We use SQL Server Reporting Services to provide real-time reporting to our clients business units across Australia.

Anyone who has ever done any SSRS development knows that deploying these reports manually is a real pain in the backside - nothing is more painful than trying to upload a large number of files through the clunky SSRS web interface.

It's even worse when you include though having to manually execute database scripts across multiple databases!

The first solution which people usually move to is automating the deployment through Powershell scripts which was also the first step we took, however having to manually run the scripts along with ensuring that the databases were updated with the latest versions of the stored procedures and functions was too tedious.

Complete deployment automation was the goal - we wanted a totally hands free process which we could reliably and repeatably run in all environments. As as we were already using the awesome [Octopus Deploy](http://www.octopusdeploy.com/) to deploy the other applications in our solution it was the logical choice to drive this.

**Packaging**

Octopus Deploy requires a nuget package which contains the artefacts to be deployed which are defined by a nuspec file in the repository.

|  |
| --- |
| <?xml version="1.0"?> |
|  | <package xmlns="http://schemas.microsoft.com/packaging/2010/07/nuspec.xsd"> |
|  | <metadata> |
|  | <id>Reports</id> |
|  | <title>Report PAckage</title> |
|  | <version>1.0.0</version> |
|  | <authors>Ajilon</authors> |
|  | <owners>Ajilon</owners> |
|  | <requireLicenseAcceptance>false</requireLicenseAcceptance> |
|  | <description>Contains all SQL Server Reporting Services reports and scripts required to create a fully functional environment</description> |
|  | </metadata> |
|  | <files> |
|  | <file src="\*.rdl" target="Reports" /> |
|  | <file src="..\sql\\*.sql" target="Sql" /> |
|  | <file src="..\sql\Scripts\\*.sql" target="Sql\Scripts" /> |
|  | <file src="..\sql\Scripts\Dependencies\\*.sql" target="Sql\Scripts\Dependencies" /> |
|  | <file src="\*.ps1" target="" /> |
|  | <file src="..\\*.ps1" target="" /> |
|  | </files> |
|  | </package> |

The deployment process relies on a specific structure to allow, for example, the SQL functions to be deployed before the SQL Stored Procedures which may depend on them

* Reports - contains all SSRS artefacts to upload
* SQL\Scripts - contains the stored procedures to be deployed to each business unit database
* SQL\Dependencies - contains the functions or other SQL artefacts which must be deployed before the contents of the SQL\Scripts directory.

Using our existing TeamCity installation, I created a new build configuration to retrieve the SSRS project from GitHub and build the package according to the nuspec. and trigger a deployment to our *Development Testing* environment.

### TeamCity Build ConfigurationDeployment

The deployment process leverages two PowerShell scripts invoked from the default.ps1 Octopus script.

The first script, [DeployReports.ps1](https://gist.github.com/sbedford/0693ab338a86aed34d98#file-deployreports-ps1) is responsible for deploying the reports and data sources into SSRS. The second script, [DeploySQL.ps1](https://gist.github.com/sbedford/0693ab338a86aed34d98#file-deploysql-ps1) is responsible for pushing database scripts out to ensure that the database version matches the report version.

The Octopus deployment process uses a number of variables to drive the deployment:

* SSRSReportServerUrl - contains the full url to the SSRS web service i.e.<http://server/reportserver/reportservice2005.asmx>
* SSRSReportFolder - the root folder to deploy the reports into (we use the project name)
* SSRSSharedDataSourcePath - the path to a Shared DataSource required by some of our reports. It is expected this is already setup and configured (this is the only manual step)
* SSRSDynamicDataSourceCredentialsUsername - the username to connect to the business unit database with
* SSRSDynamicDataSourceCredentialsPassword - the password for the business unit database user.
* nEDRMappingConnectionStrings - a comma separate string of Oracle ezConnect strings used to connect to SQLPlus\* with.

**Deploying Reports**

Report deployment is fairly straight forward - use the web service endpoint exposed by SSRS to setup the folder structures, upload all reports and finally configure each reports data sources.

