Database Lifecycle Management

Integrate your Database into your Application Lifecycle Management Process on the Microsoft Stack

Microsoft Tools
SQL Server 2012 R2
SQL Server Management Studio 2012
Visual Studio 2013
Team Foundation Server 2013
Release Management for VS 2013 Update 4

Redgate Tools
SQL Source Control
SQL Automation Pack (SQL Compare, SQL Data Compare, SQL CI, and SQL TFS Plugin)
SQL Test
SQL Release
SQL Monitor

Table of Contents

Introduction

Background

Application developers wouldn't think about writing code or working on a team that wasn't using source control. And, if they're working on a team and doing agile development, then they're probably doing continuous integration as well. These good practices are part of the overall Application Lifecycle Management process, and we want to leverage these for your database so that you can deploy you database changes safely, quickly and efficiently.

In this scenario, our team is using TFS for source control and Team Build for Continuous Integration. Let's get your database into the development process.

If you have any questions/queries/ideas, please email us at dlm@red-gate.com.

Prerequisites

This lab is designed so that you can follow along with each exercise on a VS ALM VM that Microsoft provides. The VS ALM VM is also known as the Brian Keller VM. We'll refer to it as the BK VM or just VM throughout this document.

The BK VM comes pre-loaded with TFS, Visual Studio, Release Management, and a variety of other goodies to show off Microsoft ALM technology. It also comes with two demo applications: TailSpinToys and FabrikamFiber. We'll be using FabrikamFiber in this lab.

So to get started you will need to:

1) Run Hyper-V Manager – I'm using v 6.3.9600
2) Download the BK VM from Brian Keller's blog . It's a 26Gb download, which unzips to 80Gb.
3) Start up the BK VM in Hyper-V manager.
4) Download and install the Redgate SQL Developer Bundle.
5) Using Hyper-V Manager, checkpoint the VM so you can get back to this fresh state if needed.

Make sure you're able to access the VM through remote desktop. Log on details:

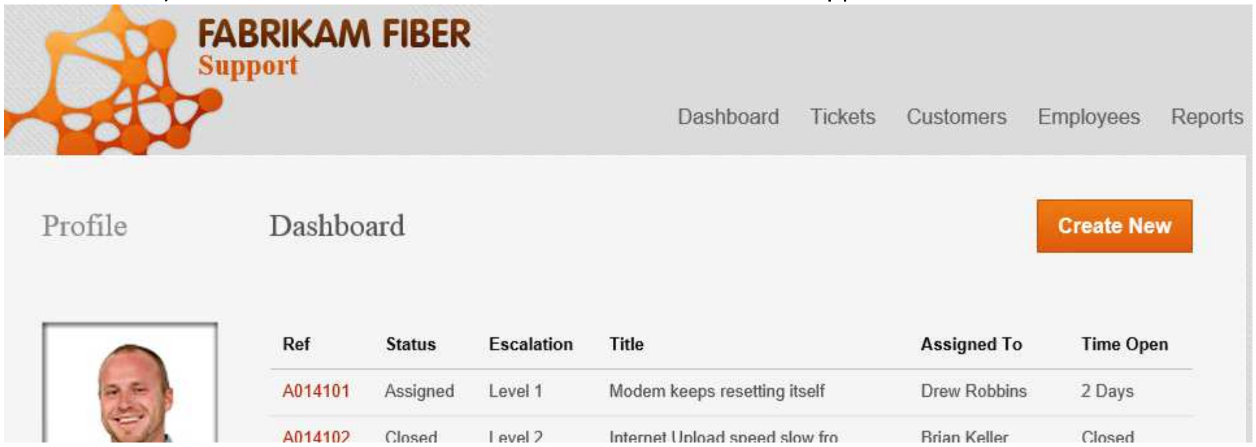
Username: Brian
Password: P2ssw0rd

Please use the Brian account for this lab. Using another account may cause permission issues and distract from the main purpose of the lab. The password P2ssw0rd is used across the VM.

It may also be useful to have a look at the FabrikamFiber application we will be using. It is a web application for customer support of an ISP. It is an ASP.NET web application, backed by a SQL Server database. It is source-controlled in TFS, and builds in TFS. To take a look at it:

1) Open Visual Studio 2013.
2) Open the FabrikamFiber.CallCenter solution. (`C:\Users\Administrator\Source\Workspaces\FabrikamFiber\Dev\FabrikamFiber.CallCenter\FabrikamFiber.CallCenter.sln`) under **Recent** on the **Start Page.**

3) Open **Team Explorer** and **Source Control Explorer,** to see the files in source control.
4) You can run the solution to see the FabrikamFiber Support dashboard in IE. It looks like this:



Ref	Status	Escalation	Title	Assigned To	Time Open
A014101	Assigned	Level 1	Modem keeps resetting itself	Drew Robbins	2 Days
A014102	Closed	Level 2	Internet Upload speed slow fro	Brian Keller	Closed

The document Embracing Continuous Delivery with Release Management for Visual Studio 2013 contains more details about this application.

Recommendations

2 monitors: 1 for these instructions and 1 to run the BK VM in full screen mode so you can follow along.

Disclaimer

Produced as of 28th February 2015. Microsoft and Redgate release frequently, so things may be a little out of date or there may be a better way to do things. If you find any problems, please get in touch with us at dlm@red-gate.com.

The BK VM expires in 10 days if you don't activate it, and 180 days if you do. (See "Working with…" document for details about activation.) Redgate tools expire after 28 days (14 days for some products). If you need more time, please contact dlm@red-gate.com to extend your trial.

All the work in the following lab will be done in the BK VM because it has all the software we need already installed on it. For our purposes, our QA, Test, Staging, and Production environments are all on the same machine. This is a little easier because we don't have to worry about network communications, firewalls, extra VMs, etc. The purpose of this lab is to familiarize you with the tools and how you can use them to make your database development easier. Once you decide to go with our tools, then you can build the infrastructure as you require. Every environment is unique, and the best practices will depend on your situation.

FAQs

Help! I'm lost!

No problem. Just email us at dlm@red-gate.com, and we'll help you out.

But I don't want to use the Brian Keller VM!

That's fine. Most of this lab can be performed on your own infrastructure. Just make sure you have at least:

a) Microsoft SQL Server 2012 with Management Studio.
b) Microsoft Team Foundation source control (aka TFS).
c) Microsoft Team Foundation Build server (aka TF Build).
d) Microsoft Visual Studio 2013 Update 3.
e) Microsoft Release Management (aka MS RM) for Visual Studio 2013 Update 3.

Earlier and later versions of these will probably work too, but I can't guarantee it.

You will also need:

1) A Microsoft SQL Server database you want to develop and deploy.
2) (Optional) An application that uses this database.

I already have MS RM set-up. What's the minimal work I need to do to add a DB to my system?

85% of the content is still relevant. You will still need to:

1) Add your DB to source-control (exercise 1).
2) Add your DB to continuous-integration (exercise 2).
3) (Optionally) Unit-test your database (exercise 3).
4) Add SQL Release to MS RM (Microsoft Release Management) (exercises 5, 6, 7).
5) Add the necessary components and actions to your release path.

