Continuous integration

This page provides an overview of the SQL Change Automation Powershell cmdlets for setting up continuous integration on a development database.

By setting up a trigger and build step in your build server, each time a change to the development database schema is committed to source control, your build server can run SQL Change Automation Powershell cmdlets that carry out the following tasks:

- **Build**
  Validate the schema in the source control scripts folder by checking the database can be built successfully from scratch.
- **Test**
  Run tSQLt tests on the development schema.
- **Package and Publish**
  Add the development schema to a database package and publish this for later use in your deployment process.

Build

The first task in the continuous integration process is to build the scripts folder in your source control repository by checking the database can be built successfully from scratch. You can use the `Invoke-DatabaseBuild` cmdlet to do this.

Invoke-DatabaseBuild

Example

```
$validatedProject = Invoke-DatabaseBuild "C:\Work\scaProject.sqlproj"
```

In the example above, we've assigned the output of the `Invoke-DatabaseBuild` cmdlet to the `$validatedProject` variable, so we can reuse it as the input for other SQL Change Automation Powershell cmdlets.

The SQL Change Automation add-ons for build servers, such as TeamCity, have a "Build" step. As well as validating the scripts folder, the Build step also places the schema in a database package, ready to be published to a release management system.

If you want to replicate the behavior of the Build step, you can add cmdlets that build the database package and export it to disk, as in the example below:

Example

```
$validatedProject = Invoke-DatabaseBuild "C:\Work\scaProject.sqlproj"
$buildArtifact = New-DatabaseBuildArtifact $validatedProject -PackageId "AdventureWorks" -PackageVersion "1.0"
Export-DatabaseBuildArtifact $buildArtifact -Path "c:\buildArtifacts"
```

However the SQL Change Automation Powershell cmdlets are more versatile than the SQL Change Automation add-ons, and we recommend you leave building your database package until after the other continuous integration tasks of testing and syncing have succeeded. To find out more about building your database, see Package and publish.

Test

There are two SQL Change Automation Powershell cmdlets related to running tSQLt tests: `Invoke-DatabaseTests` and `Export-DatabaseTestResults`.

Invoke-DatabaseTests

This cmdlet runs any tSQLt tests included in the scripts folder, such as static analysis, unit tests or integration tests.

Example

```
$testResults = Invoke-DatabaseTests "C:\Work\scaProject.sqlproj"
```

Don't confuse `Invoke-DatabaseTests` with the `Test-DatabaseConnection` cmdlet that's used for testing a database connection (see Setting up database connections).

Export-DatabaseTestResults

This cmdlet exports the output of the `Invoke-DlmDatabaseTest` cmdlet to disk.

Example

```
Invoke-DatabaseTests "C:\Work\scaProject.sqlproj" | Export-DatabaseTestResults -OutputFile "C:
\Work\TestResults\scripts.junit.xml"
```

In the example above, we've used the pipe (|) symbol to take the output of `Invoke-DatabaseTests` and use it as the input for `Export-DatabaseTestResults`.

Package and publish

After you've validated, tested and synced the latest version of your development schema, you can use SQL Change Automation PowerShell cmdlets to add the validated schema to a database package for later use. If you use a tool with a NuGet feed, such as Octopus Deploy, you can publish the package to this feed.

New-DatabaseBuildArtifact

This cmdlet packages the validated database schema that's produced by the `Invoke-`DatabaseBuild cmdlet.

Example

```
$validatedProject = Invoke-DatabaseBuild "C:\Work\scaProject.sqlproj"
$buildArtifact = New-DatabaseBuildArtifact $validatedProject -PackageId "MyDatabase" -PackageVersion "1.0.0"
```

In the example above, we've used the `New-DatabaseBuildArtifact` cmdlet to create a NuGet package from the output of `Invoke-DatabaseBuild`.

The `PackageID` parameter specifies the unique identifier for the package. You also need to specify the version of the package.

Export-DatabaseBuildArtifact

Use this cmdlet if you want to export the NuGet database package to an output folder.

Example

```
$validatedProject = Invoke-DatabaseBuild "C:\Work\scaProject.sqlproj"
$buildArtifact = New-DatabaseBuildArtifact $validatedProject -PackageId "MyDatabase" -PackageVersion "1.0.0"
Export-DatabaseBuildArtifact $buildArtifact -Path "C:\packages"
```

In the example above, on line 4, the `Export-DatabaseBuildArtifact` cmdlet will produce the file *MyDatabase.1.0.0.nupkg* in the folder *C:\packages*. The file name for database package is produced by combining the `PackageId` and `PackageVersion` parameters of the `New-DatabaseBuildArtifact` cmdlet.

Publish-DatabaseBuildArtifact

Use this cmdlet if you want to publish the database package to the NuGet feed of a release management tool, such as Octopus Deploy.

Example

```
$validatedProject = Invoke-DatabaseBuild "C:\Work\scaProject.sqlproj"
$buildArtifact = New-DatabaseBuildArtifact $validatedProject -PackageId "MyDatabase" -PackageVersion "1.0.0"
Publish-DatabaseBuildArtifact $buildArtifact -NuGetFeedUrl "http://localhost:4000/nuget/packages" -NuGetApiKey
"ed6d7c98-9847-4599-b5a8-323f215b5c89"
```

In the example above, on line 4, we've used the Publish-DatabaseBuildArtifact cmdlet to send the database package to a NuGet feed that requires an API key.

An example script for continuous integration

Let's now combine all the continuous integration tasks we've looked at into a single PowerShell script:

Example

```
# Validate the SQL Change Automation project
$validatedProject = Invoke-DatabaseBuild "C:\Work\scaProject.sqlproj"

# Run tSQLt tests
Invoke-DatabaseTests $validatedProject | Export-DatabaseTestResults -OutputFile "C:\Work\TestResults\scripts.
junit.xml"

# Package and Publish the schema
$databaseBuildArtifact = New-DatabaseBuildArtifact $validatedProject -PackageId "MyDatabase" -PackageVersion
"1.0.0"
Publish-DatabaseBuildArtifact $databaseBuildArtifact -NuGetFeedUrl "http://localhost:4000/nuget/packages" -
NuGetApiKey "ed6d7c98-9847-4599-b5a8-323f215b5c89"
```

In the example above:

- On line 2, we've used `Invoke-DatabaseBuild` to validate the project and assigned the output of this cmdlet to the `$validatedProject` variable.
- On line 5, we've used `Invoke-DatabaseTests` to run tSQLt tests on the validated scripts folder and then used `Export-DatabaseTestResults` to export the results to disk.
- On line 8, we've used `New-DatabaseBuildArtifact` to package the validated schema.
- On line 9, we've used `Publish-DatabaseBuildArtifact` to publish the package to a NuGet feed.

What next?

We've now looked at the full continuous integration process that validates, test, syncs, packages and publishes the schema in a scripts folder. You can use your build server to set up a trigger and build step to run these tasks every time there's a change to the schema in source control. However, that's not the end of using the SQL Change Automation PowerShell module. It also has a set of cmdlets for deploying your database. To find out more, see Automated deployments.

Cmdlet reference

For full details about all the SQL Change Automation cmdlets, see the SQL Change Automation cmdlet reference.

Related Content (how-to article)

SQL Change Automation with PowerShell Scripts: getting up-and-running:

Provides a full PowerShell script that uses the database build cmdlets (`Invoke-DatabaseBuild`, `New-DatabaseBuildArtifact` and `Export-DatabaseBuildArtifact`) to create a validated build artifact, including database documentation, and then uses this artifact to update an existing database to the same version (`Sync-DatabaseSchema`).