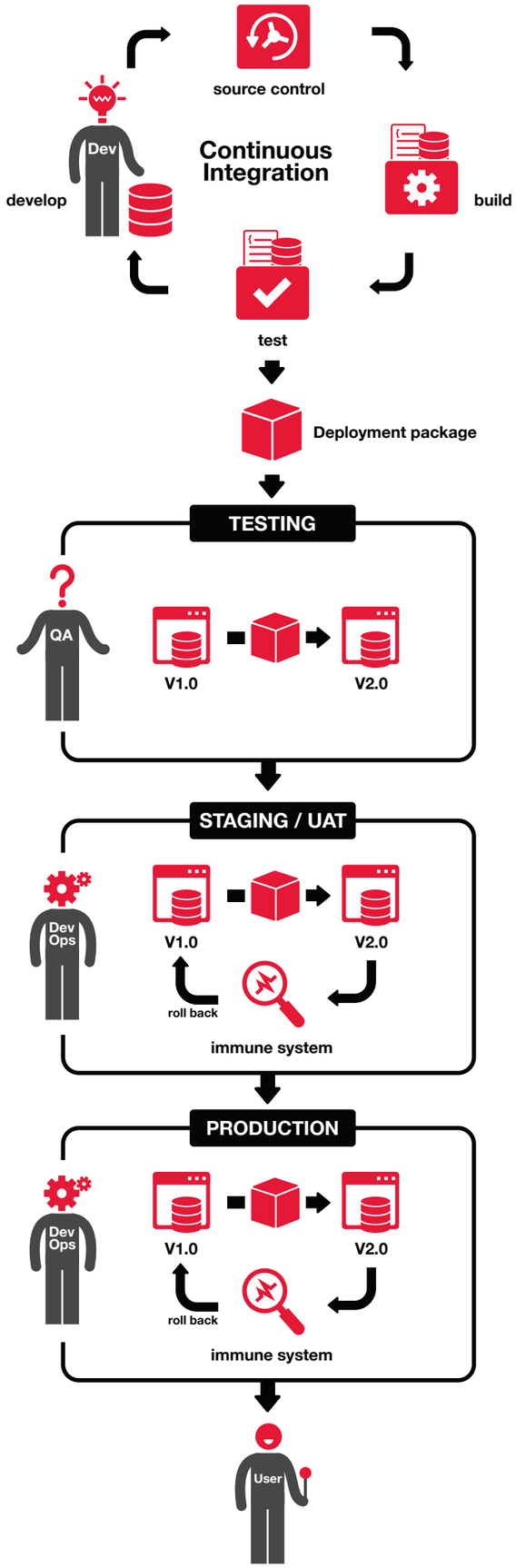
Whitepaper

Automated deployment using Red Gate tools

A technical overview

redgate

ingeniously simple



Automated deployment using Red Gate tools

Introduction

This whitepaper examines the benefits and challenges of automating deployments of applications and databases in a unified way. It describes how Red Gate tools can be used to manage the whole deployment process, leaving you to get on with building the software.

Simple examples using Red Gate Deployment Manager for a typical application are included in this whitepaper. The toolset, architecture, and configuration are explained openly, so that you will be able to apply these ideas to your projects and extend them to support the various technologies your application may use.

Automated deployment is a key part of Red Gate's Continuous Delivery story. There is also a partner whitepaper Continuous integration for databases using Red Gate tools, which explains how to incorporate databases into your build automation process.

This whitepaper covers deployment of the whole application, including ASP.NET websites, dependent assemblies, and databases. It is organized into the following sections:

I. Why automate deployment?

If you're not automating deployments, then you're doing them by hand. There are varying levels of automation, but they all attempt to solve some or all of the problems with manual deployment. Specifically, manual deployments are:

Time consuming. You need to back up your database and application. You need to prepare a script for the database, check it over, and run it in SQL Server Management Studio. You need to copy the latest application binaries and update IIS.

You need to do this for development, testing, user acceptance testing, staging, and production environments. You may also have a number of clients on different versions. That's a lot of work, and it often results in coming into the office out of hours to do deployments.

Error prone. At any one of these stages, a small mistake could have serious consequences - the backup wasn't created correctly, the update script mistakenly dropped a column, a file was missing from the website. Any one of these things can cause big problems for the business and unnecessary headaches for you.

