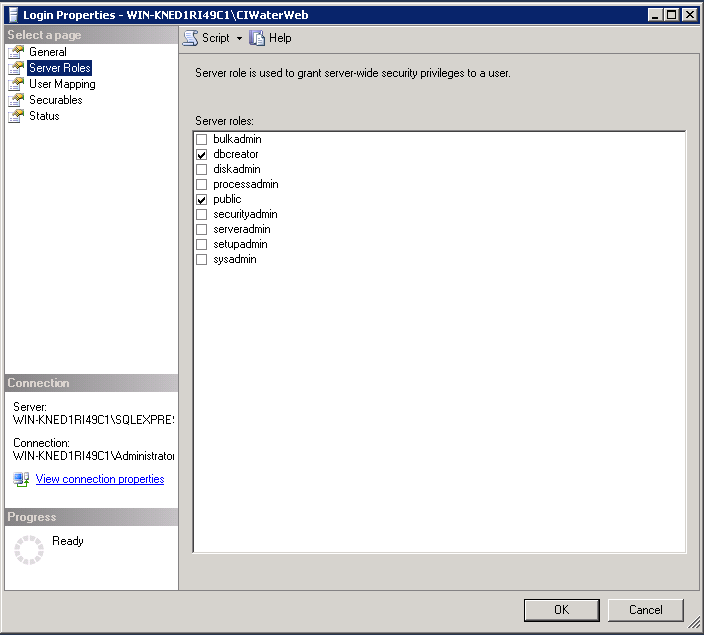
1. Click the Search button on the above window. This should display the Select User or Group dialogbox as shown below.



1. In the Enter the Object name to select field, type CIWaterWeb and then select the Check Names button. The dialogbox now should show the fully qualified name of the account as shown here.



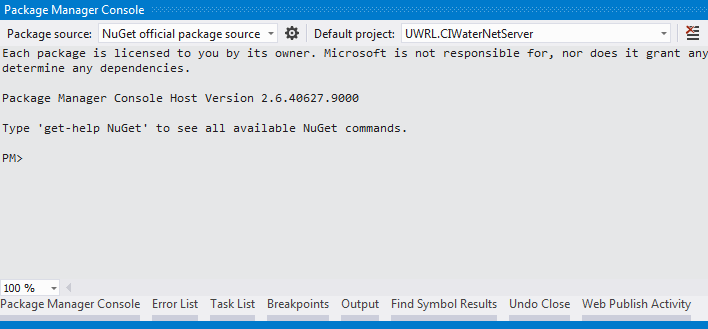
1. Select the OK button to return to the Login-New window. Make sure the radio button for Windows authentication is selected. Then select the Server Roles section from the left pane and check the checkboxes for dbcreator and publick as shown below. Then click OK to close this window.



**How to generate scaffolding migration code and write code for migration to work:**

**Ref:** <http://msdn.microsoft.com/en-us/data/jj591621.aspx>

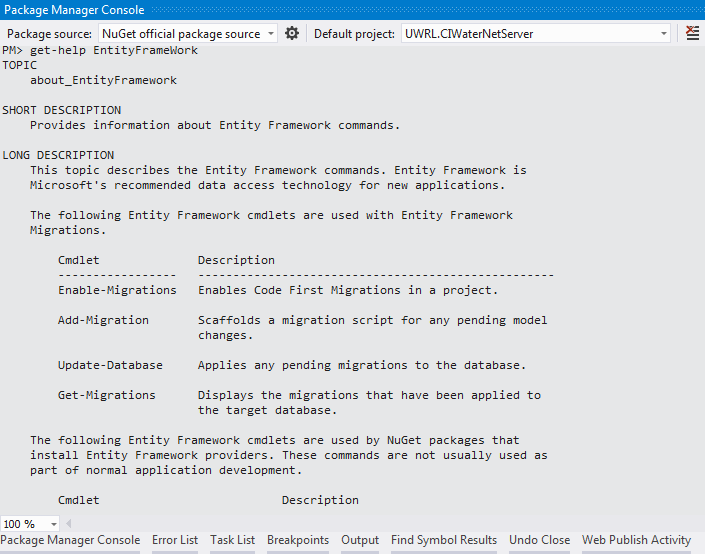
1. This procedure assumes that you have already enabled migration of schema. When migration is enabled, a Migrations folder is created in the project. See the 2nd link above to know how to enable migration of database schema.
2. Creating the initial migration in which case the production server has no database yet.
   1. All migration commands executed using the Package Manager Console within VS IDE. To go this console, select TOOLS/Library Package Manager/Package Manager Console. This will show the console window within VS IDE as shown below:



