Continuous integration

If you haven't already:

1. Install the ReadyRoll Visual Studio extension on your dev machine.
2. Create a ReadyRoll project, add a migration script, and commit it to source control, see Getting started.
3. Install the ReadyRoll Build components on your build system.

If you are using VSTS/TFS, then see Create VSTS/TFS build.

Recommended

1. Open the configuration page for your existing continuous integration server (eg, TeamCity, Jenkins, Bamboo, etc)
2. Add a build step to start LocalDB to use it for the *shadow database*:

   Command Line

   ```
   "C:\Program Files\Microsoft SQL Server\130\Tools\Binn\SqlLocalDB.exe"
   ```

   Arguments

   ```
   create ReadyRollShadow -s
   ```

3. Add a build step to use MSBuild to compile either the solution or the ReadyRoll project with these properties:

   MSBuild Properties

   ```
   /p:GenerateSqlPackage=True /p:ShadowServer="(localdb)\ReadyRollShadow" /p:TargetServer="$(TargetServer)"
   /p:TargetUserName="$(TargetUserName)" /p:TargetPassword="$(TargetPassword)" /p:
   TargetDatabase="$(TargetDatabase)"
   ```

   The *target database* is only used for generating the build artifacts and it will not be modified.

   If your CI server doesn't have a MSBuild task then you can execute MSBuild via a command line task:

   Command Line

   ```
   "C:\Windows\Microsoft.NET\Framework\v4.0.30319\MSBuild.exe" AdventureWorks.sln <properties from above...>
   ```

   Visual Studio 2017

   If you have Visual Studio 2017 installed on your build agents, then instead use for example: `C:\Program Files (x86)\Microsoft Visual Studio\2017\Enterprise\MSBuild\15.0\Bin\MSBuild.exe`

4. Add variables for `TargetServer`, `TargetUserName`, `TargetPassword`, and `TargetDatabase` for a database in one of your environments, for example test, staging, or production. This database is only used for generating the build artifacts and it will not be modified.
5. By default, ReadyRoll creates the build artifacts in the `bin\Release` or `bin\Debug` subfolder (change the MSBuild property `<OutputPath>` to use a different subfolder).

   Configure the build so that any files written to that folder will be saved as build artifacts.

This will create the following build artifacts:

- *Package deployment script*
- *Preview report*
- *Diff report*

- *Drift report* and *drift correction script*
- *Patch deployment script*

Build validation

By default, during a build, the migrations and programmable objects in the project are validated using the Microsoft Transact-SQL parser.

If a *shadow database* is specified, then there is an additional validation step of running the migration scripts on that database.

Any validation errors will fail the build.

`SkipProjectVerification`	Set this to `True` to produce the build artifacts without performing this validation.

Build artifacts

There are 6 different kinds of build artifacts that can be created:

1. Package deployment script

A *package deployment script* can be deployed to any environment, either by:

- Running the `ProjectName_DeployPackage.ps1` file. This PowerShell script can be used to execute the database deployment from a command prompt. Simply provide a value for the `$DatabaseServer` variable (and optionally the `$DatabaseUserName` and `$DatabasePassword` if using SQL Server Authentication to connect to your database). A set of default SQLCMD variables will be provided to the SQL script, which can be overridden as needed. Also, it will save the build artifact `ProjectName_Snapshot.nupkg.bin` into the `dbo.__SchemaSnapshot` table to record the schema that was deployed and to enable reporting in the future.
- Alternatively, in SQL Server Management Studio (SSMS), open the `ProjectName_Package.sql` file and enable **SQLCMD Mode** (from the **Query** menu). Then, uncomment the **SQLCMD Variables** section at the beginning of the script, make any adjustment to the variables as appropriate to the target environment, and then deploy to the target server(s).

`GenerateSqlPackage`	Set this to `True` to generate the *package deployment script* and accompanying PowerShell script.
`DatabasePackageVersion`	Set this to the version number to use in the *package deployment script* (e.g. `3.1.4.1337`). If it is not specified, then, if set, the `OctoPackPackageVersion` property will be used, otherwise the `AssemblyInfo.cs` file will be used to determine the version number.
`SkipVariableValidation`	By default, the PowerShell script that ReadyRoll generates (used by VSTS/TFS, Octopus Deploy, and PowerShell command line deployment) includes validation to ensure that each of the required parameters is specified. Set this to `True` to skip variable validation during execution of the PowerShell script at deployment time.

2. Octopus Deploy package

If you would like to use Octopus Deploy to deploy the database, then see Octopus Deploy.

`OctopusNuSpecFileName`	Overrides the default `.nuspec` filename for the OctoPack process, which, by default, is `ProjectName.nuspec`
`OctoPackTargetsPath`	Overrides the `OctoPackTargetsPath`, which, by default, is the version bundled with the ReadyRoll build components.

3. Preview/Diff report

The *preview/diff report* is not available in ReadyRoll Core; it is only available in ReadyRoll Pro.

If a *shadow database* and a *target database* is specified, then a report containing the schema differences between the project state and the current version of the database is generated (as stored within your target database's [__SchemaSnapshot] table). This effectively gives you a preview of the changes that are to be made to the target database during deployment.

The *patch deployment script* is shown in the *diff report*, which includes a delta of migrations, programmable objects etc that have yet to be deployed to the database.

> The *diff report* will only be produced if the *target database* was deployed using either the *package deployment PowerShell script, the MSBuild CLI* or the *Octopus Deploy package* . This is due to the fact that the schema snapshot is not inserted into the target database unless one of these methods is used.
>
> If the *target database* was deployed using SSMS, as an Embedded Resource or directly with SQLCMD.EXE, then the schema snapshot will not be available, therefore report generation will not be available.

