

Variables

SQL Change Automation allows you to deploy from the same project folder to multiple environments. However, there may be environment specific requirements for each deployment. For instance, you may want to produce more detailed logs when deploying to a User Acceptance Test environment. Instead of hard-coding these variations into your project scripts, it's possible to parameterize your deployments with the help of SQLCMD variables.

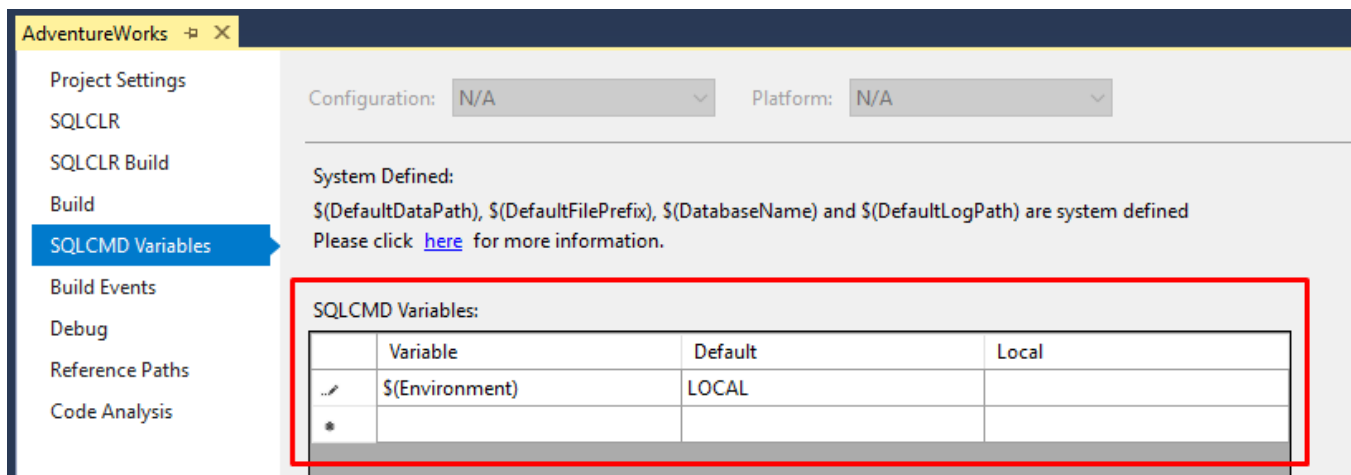
You start by adding variable definitions to your SQL Change Automation project within Visual Studio, along with default values, and when it comes to deploy your database outside of the IDE, you can supply values that are appropriate to each of your target environments. These custom values can be provided by a Continuous Integration server like TeamCity, TFS or Bamboo, using a custom installer (e.g. an InstallShield or Wix package), or with a deployment tool like [Octopus Deploy with SQL Change Automation Octopus Packages](#).

Download the [sample project \(VariableSampleProj.zip, 6kB\)](#) used in this article.

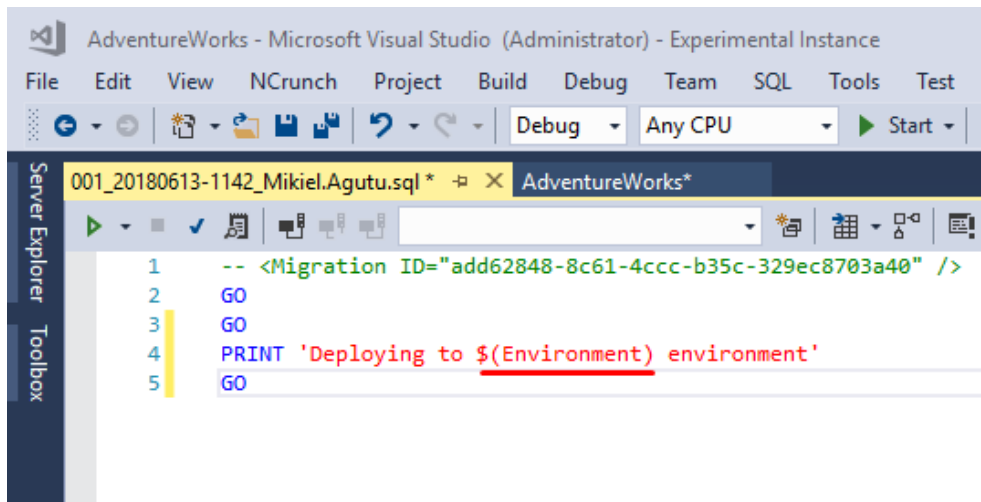
Adding a SQLCMD Variable to your project

You can add SQLCMD variables to your SQL Change Automation project within the *SQLCMD Variable* tab of project properties.