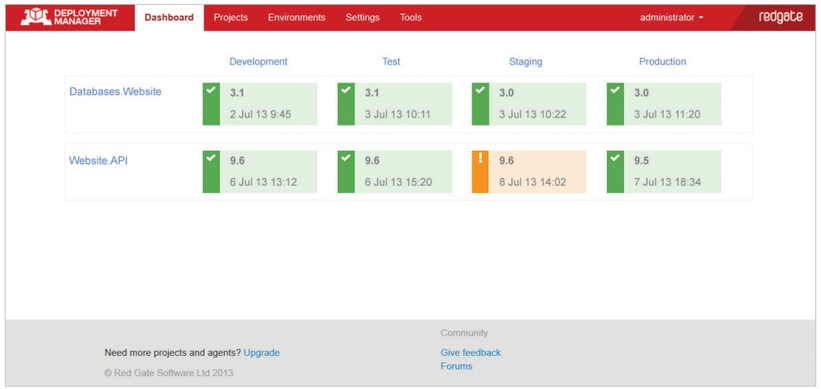
Single point of failure. Typically, few people have access to the production server. The deployment scripts are complicated, and few people know how they work. If those people are unavailable, you may need to delay a release. If problems occur, it could be down to one person to apply the necessary fixes.

Opaque. Which version of our software does customer X have? Which version is in QA? Did the latest version on production go through the appropriate test environments first? Who do I need to talk to about a deployment gone wrong? All of these questions are hard to answer when deploying manually.

An automated deployment process seeks to solve these problems.

II. Introducing Deployment Manager

Deployment Manager is Red Gate's new release management tool. It is designed to make deployments as easy as possible.



Deployment Manager installs two main components, a server and an agent. The server component provides a web UI for running deployments. If you want, the whole team can have access. The agent is installed onto all machines that you wish to deploy to. A secure connection (public/private key cryptography, as used with SSH or HTTPS) is established between the server and the agents.

Using the Deployment Manager interface, you can define the configuration of your projects, using a collection of concepts:

Environment. An environment such as Development, Testing, or Production.

Machine. A physical or virtual machine that you wish to deploy to. These are added to environments. You can deploy to multiple machines per environment. You can also configure a machine to deploy a database to SQL Azure.

Package. Packages are the artifacts that contain your application. A package is created for each component of your application: websites, web services, and databases. These are typically created by a continuous integration server, But they can also be created from Visual Studio and SQL Server Management Studio using the Deployment Manager plug-ins for those development environments. They can also be created via scripts. More detail on packages and the various formats can be found in the next section.

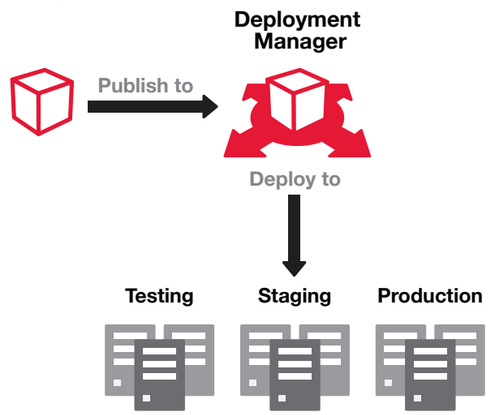
Feed. A package feed is where all of your packages live, a bit like an artifact repository.

Variables. Variables are used to customize the deployment process, including between environments.

Project. A project is a component that you wish to deploy as a single unit, and comprises multiple packages and variables.

III. Package-based deployment

Packages are a key component of Red Gate's Continuous Delivery story. Packages gather together all your changes and apply them to an environment in the correct order. As a result, you don't have to apply each change manually at the appropriate stage of deployment. These packages are created either by a continuous integration system, or from scripts. Once created, they do not change, and are added to a repository of available packages so that they can be accessed by other parts of the build and deployment process.



Red Gate deployment packages are based on the NuGet package format, which presents a number of advantages:

Feeds. Microsoft offers a free NuGet package server, and some continuous integration servers are adding native support for this.

Industry-standard. NuGet is a Microsoft technology that has support from a number of community tools, such as NuGet Package Explorer.

Versioning. NuGet packages contain versioning meta-data.