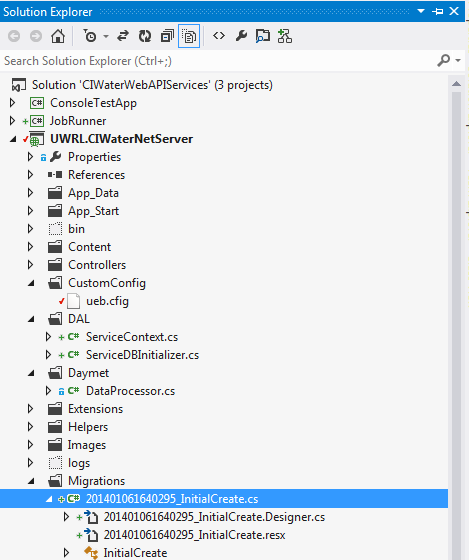
* 1. To see a list of available migration related commands, type at the PM> prompt:

PM> get-help EntityFramework

This will display the following output from the above command.



* 1. To create the initial migration which will create the database at the production server when the project is published to the server, type at the above prompt:
  2. PM>add-migration InitialCreate
  3. This will generate a class whose file name will start with {datestamp}\_InitialCreate.cs added to the Migrations folder as shown below:



If you open the InitialCreate.cs file, you will see it has code to generate the database with all tables necessary as defined by all the entity classes under the Models folder.

The enable-migration command will also create a Configuration.cs file under the Migrations folder. If you open the Configuration.cs file, you will see that it has a seed() method. The purpose of the seed() method is to initialize some of the tables in the database with data. These are mainly lookup tables. Here, you can also write code to update any data in any table or delete data from any table. If you write code that updates/delete data in any table resulting to FK constraint violation or other schema specific constraints then the migration will fail. Here is an example of the seed() method in my Configuration.cs file:

internal sealed class Configuration : DbMigrationsConfiguration<UWRL.CIWaterNetServer.DAL.ServiceContext>

    {

        public Configuration()

        {

            AutomaticMigrationsEnabled = false;

            ContextKey = "UWRL.CIWaterNetServer.DAL.ServiceContext";

        }

        protected override void Seed(UWRL.CIWaterNetServer.DAL.ServiceContext context)

        {

            //  This method will be called after migrating to the latest version.

            //  You can use the DbSet<T>.AddOrUpdate() helper extension method

            //  to avoid creating duplicate seed data. E.g.

            //

            //    context.People.AddOrUpdate(

            //      p => p.FullName,

            //      new Person { FullName = "Andrew Peters" },

            //      new Person { FullName = "Brice Lambson" },

            //      new Person { FullName = "Rowan Miller" }

            //    );

            //

            // This an example of deleting some of the existing recordd during the seeding process

            //var selectedServices = context.Services.ToList().FindAll(s => s.ServiceID > 8);

            //context.Services.RemoveRange(selectedServices);

            //context.SaveChanges();

            var services = new List<Service>

                {

                    new Service{ServiceID = 1, APIName = "EPADelineateController.GetShapeFiles", IsAllowConcurrentRun=true},

                    new Service{ServiceID = 2, APIName = "ShapeLatLonValuesController.GetShapeLatLonValues", IsAllowConcurrentRun=true},

                    new Service{ServiceID = 3, APIName = "CheckUEBPackageBuildStatusController.GetStatus", IsAllowConcurrentRun=true},

                    new Service{ServiceID = 4, APIName = "CheckUEBRunStatusController.GetStatus", IsAllowConcurrentRun=true},

                    new Service{ServiceID = 5, APIName = "GenerateUEBPackageController.PostUEBPackageCreate", IsAllowConcurrentRun=false},

                    new Service{ServiceID = 6, APIName = "GetUEBPackageController.GetPackageDownload", IsAllowConcurrentRun=true},

                    new Service{ServiceID = 7, APIName = "RunUEBController.PostRunUEB", IsAllowConcurrentRun=false},

                    new Service{ServiceID = 8, APIName = "UEBModelRunOutputController.GetModelRunOutput", IsAllowConcurrentRun=true}