The value you provide in the *Default* column will be stored in the project file (and therefore shared with other team members) however the *Local* value is specific to your machine (stored in the non-source controlled .user file). If you leave the Local column blank, the Default will be used when deploying inside Visual Studio.



Using the `$(VariableName)` notation, reference the variable in a new migration script.





Variable qualifiers

When using variables in your scripts, qualifiers must be used in order for your scripts to be successfully parsed and validated by the build engine.

- When using a variable within an object or database name, the variable must be enclosed by [] square brackets, e.g. `SELECT * FROM [$(MyObjectNameVariable)]`
- When using a variable within a string, the variable must be enclosed by quotation marks, either with the " single-quote or "" double-quote characters, e.g. `PRINT '$(HelloWorldVariable)'`
- When using a variable that contains a numeric value, the variable must be treated like a string (as above) and cast as a numeric data type before it can be used. For example `DECLARE @myNumber int = CAST('$(MyNumberVariable)' AS int)`

When you build the project, the variable will be substituted with the *Default* value (or *Local* value, if it was provided).

```

Deploying "obj\Debug\AdventureWorks.sql" to the [AdventureWorks] database on [(localdb)\ProjectsV13]
----- executing pre-deployment script "Pre-Deployment\01_Create_Database.sql" -----
# Beginning transaction
**** EXECUTING MIGRATION "Migrations\002_20180613-1147_Mikael.Agutu.sql", ID: {0872d456-a1ba-49b6-97df-caff4d91cd21} ****
Deploying to LOCAL environment
**** FINISHED EXECUTING MIGRATION "Migrations\002_20180613-1147_Mikael.Agutu.sql", ID: {0872d456-a1ba-49b6-97df-caff4d91cd21} ****
# Committing transaction
1 migration(s) deployed successfully
----- executing post-deployment script "Post-Deployment\01_Finalize_Deployment.sql" -----
Deployment completed successfully.

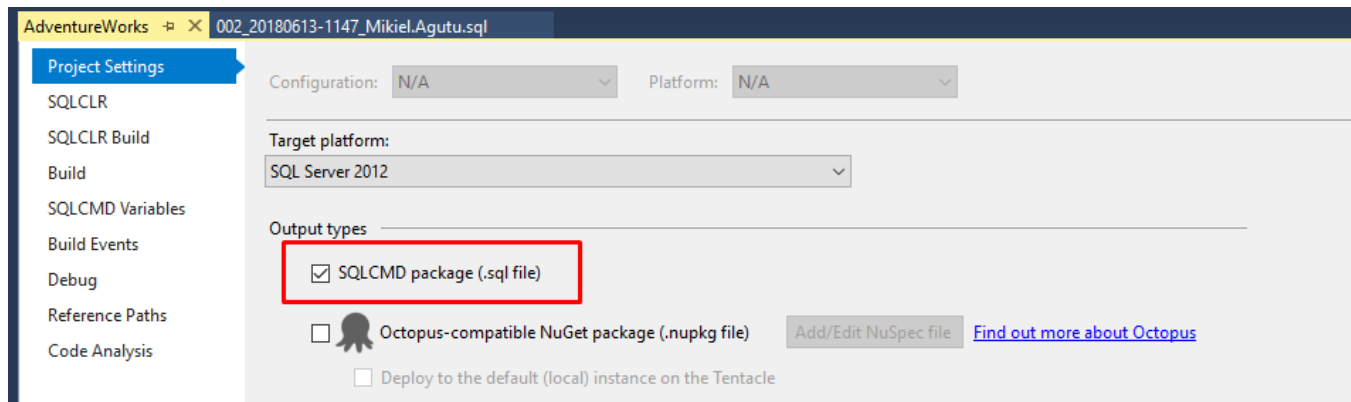
```

Deploying with environment-specific values

When building your SQL Change Automation project outside of Visual Studio, SQL Change Automation can produce a re-usable type of deployment artifact called a *SQLCMD package* (equivalent to a *DACPAC* file) which contains all of your project's migrations, SQLCLR assemblies and pre/post-deployment scripts.