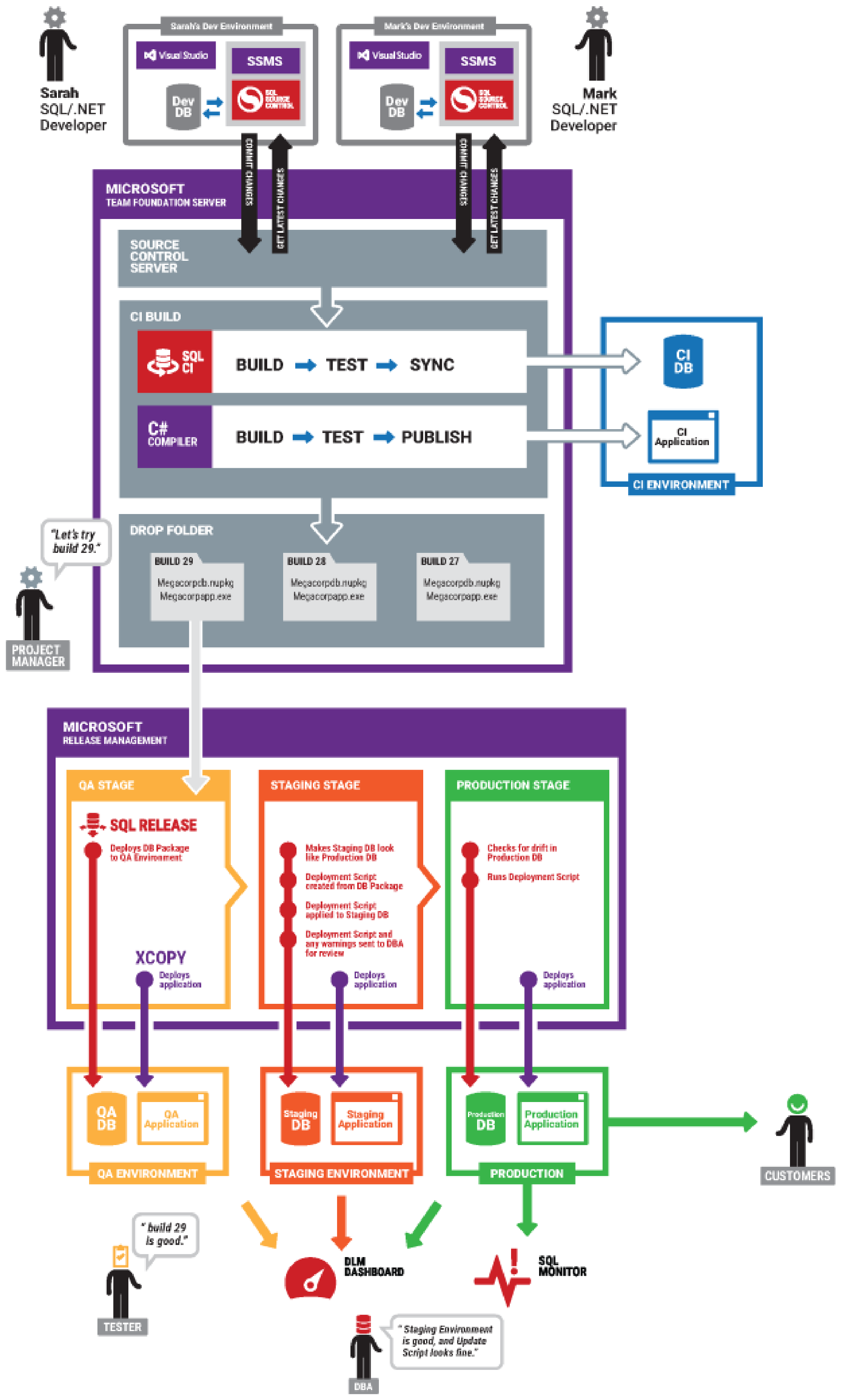
You will also need to ensure the correct DB is referenced from each of your environments. This will probably involve rewriting a web.config using variable replacement (see exercise 8).

System Overview

We will create an ALM/DLM (application lifecycle management/database lifecycle management) system. In this lab, we are most concerned with the database aspects of the release. Briefly, the lifecycle is:

1) Code will be checked-into TFS Source Control.
2) This will trigger a build in TFS Team Build.
3) TFS Team Build produces a build, consisting of both a web-app and a database package.
4) When confirmed, this build enters the MS RM (Microsoft Release Management) pipeline:
    a) The build is released to a **QA** environment.
    b) Testers verify this build, and sign-off on build quality.
    c) The build is then released to a **Staging** Environment.
    d) A DBA checks the Staging environment, and reviews the release artifacts. He then signs-off on the quality, and permits a production release.
    e) The build is released to P**roduction**. The DBA checks the production environment is OK.

This diagram demonstrates the system:


Sarah SQL/.NET Developer
Sarah's Dev Environment
Visual Studio
SSMS
Dev DB
SQL SOURCE CONTROL
Mark's Dev Environment
Visual Studio
SSMS
Dev DB
SQL SOURCE CONTROL
Mark SQL/.NET Developer
COMMIT CHANGES
GET LATEST CHANGES
COMMIT CHANGES
GET LATEST CHANGES
MICROSOFT TEAM FOUNDATION SERVER
SOURCE CONTROL SERVER
CI BUILD
SQL CI
BUILD → TEST → SYNC
C# COMPILER
BUILD → TEST → PUBLISH
CI DB
CI Application
CI ENVIRONMENT
DROP FOLDER
BUILD 29 Megacorpdb.nupkg Megacorpapp.exe
BUILD 28 Megacorpdb.nupkg Megacorpapp.exe
BUILD 27 Megacorpdb.nupkg Megacorpapp.exe
"Let's try build 29."
PROJECT MANAGER
MICROSOFT RELEASE MANAGEMENT
QA STAGE
SQL RELEASE
Deploys DB Package to QA Environment
XCOPY
Deploys application
STAGING STAGE
Makes Staging DB look like Production DB
Deployment Script created from DB Package
Deployment Script applied to Staging DB
Deployment Script and any warnings sent to DBA for review
Deploys application
PRODUCTION STAGE
Checks for drift in Production DB
Runs Deployment Script
Deploys application
QA DB
QA Application
QA ENVIRONMENT
Staging DB
Staging Application
STAGING ENVIRONMENT
Production DB
Production Application
PRODUCTION
CUSTOMERS
"build 29 is good."
TESTER
DLM DASHBOARD
SQL MONITOR
"Staging Environment is good, and Update Script looks fine."
DBA

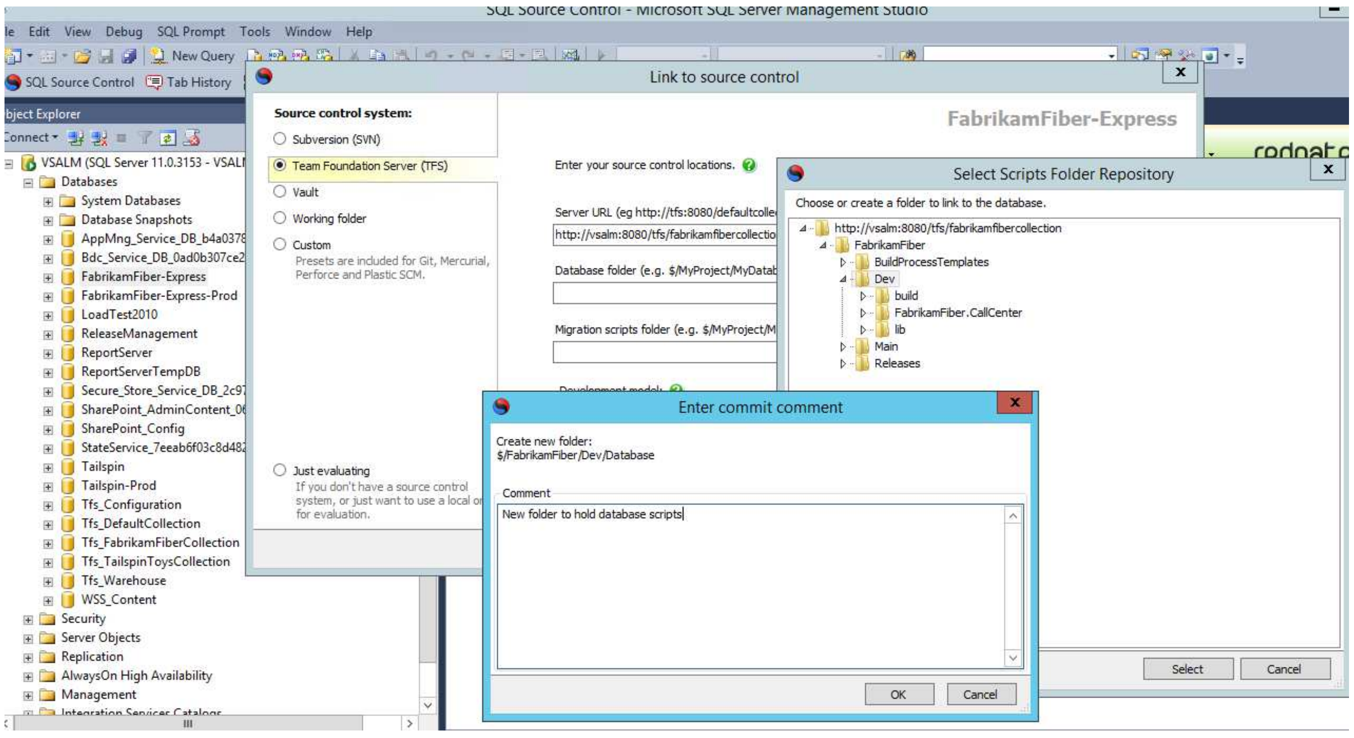
Exercise 1 – Source Controlling your Database