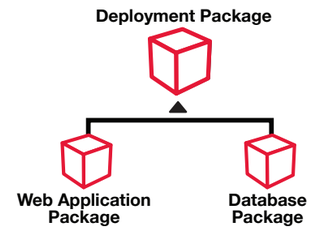
Packages can contain anything you need, and custom deployment logic can be added to any package using PowerShell scripts. On deployment of a package, Deployment Manager will look for and run the following named scripts:

PreDeploy.ps1 Called before deployment.

Deploy.ps1 Called on deployment.

PostDeploy.ps1 Called after a successful deployment.

There are two special types of deployment package supported by Deployment Manager: web application packages and database packages.



There are two command line tools that can be used to create these packages easily, RgPublish.exe and sqlCI.exe. These can be downloaded together, along with DeploymentManager.exe, from: www.red-gate.com/DM

The next two sections will detail the format of these two special package types, how they are created, and how they are deployed.

A. Publishing web application packages to Deployment Manager

Web application packages are intended for deployment of website applications and services. The structure of a typical web application package is shown below:

Web application packages are identified by the presence of a Web.config file. On deployment of a web application package, the target website is reconfigured in IIS to point to the contents of the deployed package. The website to use is identified by the RedGateWebSiteName variable. If none is specified, the package name is used by default.

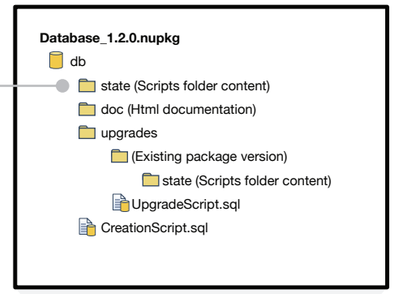
Different Web.config files may needed for each environment. For example, in the production environment the application may connect to a different database, and need a different connection string. You can transform Web.config files on a per environment basis by adding a web config transform file with the environment name in the middle - Web.Production.config, for example.

Deployment Manager variables are configured in the web interface and can be used to customize deployments based on which environment they will be deployed to. For example, you can set the RedGateCreateWebSiteOnPort variable for different environments so that the web application is installed on a different port in production than in testing.

B. Publishing databases to Deployment Manager

Database packages are intended for deployment of SQL Server databases. The structure of a database package is shown below:

Scripts folders are Red Gate format scripts folders, as used by SQL Compare and SQL Source Control. They store the state of the database, including the schema and any static lookup data, as a collection of SQL files.



All items are optional. The minimum needed to deploy a database the state, or a single upgrade script.

On deployment of a database package, the target database is upgraded. There are two types of database upgrade:

Dynamic Upgrade. The upgrade is performed automatically on the agent, using the SQL Compare engine. This is useful for keeping development and testing environments up to date with the minimum amount of effort, as target versions do not have to be specified.

Post-deploy validation can be performed, to ensure the database has the expected state after the upgrade. This is performed using the SQL Compare engine, to ensure there are no differences.

Please note that a valid license for SQL Compare is required on the machine using the SQL Compare engine.

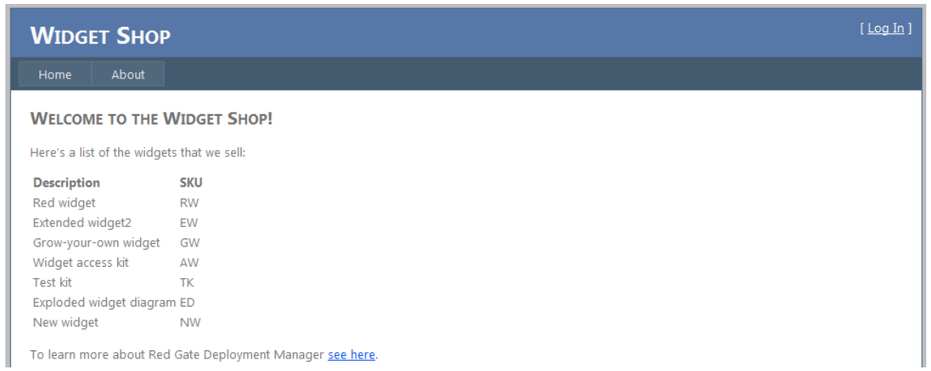
Static Upgrade. Upgrade scripts for specific target versions of the database are generated using the SQL Compare engine and included in the package when it is first created. State can optionally be specified to allow pre-deploy validation, ensuring that the database has not drifted and is in the expected state before deployment. Database drift is often a challenge of reliable deployment, but by using the SQL Compare engine to validate database state, you can ensure upgrade scripts run reliably.