This packaging format is designed with the *build once, deploy many times* approach in mind, meaning that you can use the same package to deploy to your Development, Staging & Production environments. To cater for differences between your environments, you can provide values for variables at the command line, or within the configuration of your Continuous Integration/Deployment server.

To enable package creation as part of your build, firstly enable the *SQLCMD package* option under **Output** within project properties:



Octopus Deploy

SQL Change Automation has first-class support for Octopus Deploy: any variable that you define on the **SQLCMD Variable** tab will automatically be sourced from your Octopus project variables at deploy-time.

Additionally, if you intend to use Octopus to orchestrate your database deployments, you do not need to enable the above option. For more details, see our guide on setting up SQL Change Automation projects with [Octopus Deploy with SQL Change Automation Octopus Packages](#).

Build your project to produce the package deployment T-SQL script and accompanying PowerShell script (i.e. *VariableSampleProj_Package.sql* and *VariableSampleProj_DeployPackage.ps1*).

To deploy your database, open a Command Prompt session, navigate to the output folder for your project (e.g. *bin/debug*) and execute the following command, substituting in the relevant values:

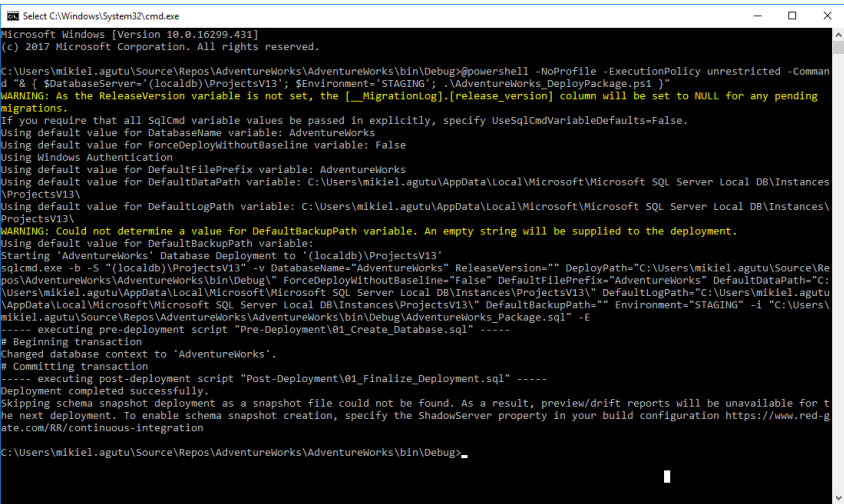
```

@powershell -NoProfile -ExecutionPolicy unrestricted -Command "& { $DatabaseServer='(localdb)\ProjectsV13';
$Environment='STAGING'; .\AdventureWorks_DeployPackage.ps1 }"

```

Or to deploy directly within PowerShell, use this command:

```
& { $DatabaseServer='(localdb)\ProjectsV13'; $Environment='STAGING'; .\AdventureWorks_DeployPackage.ps1 }
```



(Click to enlarge)


Note the list of default values that is emitted during deployment. These values are taken from the Default column in the SQLCMD variable definition, but can be overridden by simply specifying additional PowerShell variables as part of the deploy command. For example, in the above deployment, we provided a value for the **Environment** variable ("STAGING"). We also provided a value for the built-in **DatabaseServer** variable ("(local)\VA"), which is used to instruct the **SqlCmd** tool to deploy to a given instance of SQL Server.

However you can also override any of the other built-in variables. For example, you may decide to override **DatabaseName**, if you want to deploy multiple copies of the database to the same SQL Server instance (*MyDb_DEV*, *MyDb_STG*, *MyDb_PROD*).