Part 1: Linking a Database to source-control

1) Open **SSMS** 2012.
2) Click **No thanks** to the RG Quality Improvement Program.
3) Connect to **localhost** using **windows authentication**.
4) Right click on the **FabrikamFiber-Express** database in the Object Explorer, and select **Link to source control…**

> **Best Practice** – Only link development databases to source control. Do not source control your production environment. Changes should be made in Dev, checked into source control, and then deployed to other environments per your process.

5) Select **Team Foundation Server (TFS)** from the list of **Source control systems** on the left of the dialog.
6) The Server URL should default to http://vsalm:8080/tfs/defaultcollection. Change **defaultcollection** at the end to **fabrikamfibercollection**, so the URL is http://vsalm:8080/tfs/fabrikamfibercollection.
7) Click **Browse** for the Database folder.
8) Create a new folder called **Database** under FabrikamFiber/Dev and provide a comment (e.g. “New folder to hold database scripts”). It should look like the below. Click **OK** to the comment, **OK** to the confirmation dialog, and **Select** to confirm the Select Scripts Folder Repository dialog.
9) **Migration scripts folder** should be left blank. **Development model** should be left as ‘Dedicated database’. Now click **Link** to link the database to source-control.


SQL Source Control - Microsoft SQL Server Management Studio
le Edit View Debug SQL Prompt Tools Window Help
New Query
SQL Source Control
Tab History
bject Explorer
Connect
VSALM (SQL Server 11.0.3153 - VSALI
Databases
System Databases
Database Snapshots
AppMng_Service_DB_b4a0378
Bdc_Service_DB_0ad0b307ce2
FabrikamFiber-Express
FabrikamFiber-Express-Prod
LoadTest2010
ReleaseManagement
ReportServer
ReportServerTempDB
Secure_Store_Service_DB_2c97
SharePoint_AdminContent_0
SharePoint_Config
StateService_7eeab6f03c8d48
Tailspin
Tailspin-Prod
Tfs_Configuration
Tfs_DefaultCollection
Tfs_FabrikamFiberCollection
Tfs_TailspinToysCollection
Tfs_Warehouse
WSS_Content
Security
Server Objects
Replication
AlwaysOn High Availability
Management
Integration Services Catalogs
Link to source control
FabrikamFiber-Express
Source control system:
Subversion (SVN)
Vault
Working folder
Custom
Presets are included for Git, Mercurial, Perforce and Plastic SCM.
Just evaluating
If you don't have a source control system, or just want to use a local or for evaluation.
Enter your source control locations.
Server URL (eg http://tfs:8080/defaultcolle
Database folder (e.g. $/MyProject/MyDatab
Migration scripts folder (e.g. $/MyProject/M
Development model
Select Scripts Folder Repository
Choose or create a folder to link to the database.
http://vsalm:8080/tfs/fabrikamfibercollection
FabrikamFiber
BuildProcessTemplates
Dev
build
FabrikamFiber.CallCenter
lib
Main
Releases
Select
Cancel
Enter commit comment
Comment
OK
Cancel

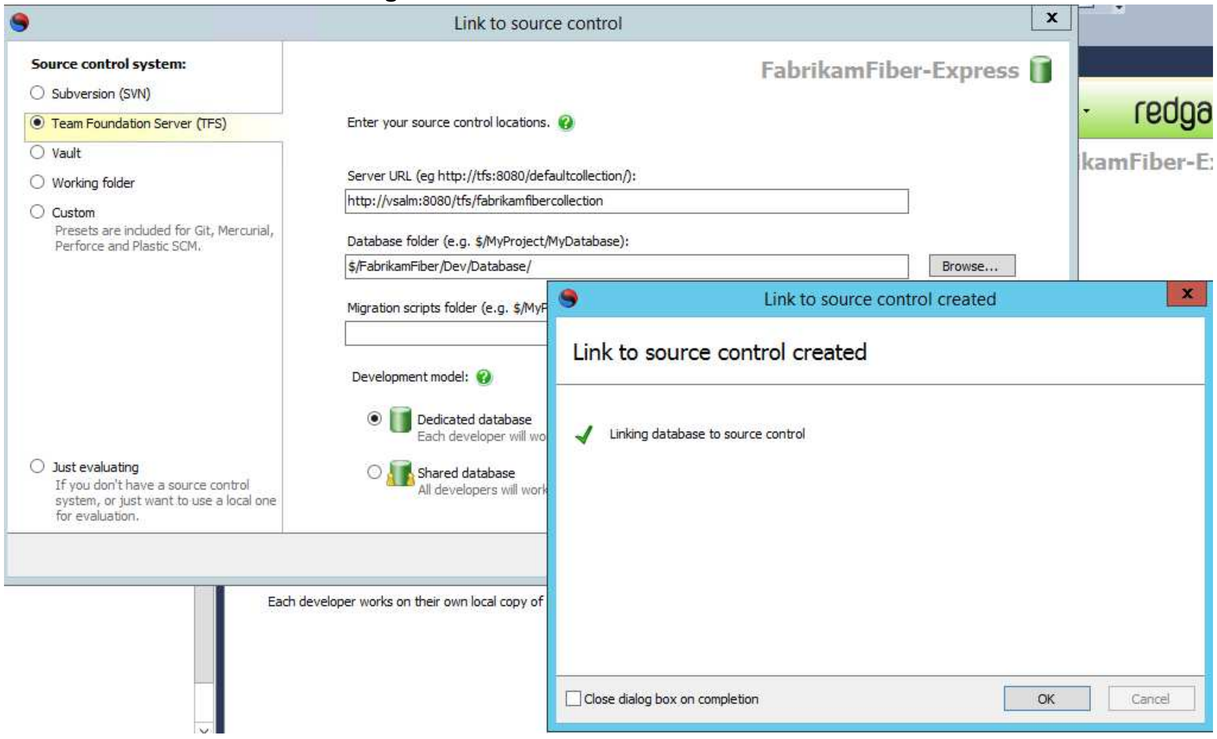
Source control system: (●) Team Foundation Server (TFS)

Server URL: http://vsalm:8080/tfs/fabrikamfibercollectio

Create new folder:
$/FabrikamFiber/Dev/Database

Comment: New folder to hold database scripts

10) You will now see a success dialog. Click **OK** to continue.