`SkipDeployPreview`	Set this to `True` to skip generating a *preview/diff report*.

4. Drift report and drift correction script

The *drift report and drift correction script* are not available in ReadyRoll Core; they are only available in ReadyRoll Pro.

If a *target database* is specified, then, by default, a *drift report* and a *drift correction script* are generated.

`SkipDriftAnalysis`	Set this to `True` to skip generating a *drift report* and a *drift correction script*.
`DBReSyncOnBuild`	Setting `DBReSyncOnBuild` will modify the *target database* schema so that it is equal to the *shadow database* schema. If set to `True`, then a build will automatically run the *drift correction script* on the *target database*. In the event that drift is detected within the target server/database and the build is successful, then the drift-correction script (`ProjectName_ReSync.sql`) will be automatically executed against the database with this property is set to `True` (presuming that the data-loss conditions as described within the below `DriftOptionBlockDataLoss` option have not been triggered, or that option is set to `False`). If this option is left as `False`, then the script will simply be output to the `bin\<Configuration>` folder to allow to be manually executed, if desired.
`DriftOptionDropMissingObjects`	Set this to `True` to cause objects that exist in the target server/database but not in the project to be dropped as part of the *drift correction script*. By default, ReadyRoll will ignore any objects that exist in the target database that are not present in the database project. This is to accommodate the scenario where an environment contains objects that have been added as part of operational tasks rather than system-development tasks (e.g. a stored procedure used by a DBA to perform regular maintenance on the database), and are therefore considered to be outside the scope of source control. Setting this option to `True` will cause such objects to be included in drift reporting, resulting in the objects being dropped as part of the `ProjectName_ReSync.sql` script file execution (presuming that the data-loss conditions as described within the below `DriftOptionBlockDataLoss` option have not been triggered, or that option is set to `False`).
`DriftOptionBlockDataLoss`	Set this to `False` to allow data-loss causing operations to be included in the *drift correction script*. By default, in the event that drift is detected (e.g. a column added directly in Production), and the resulting drift-correction operations would result in data-loss (i.e. the removal of the hypothetical column), ReadyRoll will cause the build to fail, thus preventing the drift-correction script from being copied to the `bin\<Configuration>` folder or being automatically executed if `DBReSyncOnBuild` is set `True`.

> **Additional Drift options**
>
> Any of the `sqlproj` properties listed in the Configuring comparison & script generation options page will also affect the way in which drift is both detected and corrected. With regard to the footer section of the script, the contained migration is always generated with the following option: `SyncOptionIncludeExistenceChecks=True`. This is to assist with ensuring that the script can be executed again the target environment without re-attempting the contained operations, although some editing of the script may be required to guarantee that the operations are performed idempotently.

5. Patch deployment script

A *patch deployment script* can only be run on the *target database* that it is generated from (or a database with an identical schema). It only contains the migration scripts (if any) & programmable objects (if any) that haven't already been run on that database.

If a *target database* is specified, then, by default, a *patch deployment script* is generated (called `ProjectName.sql`).

`SkipTargetPatch`	Set this to `True` to skip generating a *patch deployment script*.

Shadow database

The *shadow database* is a temporary database where ReadyRoll can run your migration scripts. By running the migration scripts on a real database, SQL Server will check that they are valid SQL with no syntax errors, missing dependencies, etc.

> As part of the shadow deployment process, a file containing a snapshot of your schema is produced and, upon deployment, will be inserted into the target database. During subsequent builds, the snapshot will be retrieved from the target database and used to generate the deployment preview and drift reports mentioned above. If at deployment time a snapshot file cannot be not found (i.e. because the shadow was not deployed at build time), a warning will be raised to indicate the impact to report generation.

The *shadow database* can either be on LocalDB, a local instance of SQL Server Express, or any other version of SQL Server.

`ShadowServer`	SQL Server instance where the *shadow database* should be deployed.
`ShadowUserName`	By default, Windows Authentication is used. Set this to a username to use with SQL Server Authentication.
`ShadowPassword`	By default, Windows Authentication is used. Set this to a password to use with SQL Server Authentication.
`ShadowDatabase`	Overrides the name of the *shadow database*, which, by default, is `ProjectName_WindowsUserName_SHADOW`

Target database

The *target database* is a database in one of your environments, for example test, staging, or production. It will not be modified unless you specify `DBReSyncOnBuild` and/or `DBDeployOnBuild`.

`TargetServer`	SQL Server instance where the *target database* is.
`TargetUserName`	By default, Windows Authentication is used. Set this to a username to use with SQL Server Authentication.
`TargetPassword`	By default, Windows Authentication is used. Set this to a password to use with SQL Server Authentication.
`TargetDatabase`	Overrides the name of the *target database*, which, by default, is `ProjectName`

Change target database

> Whilst there are these options to change the *target database* automatically during a build, the ReadyRoll Release components should generally be used instead.

`DBReSyncOnBuild`	Setting `DBReSyncOnBuild` will modify the *target database* schema so that it is equal to the *shadow database* schema. If set to `True`, then a build will automatically run the *drift correction script* on the *target database*.
`DBDeployOnBuild`	Setting `DBDeployOnBuild` will modify the *target database*. If set to `True`, then a build will automatically run the *patch deployment script* on the *target database*. See *Drift Report And Drift Correction Script options* for further details