The specific target versions (of the database) to create are specified using the sqlCI.exe tool, when the package is created. This type of upgrade is intended to be used for deploying to staging or production environments. It is intended to verify not just the software, but also the accuracy of the deployment process, through the use of consistent scripts.

IV. Example Deployment Manager use case

A. The application

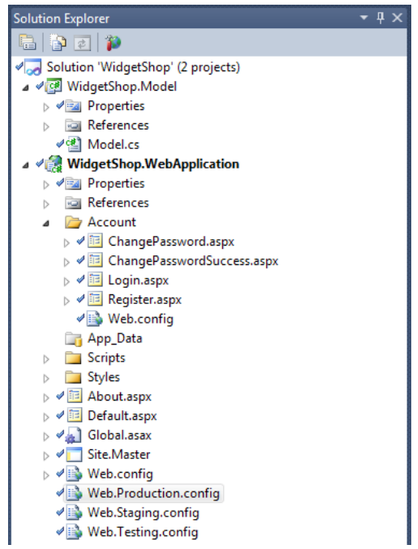
As an example, we've created a simple, multi-tier application that uses ASP.NET and a SQL Server database:

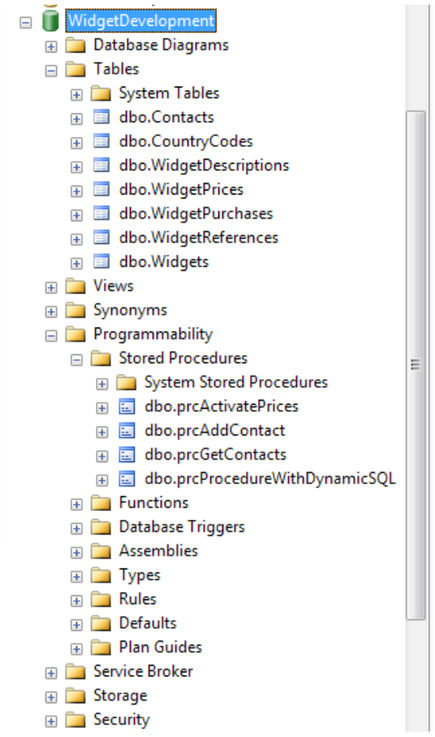


The example application and associated files are available to download from the Deployment Manager page on the Red Gate website: **www.red-gate.com/DM**

The application code is developed in Visual Studio using Subversion for source control. The database is developed in SQL Server Management Studio using SQL Prompt and SQL Source Control linked to Subversion.

Note: We used Subversion, but any source control system could be used.





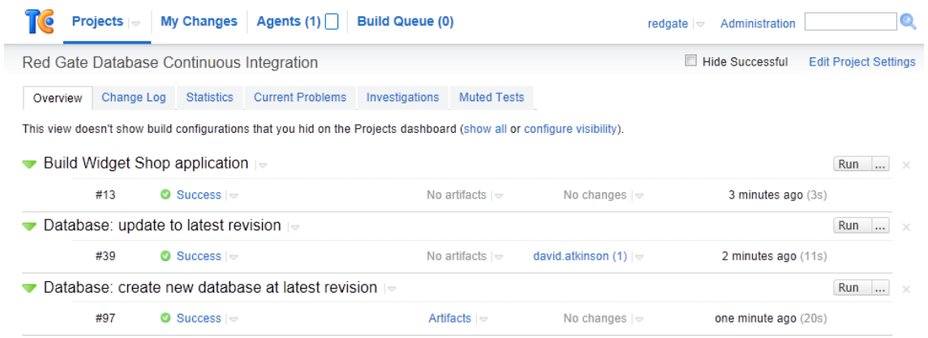
Web config transforms are used for each of the environments to override the database connection strings.