Link to source control
Source control system:
Subversion (SVN)
Team Foundation Server (TFS)
Vault
Working folder
Custom
Presets are included for Git, Mercurial, Perforce and Plastic SCM.
Just evaluating
If you don't have a source control system, or just want to use a local one for evaluation.
FabrikamFiber-Express
Enter your source control locations.
Browse...
Development model:
Dedicated database
Shared database
Link to source control created
Close dialog box on completion
OK
Cancel

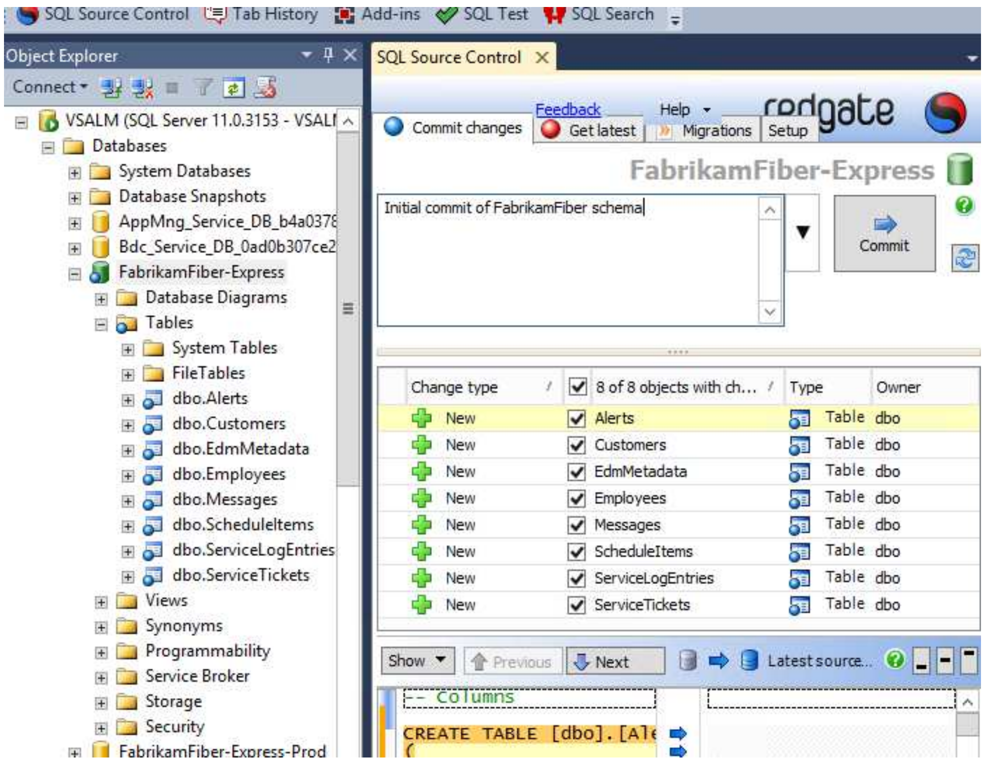
Server URL (eg http://tfs:8080/defaultcollection/): http://vsalm:8080/tfs/fabrikamfibercollection

Database folder (e.g. $/MyProject/MyDatabase): $/FabrikamFiber/Dev/Database/

Link to source control created

✓ Linking database to source control

11) On the Commit changes tab, SQL Source Control should now list all the schema objects that make up the FabrikamFiber database. Note the blue circles – these indicate that the objects have changes yet to be committed to TFS. Enter a commit comment, for instance 'Initial commit of FabrikamFiber schema' and click **Commit.**

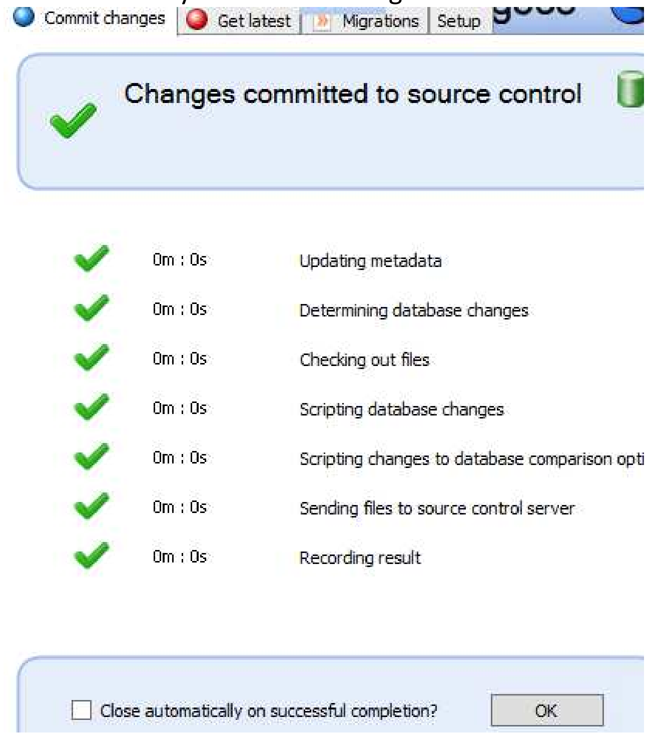


Initial commit of FabrikamFiber schema

Change type	8 of 8 objects with ch...	Type	Owner
New	✓ Alerts	Table	dbo
New	✓ Customers	Table	dbo
New	✓ EdmMetadata	Table	dbo
New	✓ Employees	Table	dbo
New	✓ Messages	Table	dbo
New	✓ ScheduleItems	Table	dbo
New	✓ ServiceLogEntries	Table	dbo
New	✓ ServiceTickets	Table	dbo

```
-- Columns
CREATE TABLE [dbo].[Ale
(
```

12) It will now tell you that the changes were committed successfully. Click **OK** to continue.

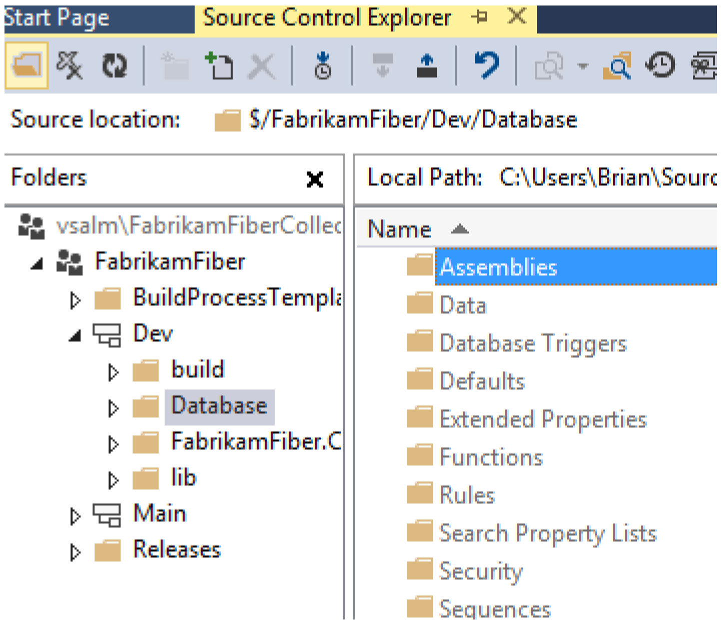


Changes committed to source control

✓	0m : 0s	Updating metadata
✓	0m : 0s	Determining database changes
✓	0m : 0s	Checking out files
✓	0m : 0s	Scripting database changes
✓	0m : 0s	Scripting changes to database comparison opti
✓	0m : 0s	Sending files to source control server
✓	0m : 0s	Recording result

☐ Close automatically on successful completion?