                };

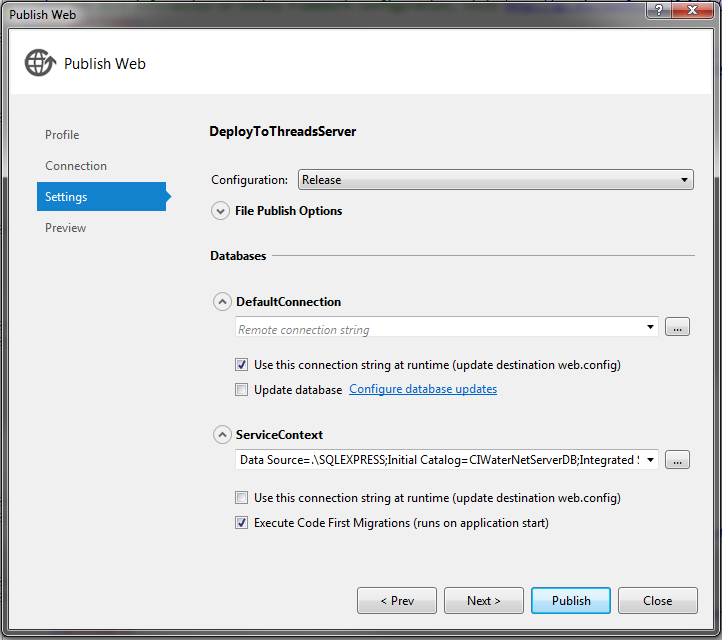
            services.ForEach(s => context.Services.AddOrUpdate(sRec => sRec.ServiceID, s));     // This will update if the ServiceID matches otherwise will insert

            context.SaveChanges();

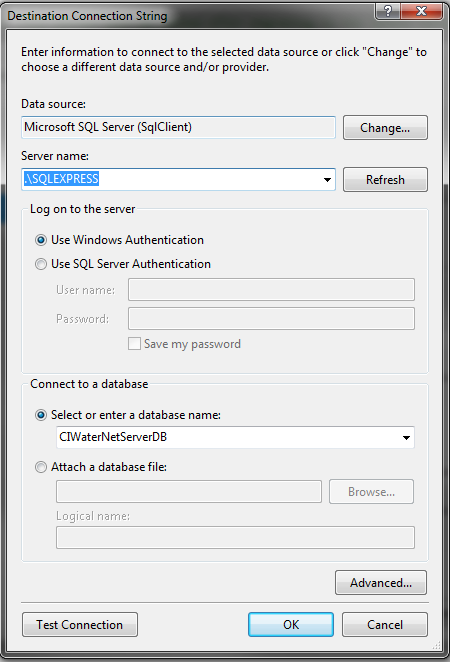
        }

    }

1. Now if you publish the project, it should create the database when the app server application first tries to use the EntityContext object defined in your code. You can test if the InitialCreate migration code will in fact create a new database on your local server, by changing the name of the database in the connect string in the web.config file and running the app server locally. In the server explorer you should see the new database created with all the necessary tables. Once you do this local testing, change the name of the database in the web.config file to the actual database name that you want on the production server and then publish the project.
   1. To deploy the database you'll use Code First Migrations (see <http://www.asp.net/mvc/tutorials/deployment/deployment-to-a-hosting-provider/deployment-to-a-hosting-provider-deploying-to-iis-as-a-test-environment-5-of-12#efcfmigrations> for details).  When you create the publish profile that you use to configure settings for deploying from Visual Studio, you'll select a check box (last checkbox) labeled **Execute Code First Migrations (runs on application start)**.  This setting causes the deployment process to automatically configure the application Web.config file on the destination server so that Code First uses the MigrateDatabaseToLatestVersion initializer class. The Publish settings dialogbox in VS is shown below:



* 1. Under the databases section in the above dialogbox. There are 2 connections that VS finds from the project. The DefaultConnection for the ASP membership local database that exists under the App\_Data folder. The ServiceContext connection is for the Entity Framework Context class that we have created (ServiceContext.cs) under the DAL folder. Here we need to set the connection properties for the ServiceContext connection for the migration to use to connect and create/update the database on destination server. Use the browse button to bring up the following window. Type the SQL server instance name in the Server name field (this is the local sql serverinstance name. Select Use Windows Authentication. Then select the database from the drop down list that your application is using. You can click the Test Connection button and that should shows the connection was successful. Then select the OK button to close this dialog box and return to the Publish settings window. Now in the Publish window you should see the connection string settings filled out for ServiceContext.