System Variables

The following is a list of system variables that you can provide to your [package-based deployment](#) from the [PowerShell command line](#) or with corresponding [Octopus variables](#).

Name	Example	Description
DatabaseServer	MYDBSERVER\INSTANCE	Required. Name of the SQL Server machine\instance to deploy to.
DatabaseUserName	sa	Optional. Provide a value for these variables if you would like to use <i>SQL Server Authentication</i> to connect to your database server, otherwise <i>Windows Integrated Authentication</i> will be used.
DatabasePassword	*****	
UseSqlCmdVariableDefaults	True	Optional. Set to False if you require that values for all non-system SqlCmd variables be passed in explicitly, rather than using the default values as set in the project file.

 The above variables are **not** passed into the deployment script and thus cannot be used in your project migrations, programmable objects etc.

The following is a list of built-in system variables that can be used in your project scripts using SQLCMD syntax, i.e. `$(VariableName)`. Unlike the above system variables, all of these apply to both the [package and patch-based deployment methods](#).

The values for each of these variables can be overridden at the command line or with corresponding Octopus variables, with the exception of the **Package Version** variable which is set at build-time and thus is read-only.

Name	Example	Description/Default value
DatabaseName	Adventure Works	The name of the database being deployed. This value is sourced from the TargetDatabase MSBuild property, and if that has not been specified, the name of the SQL Change Automation project itself.

Release Version	1.0.0-MyRelease	<p>Release number to store against deployed migrations within the [dbo].[__MigrationLog] table. There is no default value for this variable; a warning will be emitted during deployment if it has not been set.</p> <p>The exception to this is when deploying with Octopus, in which case the value is automatically sourced from the Octopus.Release.Number system variable.</p>
PackageVersion	1.0.123-MyRelease	<p>Read-only. Package version to store against deployed migrations within the [dbo].[__MigrationLog] table. If Semantic Versioning has been enabled, it will use the current SemVer for the project. Otherwise, it will be set to <i>(undefined)</i>.</p> <p>However, if using Octopus Deploy with SQL Change Automation Octopus Packages, this value will be set at build-time either from the <i>OctoPackPackageVersion</i> MSBuild property, or if that has not been provided, from the <i>AssemblyInfo.cs</i> file within the SQL Change Automation project.</p>
ForceDeployWithoutBaseline	False	<p>Default is <i>False</i>. If you are attempting to deploy to an existing database, SQL Change Automation performs a check to ensure that a baseline has been set. This is done to ensure that no objects are accidentally dropped or overwritten during deployment. To force the deployment to continue without a baseline, set this to <i>True</i>.</p>
DeployPath	c:\Source\MyProject\	<p>Provides the full path to the currently deploying project/package, useful when needing to refer to packaged files from your scripts. In Visual Studio, this will be the root of the project (i.e. where the .sqlproj file is located), and in Octopus Deploy this will be the path that the package is extracted to, e.g. C:\Octopus\Applications\MyProject\AdventureWorks.Database\1.0.0.123\. For more information, see Seed Data.</p>
DefaultFilePrefix	AdventureWorks	<p>The filename prefix for the MDF/LDF files for your database storage. The default value for this variable is the same as DatabaseName.</p>
DefaultDataPath	D:\MSSQL11\MSSQL\DATA\	<p>The default directory paths for the MDF/LDF/BKP files, which can be used to determine where your database should be stored (provided that you have consumed these variables in the CREATE statement of your <i>Pre-Deployment\01_Create-Database.sql</i> project file).</p> <p>These variables are sourced from the instance-specific branch of the SQL Server's system registry. In the event that no default paths have been set for that SQL Server instance, the directory paths to the [master] database's files will be used instead.</p>
DefaultLogPath	D:\MSSQL11\MSSQL\LOG\	
DefaultBackupPath	D:\MSSQL11\MSSQL\BACKUP\	
IsShadowDeployment	1	<p>Read-only. Indicates that the currently executing deployment is to the shadow database. Possible values are 0 or 1.</p> <p>This can be used to gate whether to include a set of statements as part of the script verification process or not.</p>



As these variables are built into SQL Change Automation, you do not need to add definitions to the project settings *SQLCMD Variables* tab in order to use them in your migration scripts.