13) Note that the blue circles in the Object Explorer have disappeared.
14) If you look at the Source Control Explorer in Visual Studio, you will now see the schema in TFS source-control:



Source location: $/FabrikamFiber/Dev/Database

Local Path: C:\Users\Brian\Sourc

```
vsalm\FabrikamFiberCollec
  FabrikamFiber
    BuildProcessTempl
    Dev
      build
      Database
      FabrikamFiber.C
      lib
    Main
    Releases
```

```
Assemblies
Data
Database Triggers
Defaults
Extended Properties
Functions
Rules
Search Property Lists
Security
Sequences
```

Exercise 2 – Getting your Database into the Continuous Integration Process

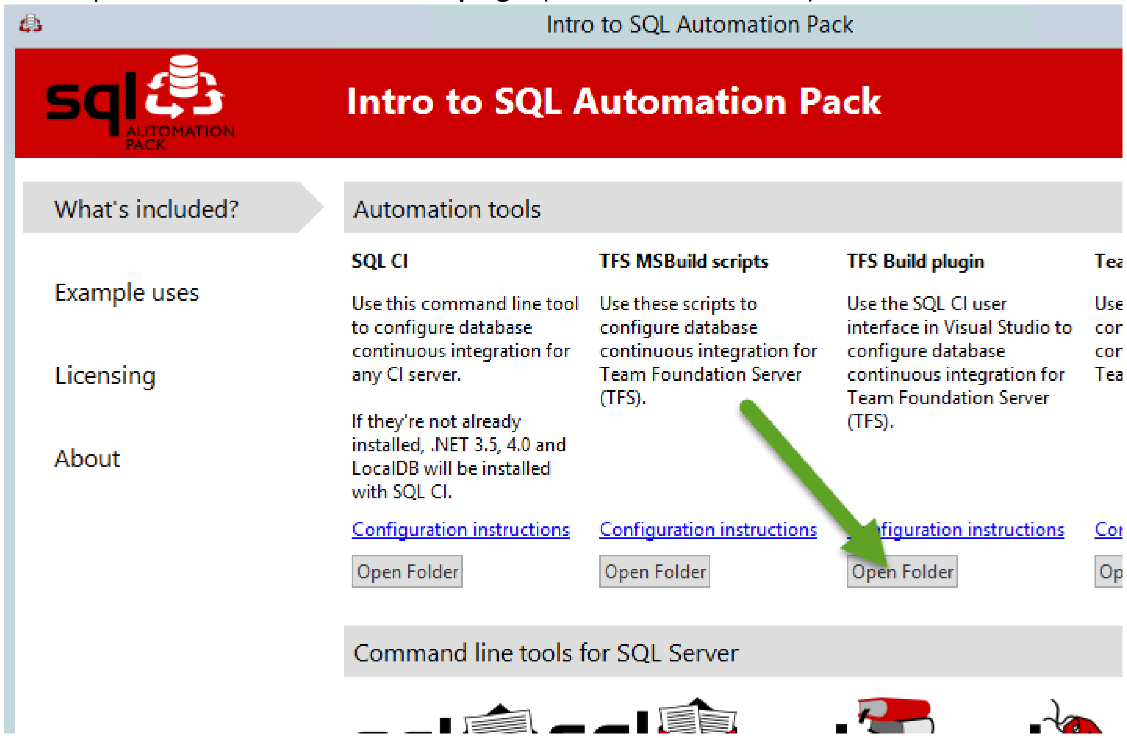
To setup Continuous Integration (CI), your database must be in source-control. This was done in Exercise 1. If you have not done Exercise 1, please do it now.

Application Developers have been using CI for years. We want to add your database to this process. CI is important: you get fast-feedback on commits, a repository of builds corresponding to commits, fast availability of changes for testers, and many other benefits.

To setup database CI, we will be using TFS Team Build and SQL CI, a Redgate tool which is part of the SQL Automation Pack.

Part 1: Installing the SQL Automation Pack

1) Download and install the SQL Automation Pack.
   Remember – This is not licensed so you only have a 28 day free trial. If you need more time, contact dlm@red-gate.com for an extension.
2) Install SQL CI
3) Install the Team Build Plugin
   a) Close Visual Studio.
   b) On the Windows Start Screen, open **Intro to SQL Automation Pack 1.**
   c) Click 'Open Folder' beneath **TFS Build plugin** (see screenshot below).



Intro to SQL Automation Pack

Intro to SQL Automation Pack

What's included?

Example uses

Licensing

About

Automation tools

SQL CI
Use this command line tool to configure database continuous integration for any CI server.

If they're not already installed, .NET 3.5, 4.0 and LocalDB will be installed with SQL CI.

Configuration instructions
Open Folder

TFS MSBuild scripts
Use these scripts to configure database continuous integration for Team Foundation Server (TFS).

Configuration instructions
Open Folder

TFS Build plugin
Use the SQL CI user interface in Visual Studio to configure database continuous integration for Team Foundation Server (TFS).

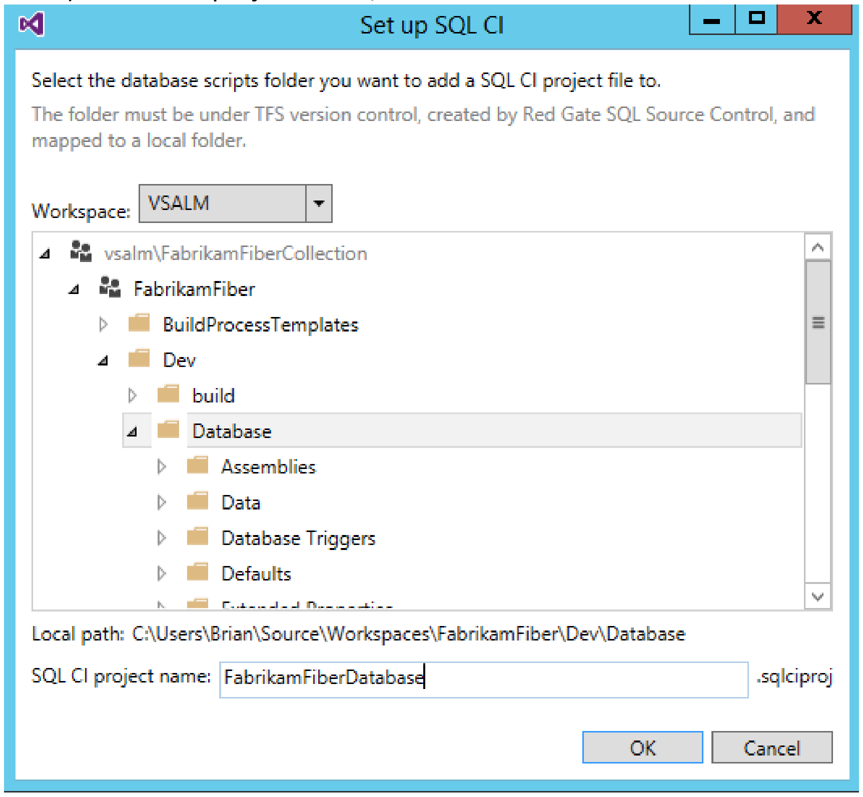
Configuration instructions
Open Folder

Command line tools for SQL Server

   d) Double-click on **TFSBuildPlugin.vsix** to install it.
   e) Reopen Visual Studio 2013 and the FabrikamFiber solution.

Part 2 – Setting up Database CI

1) In the Visual Studio top menu, select **SQL CI** > **Set up SQL CI…**
2) Navigate to **FabrikamFiber\Dev\Database.**
3) In SQL CI project name, set it to **FabrikamFiberDatabase**.



Select the database scripts folder you want to add a SQL CI project file to.

The folder must be under TFS version control, created by Red Gate SQL Source Control, and mapped to a local folder.