B. Configuration

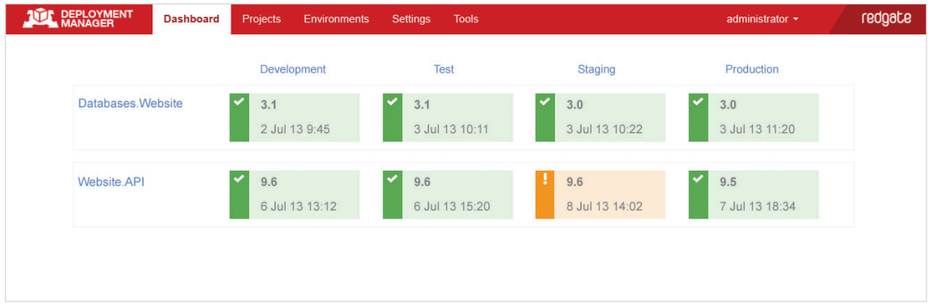
There are four different environments for this application: Development, Testing, Staging, and Production. Each environment comprises a separate website and database. Each developer has their own Development environment on their machine, but there is also a shared Development environment that is kept up to date by the continuous integration system. Testing is a shared environment used by our testing team to test the application. Staging is used to test the deployment to Production, our live environment.

We have configured our continuous integration server, in this case TeamCity by JetBrains, to build our application code and database, and run tests on the application and database. The output of this continuous integration process is a set of packages, one for each component. These packages are automatically published to the package repository after being created. We have also configured our build to keep the shared development environment up to date automatically.

For more detail on configuring continuous integration for databases, see the partner whitepaper, *Continuous integration for databases using Red Gate tools*, available from: **www.red-gate.com/CI**



We have configured our automated deployment tool, Red Gate Deployment Manager, to include the four environments and the two components in one project. This allows us to deploy the application and database together in a single step:

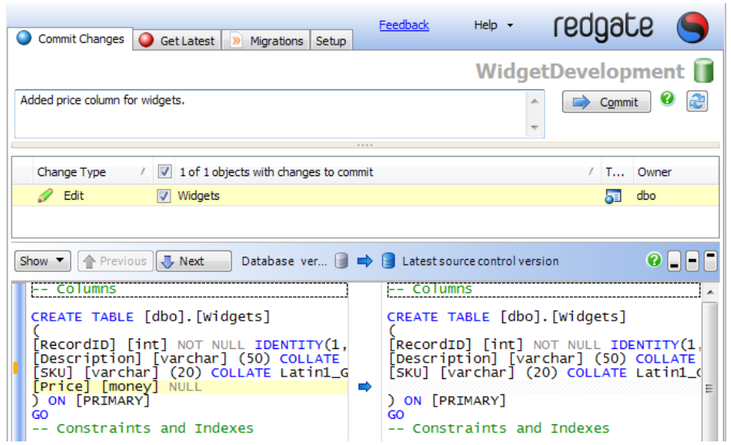


Development and Testing environments are deployed using the dynamic database upgrade type, and Staging and Production are deployed using upgrade scripts. This enables us to have complete convenience when automatically deploying to Development, and to have the consistency we need to be able to test the deployment through to Production.

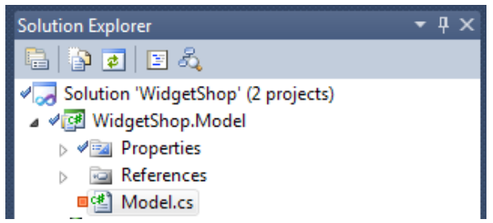
C. Making a change

Once our project has both continuous integration and automated deployment configured, we can start enjoying the benefits of this as we develop our software. Let's add an extra column to the database and update our object model to retrieve it.

First we add the column, and commit that using SQL Source Control:



Next we commit the corresponding change to our application code:



After each commit, our continuous integration process picks up the changes, builds them, tests them, creates a package, and asks Deployment Manager to create a release and deploy it to the shared Development environment. This means the team can always see the latest version of the software, shortly after it is committed.

When the QA team is ready to test the latest version, they can promote the release from Development to Testing using the Deployment Manager UI. Later, when QA have approved the build, the release manager can promote the release from Testing to Staging, and finally to Production.

Different users of Deployment Manager can have different roles. Roles allow you to configure access per user, including which environments a user can deploy to. For example, you may want developers to only be able to deploy to development environments.

This is just one example configuration, from a number of options. For example, the QA team may choose to automatically receive new builds.