* 1. Before you deploy your application make sure your web.config file does not have settings to Initialize database as shown in the commented section **(<contexts></contexts>** of the web.config file here:

<entityFramework>

    <!--<contexts>

      <context type="UWRL.CIWaterNetServer.DAL.ServiceContext, CIWaterNetServer" >

      <databaseInitializer type="UWRL.CIWaterNetServer.DAL.ServiceDBInitializer, CIWaterNetServer" />

      </context>

    </contexts>-->

    <defaultConnectionFactory type="System.Data.Entity.Infrastructure.LocalDbConnectionFactory, EntityFramework">

      <parameters>

        <parameter value="v11.0" />

      </parameters>

    </defaultConnectionFactory>

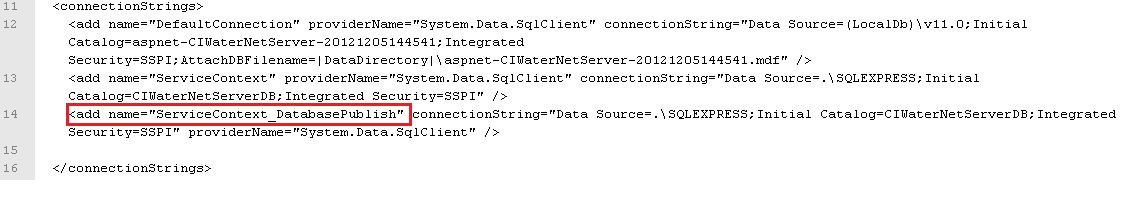
    <providers>

      <provider invariantName="System.Data.SqlClient" type="System.Data.Entity.SqlServer.SqlProviderServices, EntityFramework.SqlServer" />

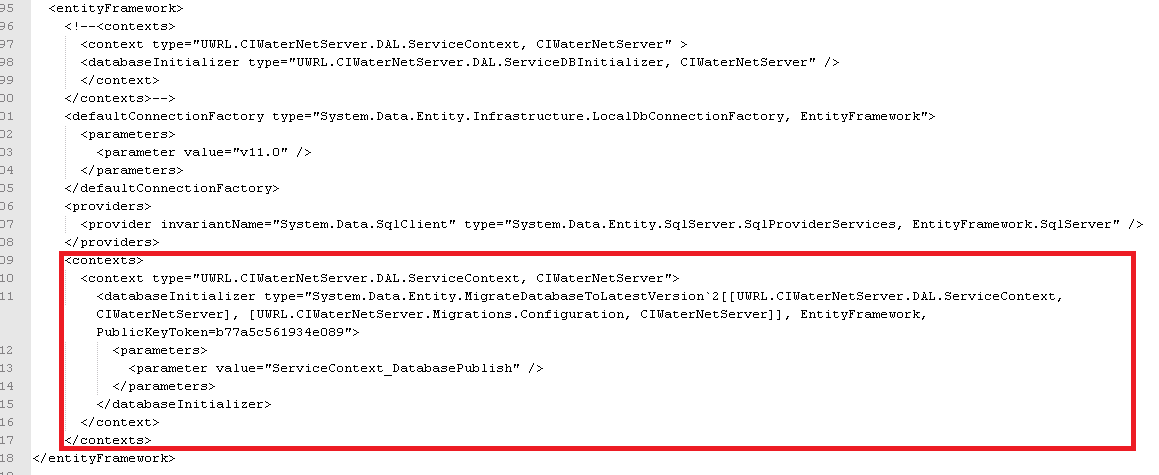
    </providers>

</entityFramework>

* 1. Now if you publish the web application project, it should modify the web.config file as shown below highlighted adding a new connection string named ServiceContext\_DatabasePublish.