Workspace: VSALM

Local path: C:\Users\Brian\Source\Workspaces\FabrikamFiber\Dev\Database

SQL CI project name: FabrikamFiberDatabase .sqlciproj

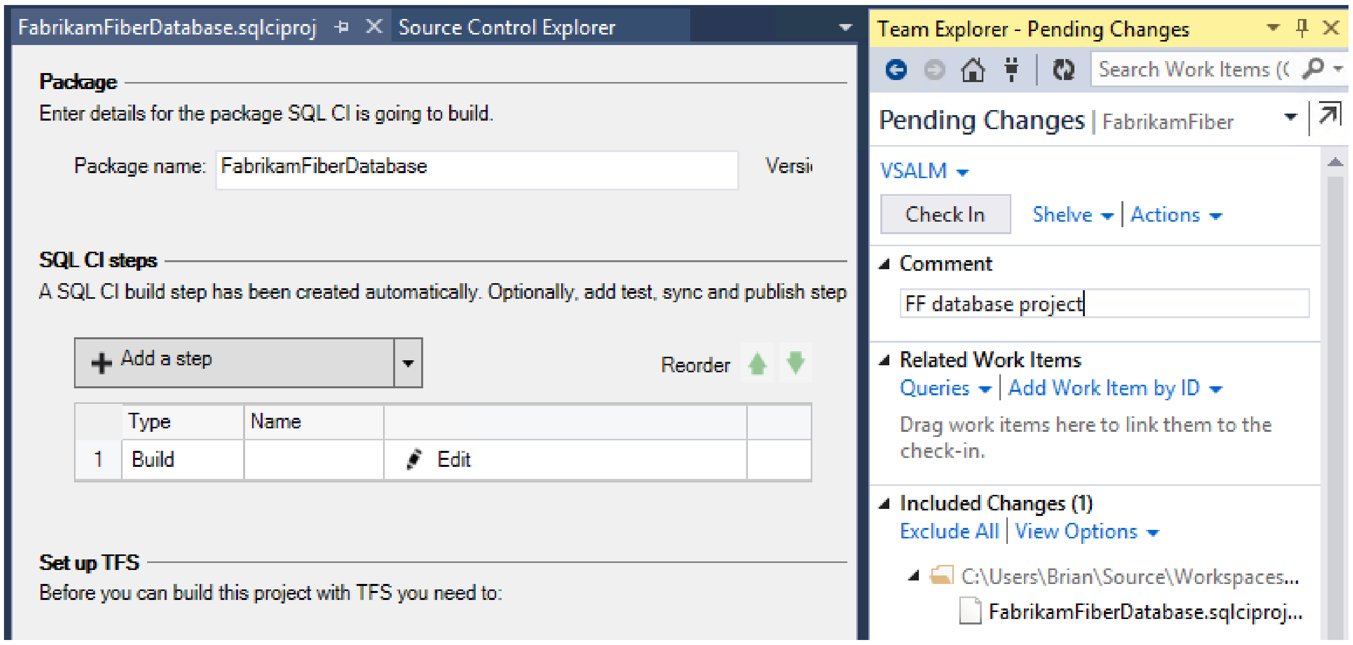
4) Click **OK.**

You will now see the project settings of your database project. This includes things such as the Package name, and what steps are involved in the build. Note a 'build' step is already created for you.

This build step will build a Nuget package from your DB scripts. This Nuget package is a single file (called something like FabrikamFiberDatabase.1.0.0.23.nupkg), which contains everything that defines your database.

When building this Nuget package, it checks that the DB creation scripts are correct, checking syntax, constraints, and dependencies by running the scripts against a new temporary database.

5) In **Team Explorer**, under **Pending Changes**, we need to check in FabrikamFiberDatabase.sqlciproj. Add a comment, such as 'FabrikamFiber database project', then click **Check In**.


FabrikamFiberDatabase.sqlciproj Source Control Explorer

Package

Enter details for the package SQL CI is going to build.

Package name: FabrikamFiberDatabase

SQL CI steps

A SQL CI build step has been created automatically. Optionally, add test, sync and publish step

	Type	Name	
1	Build		Edit

Set up TFS

Before you can build this project with TFS you need to:

Team Explorer - Pending Changes

Pending Changes | FabrikamFiber

VSALM

Comment: FF database project

Related Work Items

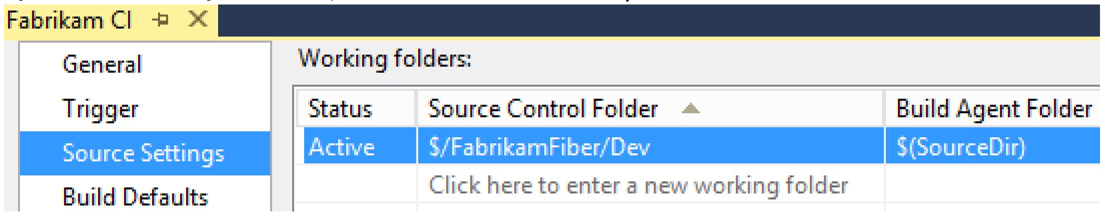
Drag work items here to link them to the check-in.

Included Changes (1)

C:\Users\Brian\Source\Workspaces...

FabrikamFiberDatabase.sqlciproj...

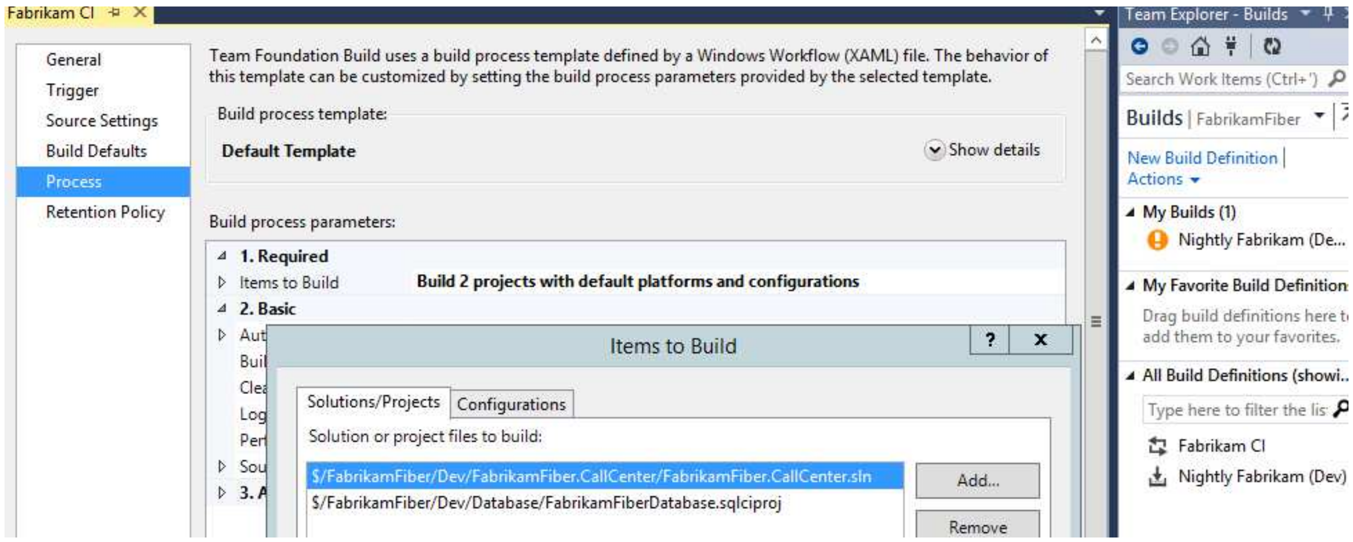
6) We now need to add a TFS Build Definition that will do a CI build of the application and database.
   a) In **Team Explorer**, under **Builds**, click **New Build Definition.** A new build definition window will now appear.
   b) In **General**, enter **Fabrikam CI** as the Build definition name.
   c) In **Trigger**, choose **Continuous Integration.**
   d) In **Source Settings**, change the source control folder from the default **$/FabrikamFiber** to **$/FabrikamFiber/Dev** Also, delete the cloaked entry:


Fabrikam CI

Working folders:

Status	Source Control Folder	Build Agent Folder
Active	$/FabrikamFiber/Dev	$(SourceDir)
	Click here to enter a new working folder	

   e) In **Build Defaults**, under **Staging location**, under **Copy build output to the following drop folder**, enter \\vsalm\ffdrops . Note: this folder may not be accessible by default – you may need to give Brian.Keller access.
   f) In **Process**, under **1. Required** -> **Items to build**, click the ellipsis (…). In the **Items to Build** dialog, click **Add…**, and choose $/FabrikamFiber/Dev/Database/FabrikamFiberDatabase.sqlciproj. Click **Add** again, and choose $/FabrikamFiber/Dev/FabrikamFiber.CallCenter/FabrikamFiber.CallCenter.sln Click **OK**.
   g) Save the build definition by pressing **CTRL + S.**



Fabrikam CI

General
Trigger
Source Settings
Build Defaults
Process
Retention Policy

Team Foundation Build uses a build process template defined by a Windows Workflow (XAML) file. The behavior of this template can be customized by setting the build process parameters provided by the selected template.