V. Command line tools

Database and web application packages can be created using two command line packaging tools. These can be integrated into a continuous integration process, but can also be called from scripts.

A. RgPublish.exe

This tool creates Red Gate deployment packages that can contain anything that you need to deploy. It can be used to create web application packages for .NET web applications and services.

B. sqlCI.exe

This tool performs all required database continuous integration tasks, including creating Red Gate Database Packages for SQL Server databases. These packages represent a specific version of a database, and contain all information needed to upgrade previously released databases. The tool is typically called from a set of NANT scripts, MSBuild scripts or our TeamCity database CI add-in. Please see our database CI pages for more details: **www.red-gate.com/CI**

Example syntax:

```
sqlCI.exe --scriptsFolder=temporary_folder –packageId=WidgetShop
--packageVersion=1.0 –packageRepository=http://dmserver:8080/nuget/
–apiKeyForPackagePublish=AAAAAAAAAAAAAAAAAAAAAAAA
--generateCreationScript
```

Switches:

Switch	Example	Description
--scriptsFolder	--scriptsFolder=WidgetShop\Database\Scriptsfolder	Path to the database scripts folder in source control.
--packageId	--packageId=MyPackage	Package name.
--packageVersion	--packageVersion=$(build_number)	Package build number.
--packageRepository	--packageRepository=http://localhost:8080/nuget/	Package repository URL for publishing and retrieving packages.
--apiKeyForPackagePublish	--apiKeyForPackagePublish=00000000-0000-0000	The API key of the NuGet Package Repository for publishing the package.
--generateCreationScript	--generateCreationScript	Generates a script that will create the source database.

See the documentation for a full list of sqlCI.exe syntax: **www.red-gate.com/CI/switches**

C. DeploymentManager.exe

This tool makes it easy to interact with the Deployment Manager server from the command line. The following commands are available:

Command arguments:

Command	Description
create-release	Creates and (optionally) deploys a release.
deploy-release	Deploys a release.
list-environments	Lists all environments.

Example syntax:

```
DeploymentManager.exe create-release --server=http://dmserver
--project=WidgetShop --deployTo=Development --releaseversion=1.4
--packageVersion=webapp=1.5 --apiKey=AAAAAAAAAAAAAAAAAAAA
AAAAAA
```

Arguments:

The following arguments should be specified:

Argument	Example	Description
--project	WidgetShop	The name of the project in Deployment Manager, to create a release for or deploy.
--server	http://dmserver	The hostname of the machine running Deployment Manager.
--releaseversion	1.4	The version of the release to create.
--packageVersion	1.5	The version of the package to deploy.
--deployTo	Development	The environment to deploy a newly created release to.
--apiKey		The API key used to authenticate calls to the Deployment Manager server.

Note: Detailed help explaining the various arguments required by the commands is available, and can be shown by prefixing the command **help** in front of the command. For example, to see all arguments for the create-release command, enter:

```
DeploymentManager.exe help create-release
```

VI. Further reading and resources

For more detail on setting up continuous integration for databases, visit **www.red-gate.com/CI**

For more detail on Deployment Manager or to download resources such as the example application shown in this whitepaper, visit **www.red-gate.com/DM**

For more information on deploying to SQL Azure, visit **http://thefutureofdeployment.com/how-to-deploy-to-sql-azure-with-deployment-manager/**

Various Red Gate tools have been mentioned in this paper:

SQL Compare

- Compares databases and creates upgrade scripts.

SQL Source Control

- Helps maintain database schema and data in a source control system within SQL Server Management Studio.
- Allows for the creation of custom migration scripts which are saved to source control.

Both these tools are available from **www.red-gate.com/products/sql-development/**

Other recommended background reading:

- The NuGet command line and NuGet repositories.
- Web.config transforms.
- Continuous integration servers.

VII. Conclusions

This whitepaper has explained how it is possible to automate your deployment process using Red Gate tools, saving lots of time and pain, essentially, opening up the deployment process to your team.

We have shown how package-based deployments make for repeatable deployments, and explained how various Red Gate package formats can be used to package the entire application.

A sample multi-tier application was presented. We explained an example configuration and we looked at some of the benefits of using continuous integration and automated deployment, specifically using Red Gate tools.

We hope that this whitepaper has been informative, and that it will help you get started with automating the deployment of your applications.

redgate

ingeniously simple