* 1. It should also add the following highlighted section to the remote web.cofig file when you publish: Notice the tag <databaseInitializer> with attribute **type** set to “MigrateDatabaseToLatestVersion”. This setting makes the migration code to run automatically after the app is deployed and any client first time access this app after the deployment.



* 1. If the instance of the sql server running on remote server has a different name or the name of the database on the remote server is different then you need edit the relevant connection string properties for the connection string “ServiceContext\_DatabasePublish”. Note that this connection string is used only for database migration. For database access during program execution, the connection string “ServiceContext” is used. Since it is the same database both the migration and the application logic will use, the connection settings for both of these connection strings need to be the same.
  2. NOTE: Visual Studio doesn't do anything with the database during the deployment process while it is copying your project to the destination server. When you run the deployed application and it accesses the database for the first time after deployment, Code First checks if the database matches the data model. If there's a mismatch, Code First automatically creates the database (if it doesn't exist yet) or updates the database schema to the latest version (if a database exists but doesn't match the model). If the application implements a Migrations Seed method (Configuration.cs file), the method runs after the database is created or the schema is updated.
  3. After you publish your web application with the first migration that suppose to create the database on the remote server, access the CKAN site and run through at least one functionality that you know will call the web application and also use the Entity Framework context object.
  4. Use SQL Server Management studio to verify that the database was created.

1. Adding a new table to the database
   1. Once you create a new entity class (say TestTable) in the Model folder of the project, select TOOLS/Library Package Manager/Package Manager Console if the PM console is not already displayed. Type at the PM> prompt:

PM> add-migration addTestTable

This will create a new class file datestamp\_addTestTable.cs under the Migrations folder. If you open this file, you will see that it has code to add a new table to the database called TestTable.

Here is an example code for the addTestTable.cs file:

namespace UWRL.CIWaterNetServer.Migrations

{

    using System;

    using System.Data.Entity.Migrations;

    public partial class addTestTable : DbMigration

    {

        public override void Up()

        {

            CreateTable(

                "dbo.TestTable",

                c => new

                    {

                        ID = c.Int(nullable: false, identity: true),

                        Name = c.String(),

                    })

                .PrimaryKey(t => t.ID);

        }

        public override void Down()

        {

            DropTable("dbo.TestTable");

        }

    }

}

In the above code the Up() method is called to implement changes for a migration. When you enter a command to roll back the update, Migrations call the Down() method.

* 1. Now if you run the PM command (-v option for verbose which allows all the sql commands executed as part of the database update)

PM> update-database -v

You will see a new table called TestTable is added to the database locally. If you publish the web application to production server, this new table will be added to the remote database when the application access the database first time after the deployment, Code First checks if there is a mismatch between the data model and the database and if there is then it will run the latest migration code to update the database.

1. Deleting a Table from the database
   1. Delete the entity module (say TestTable) from the Models folder. Run the following migration command at the PM> prompt:

PM> add-migration dropTestTable

This will create a new class file datestamp\_dropTestTable.cs under the Migrations folder. If you open this file, you will see that it has code to delete table from the database called TestTable.

namespace UWRL.CIWaterNetServer.Migrations

{

    using System;

    using System.Data.Entity.Migrations;

    public partial class dropTestTable : DbMigration

    {

        public override void Up()

        {

            DropTable("dbo.TestTable");

        }

        public override void Down()

        {

            CreateTable(

                "dbo.TestTable",

                c => new

                    {

                        ID = c.Int(nullable: false, identity: true),

                        Name = c.String(),

                    })

                .PrimaryKey(t => t.ID);

        }

    }

}

* 1. Now if you run the PM command

PM> update-database -v

You will see the TestTable is deleted from the local database. If you publish the web application to production server, this table will be deleted from the remote database when the application access the database first time after deployment, Code First checks if there is mismatch between the data model and the database and if there is then it will run the latest migration code to update the database.

1. The database migration concept related to update an entity class is similar to the adding/removing a table from the database. After you make changes to an entity class, you will use the “add-migration” command to create a new migration script. Then you will use the “update-database” command to update the schema locally. If the update would lead to loss of data, then the update-database command will fail. In that case you need to use the –Force parameter to the command as shown below to force database update even if there is loss of data:

PM> update-datatabase –force

NOTE: In the Up() method of the migration code generated when you use the “add-migration” command, you can manually add additional housekeeping code to avoid situation like FK constraint violation.