Build process template:

Default Template Show details

Build process parameters:

- **1. Required**
  - Items to Build — **Build 2 projects with default platforms and configurations**
- **2. Basic**
  - Aut
  - Buil
  - Clea
  - Log
  - Perf
  - Sou
- **3. A**

Items to Build

Solutions/Projects | Configurations

Solution or project files to build:

$/FabrikamFiber/Dev/FabrikamFiber.CallCenter/FabrikamFiber.CallCenter.sln
$/FabrikamFiber/Dev/Database/FabrikamFiberDatabase.sqlciproj

Add... Remove

Team Explorer - Builds

Search Work Items (Ctrl+')

Builds | FabrikamFiber

New Build Definition | Actions

- My Builds (1)
  - Nightly Fabrikam (De...
- My Favorite Build Definition
  - Drag build definitions here t add them to your favorites.
- All Build Definitions (showi...
  - Type here to filter the lis
  - Fabrikam CI
  - Nightly Fabrikam (Dev)

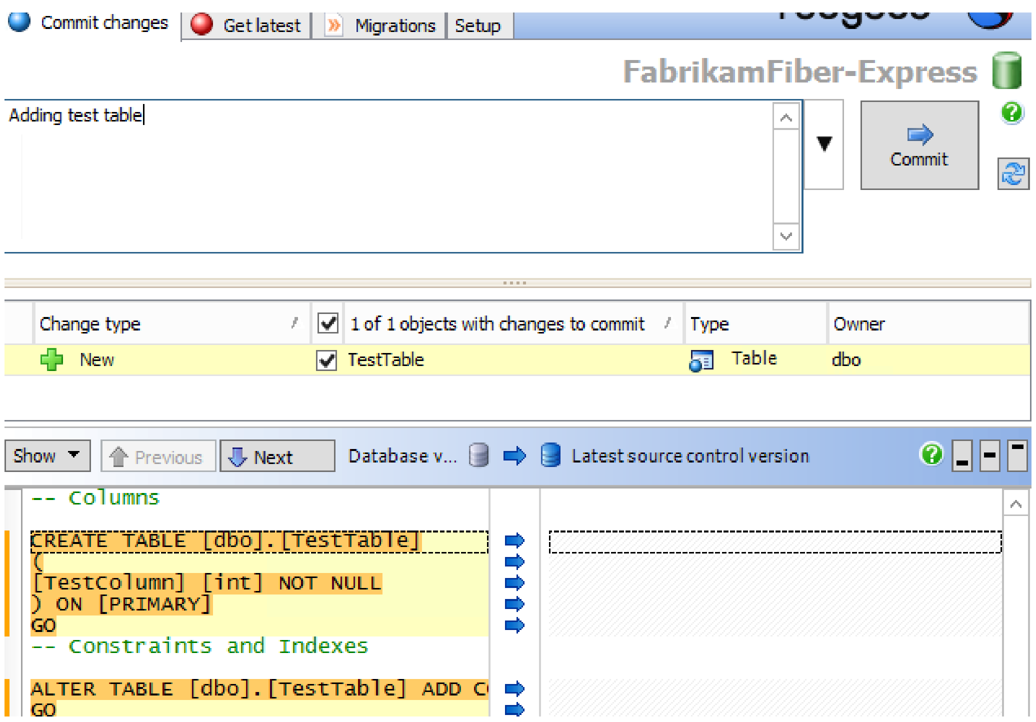
You should now see the Build Definition in Team Explorer - Builds. It will also be visible on the TFS website at http://vsalm:8080/tfs/FabrikamFiberCollection/FabrikamFiber/_build

Step 3 – Try it!

1) Create a table in the FabrikamFiber-Express database in SSMS. For instance, run this T-SQL:

```
CREATE TABLE TestTable ( TestColumn INT PRIMARY KEY )
```

2) Check this in with Redgate Source Control. To do this:
   a) Right click the **FabrikamFiber-Express** database in the Object Explorer and select **Commit changes to source control..**.
   b) You may need to click the Refresh Icon so SQL Source Control can see your latest changes.
   c) It should show you one change – the new TestTable.
   d) Click **Commit** to commit this. You may wish to enter a comment, e.g. 'Adding TestTable':

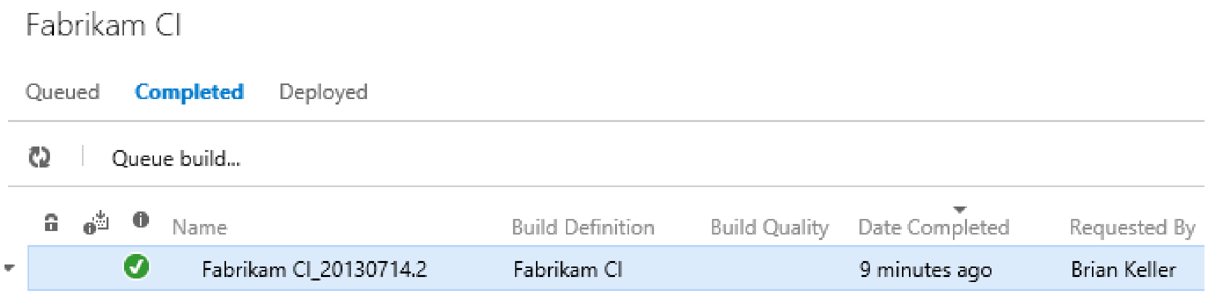


Adding test table

Change type	1 of 1 objects with changes to commit	Type	Owner
New	✓ TestTable	Table	dbo

```
-- Columns
CREATE TABLE [dbo].[TestTable]
(
[TestColumn] [int] NOT NULL
) ON [PRIMARY]
GO
-- Constraints and Indexes
ALTER TABLE [dbo].[TestTable] ADD C
GO
```

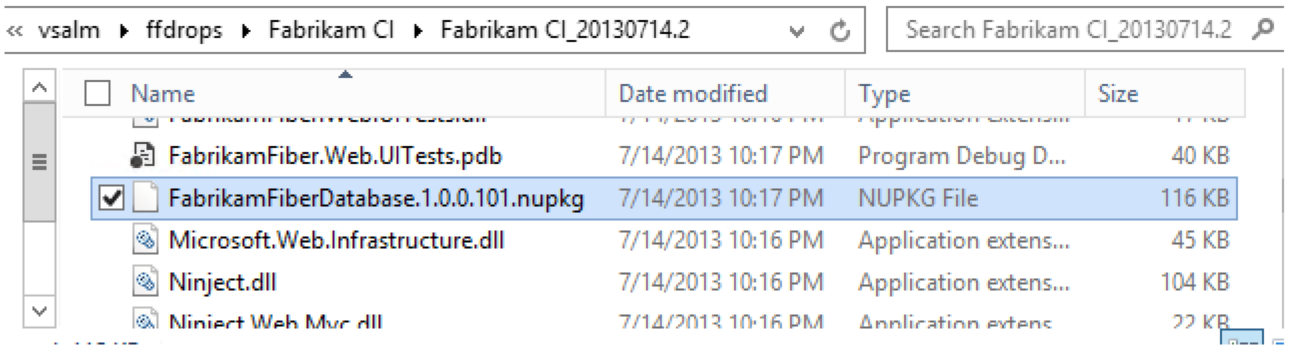
3) A build will now be queued. It builds quickly – it should take only a few seconds to be completed:



Name	Build Definition	Build Quality	Date Completed	Requested By
Fabrikam CI_20130714.2	Fabrikam CI		9 minutes ago	Brian Keller

Note: You may see the test *CreateNullCustomer* fail, which fails the build. This is a test failure built into FabrikamFiber. Either delete the test, repress C# tests running, or implement a fix. Details of the fix can be found on the internet.