As we share a single SSRS server across a few environemnts (DEV and UAT), our deployment process ensures that when we are deploying to the *Development Testing*environment, the reports in the *UAT* environment are not affected.

We do this by using the name of the *Octopus Deploy Environment* we are deploying into and creating the appropriate directory structure off that:

|  |  |
| --- | --- |
|  | $environment = $OctopusParameters["Octopus.Environment.Name"] |
|  |  |
|  | # create proxy object to http://ssrs/reportserver/reportservice2005.asmx |
|  | $ssrsProxy = New-WebServiceProxy -Uri $ssrsreportserverurl -UseDefaultCredential |
|  |  |
|  | # create top level (project) folder |
|  | $ssrsProxy.CreateFolder($SSRSReportFolder, $reportPath, $null) |
|  |  |
|  | # create the environment sub folder. |
|  | $reportPath = [string]::Concat($reportPath, $SSRSReportFolder) |
|  | $ssrsProxy.CreateFolder($environment, $reportPath, $null) |

[**view raw**](https://gist.github.com/sbedford/0693ab338a86aed34d98/raw/ced9e23cea79acc3ddd2a719ae24becb2e8ef796/Report_Folder.ps1)[**Report\_Folder.ps1**](https://gist.github.com/sbedford/0693ab338a86aed34d98#file-report_folder-ps1) hosted with ❤ by [**GitHub**](https://github.com/)

Once the reporting folders have been configured it's time to upload each of our report definition files.

|  |  |
| --- | --- |
|  | Get-ChildItem './Reports' -Filter \*.rdl | |
|  | Foreach-Object{ |
|  |  |
|  | $byteArray = gc $rdlFile -encoding byte |
|  |  |
|  | Write-Host "Uploading $reportName to $reportFolder" |
|  |  |
|  | $ssrsProxy.CreateReport($reportName,$reportFolder,$force,$byteArray,$null) |
|  | } |

[**view raw**](https://gist.github.com/sbedford/0693ab338a86aed34d98/raw/ced9e23cea79acc3ddd2a719ae24becb2e8ef796/Report_Upload.ps1)[**Report\_Upload.ps1**](https://gist.github.com/sbedford/0693ab338a86aed34d98#file-report_upload-ps1) hosted with ❤ by [**GitHub**](https://github.com/)

While you can run the reports now - SSRS will crash and complain about the datasources because

1. Credentials in an embedded data source are not saved when uploading an RDL - also, we're not using the same credentials in development and production environments are we?
2. The link between a Report and a Shared Datasource must be re-established before the report can be run.

To fix this we need to retrieve the report from the server and specifically modify the data sources using variables provided by Octopus Deploy:

|  |  |
| --- | --- |
|  | $dsPath = [string]::Concat($reportFolder, "/", $reportName) |
|  | $ds = $ssrsProxy.GetItemDataSources($dsPath) |
|  | $ds | ForEach-Object { |
|  | # Embedded Datasource |
|  | if($\_.Name -eq "dsDynamic") { |
|  | if ($reportCredentialsUsername -ne "") { |
|  | Write-Host "Datasource dsDynamic found. Setting credentials" |
|  | $\_.Item.CredentialRetrieval = "Store" |
|  | $\_.Item.UserName = $reportCredentialsUsername |
|  | $\_.Item.Password = $reportCredentialsPassword |
|  | $found = 1 |
|  | } |
|  | # Shared Datasource |
|  | } elseif ($\_.Name -eq "dsTAS") { |
|  | if ($SSRSSharedDataSourcePath -ne $null){ |
|  | $proxyNamespace = $ssrsProxy.GetType().Namespace |
|  | $\_.Item = New-Object("$proxyNamespace.DataSourceReference") |
|  | $\_.Item.Reference = $SSRSSharedDataSourcePath |
|  | $found = 1 |
|  | } |
|  | } |
|  | } |
|  | if ($found -eq 1){ |
|  | $ssrsProxy.SetItemDataSources($dsPath, $ds) |
|  | } |

[**view raw**](https://gist.github.com/sbedford/0693ab338a86aed34d98/raw/ced9e23cea79acc3ddd2a719ae24becb2e8ef796/Report_Datasource.ps1)[**Report\_Datasource.ps1**](https://gist.github.com/sbedford/0693ab338a86aed34d98#file-report_datasource-ps1) hosted with ❤ by [**GitHub**](https://github.com/)

**Deploying SQL Procedures and Functions**

The script for deploying our SQL atefacts is responsible for 3 things:

1. Deploy everything in the \SQL\Dependencies folder
2. Deploy everything in the \SQL folder
3. Insert a new record into a VersionInfo table (thanks [FluentMigrator](https://github.com/schambers/fluentmigrator)!) to record the current version of the reports.

As mentioned before, our deployment process deploys to multiple databases in series; this is done by adding a CSV string of SQL Plus commands as an environment specific variable in octopus deploy.

|  |  |
| --- | --- |
|  | function Invoke-SqlPlusWithFile($connectionString, $file) { |
|  | Write-Output "$file"; |
|  |  |
|  | # Wrap script to make sqlplus stop on errors (because Oracle doesn't do this |
|  | # by default). Also exit the prompt after the script has run. |
|  | $lines = Get-Content $file; |
|  | $lines = ,'whenever sqlerror exit sql.sqlcode rollback;' + $lines; |
|  | $lines += 'exit'; |
|  |  |
|  | Invoke-SqlPlusCommand $connectionString $lines |
|  | } |
|  |  |
|  | function Invoke-SqlPlusCommand($connectionString, $command) { |
|  |  |
|  | $command | sqlplus $connectionString | Tee-Object -Variable output | out-default; |
|  |  |
|  | if (!$? -or # Stop on non-zero exit codes. |
|  | # Stop on script errors. Have to detect them from output |
|  | # unfortunately, as I couldn't find a way to make SQL\*Plus halt on |
|  | # warnings. |
|  | $output -match "compilation errors" -or |
|  | $output -match "unknown command" -or |
|  | $output -match "Input is too long" -or |
|  | $output -match "unable to open file") { |
|  | throw "Script failed: $fileNameOnly"; |
|  | } |
|  | } |
|  |  |
|  | $environment = $OctopusParameters["Octopus.Environment.Name"] ; |
|  | $version = $OctopusParameters["Octopus.Release.Number"]; |
|  |  |
|  | $versionCommand = " |
|  | INSERT INTO VersionInfo VALUES ('$environment','$version', sysdate); |
|  | COMMIT; |
|  | /"; |
|  |  |
|  | foreach ($connection in $nEDRMappingConnectionStrings.Split(",")) |
|  | { |
|  | Write-Host "Executing scripts for Connection $connection" |
|  |  |
|  | $files = Get-ChildItem "./SQL/Scripts/Dependencies" -Filter \*.sql | Sort-Object |
|  | foreach ($file in $files) { |
|  | Invoke-SqlPlusWithFile $connection $file.FullName |
|  | } |
|  |  |
|  | $files = Get-ChildItem "./SQL/Scripts" -Filter \*.sql | Sort-Object |
|  | foreach ($file in $files) { |
|  | Invoke-SqlPlusWithFile $connection $file.FullName |
|  | } |
|  |  |
|  | Invoke-SqlPlusCommand $connection $versionCommand |
|  | } |

[**view raw**](https://gist.github.com/sbedford/0693ab338a86aed34d98/raw/ced9e23cea79acc3ddd2a719ae24becb2e8ef796/DeploySQL.ps1)[**DeploySQL.ps1**](https://gist.github.com/sbedford/0693ab338a86aed34d98#file-deploysql-ps1) hosted with ❤ by [**GitHub**](https://github.com/)

(Admittedly, this script is a little clunky and while it works for us, I haven't tested it in any other configurations that how we are using it.)