4) If you now double-click on the build, you will go to a Summary page. Click **Open Drop Folder** to inspect the build artifacts. The build output contains the website build artifacts, and FabrikamFiberDatabase.1.0.0.101.nupkg (or similar). Later on, we will be using this Nuget package to do database deployments.



« vsalm ▸ ffdrops ▸ Fabrikam CI ▸ Fabrikam CI_20130714.2

Search Fabrikam CI_20130714.2

Name	Date modified	Type	Size
FabrikamFiber.Web.UITests.pdb	7/14/2013 10:17 PM	Program Debug D...	40 KB
✓ FabrikamFiberDatabase.1.0.0.101.nupkg	7/14/2013 10:17 PM	NUPKG File	116 KB
Microsoft.Web.Infrastructure.dll	7/14/2013 10:16 PM	Application extens...	45 KB
Ninject.dll	7/14/2013 10:16 PM	Application extens...	104 KB
Ninject.Web.Mvc.dll	7/14/2013 10:16 PM	Application extens...	22 KB

Exercise 3 – DB Unit Testing

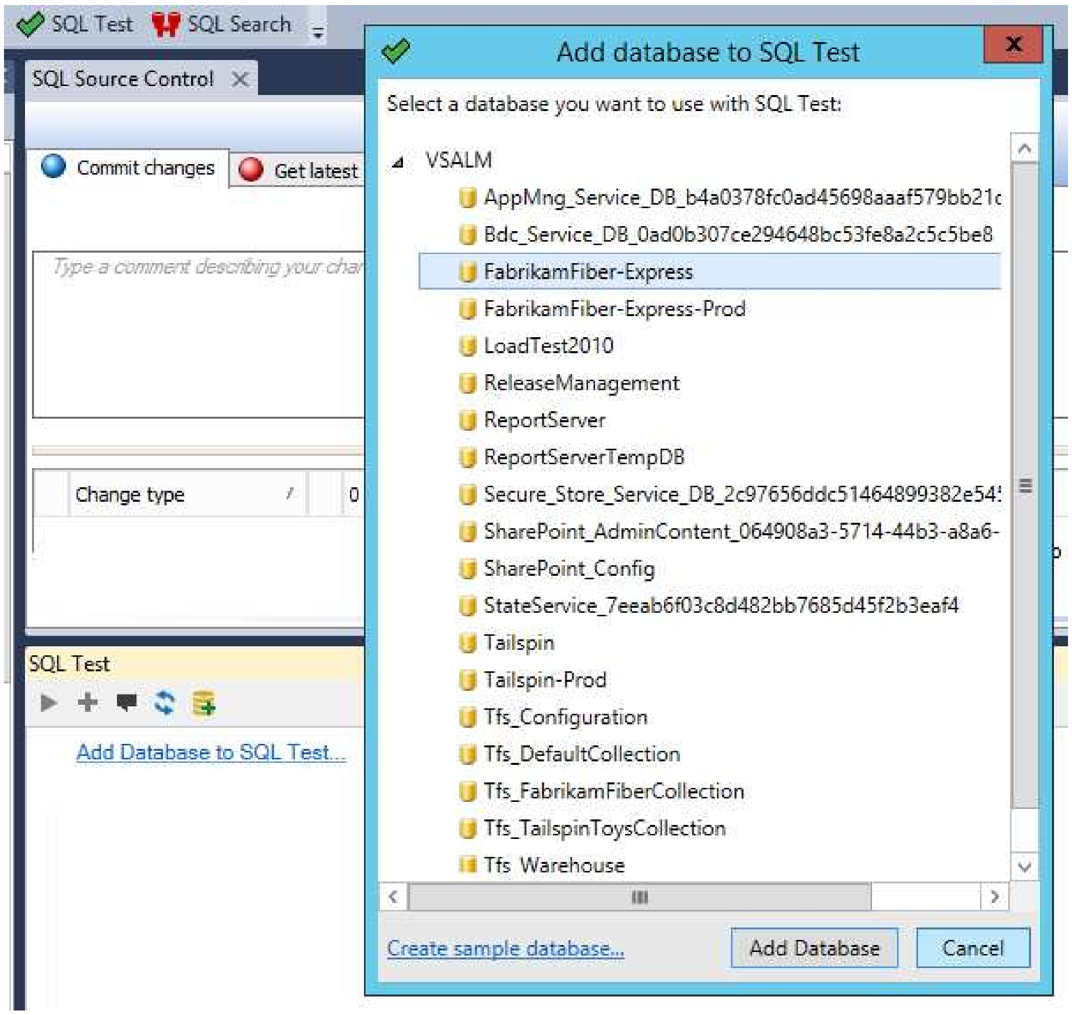
Here, we'll add database unit testing to the FabrikamFiber database. Database unit-testing is justified for the same reasons as unit testing of code. It helps prevent regressions, verify new functionality, and ensure compliance with coding standards. It becomes a gatekeeper that ensures that bad T-SQL is never deployed.

It is not necessary to complete this exercise to proceed to the next exercises.

Redgate SQL Test is an SSMS add-in which uses the tSQLt framework. It is installed as part of the Redgate SQL Developer Bundle, which you installed earlier.

Part 1 – Adding the tests

1) Open SSMS, and open the FabrikamFiber-Express database in the Object Explorer.
2) Click the SQL Test icon ✓ SQL Test in the SSMS toolbar.
3) In the SQL Test window that appears, click **Add Database to SQL Test….**
4) Select FabrikamFiber-Express, and click **Add Database.**


SQL Test | SQL Search | SQL Source Control | Commit changes | Get latest | Type a comment describing your char | Change type | SQL Test | Add Database to SQL Test...

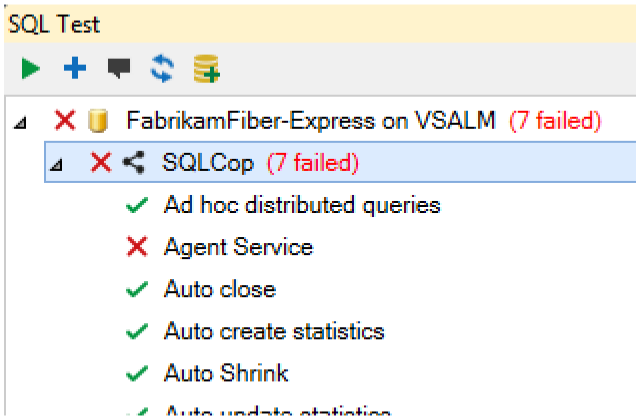
Add database to SQL Test

Select a database you want to use with SQL Test:

- VSALM
  - AppMng_Service_DB_b4a0378fc0ad45698aaaf579bb21c
  - Bdc_Service_DB_0ad0b307ce294648bc53fe8a2c5c5be8
  - FabrikamFiber-Express
  - FabrikamFiber-Express-Prod
  - LoadTest2010
  - ReleaseManagement
  - ReportServer
  - ReportServerTempDB
  - Secure_Store_Service_DB_2c97656ddc51464899382e54!
  - SharePoint_AdminContent_064908a3-5714-44b3-a8a6-
  - SharePoint_Config
  - StateService_7eeab6f03c8d482bb7685d45f2b3eaf4
  - Tailspin
  - Tailspin-Prod
  - Tfs_Configuration
  - Tfs_DefaultCollection
  - Tfs_FabrikamFiberCollection
  - Tfs_TailspinToysCollection
  - Tfs_Warehouse

Create sample database... | Add Database | Cancel

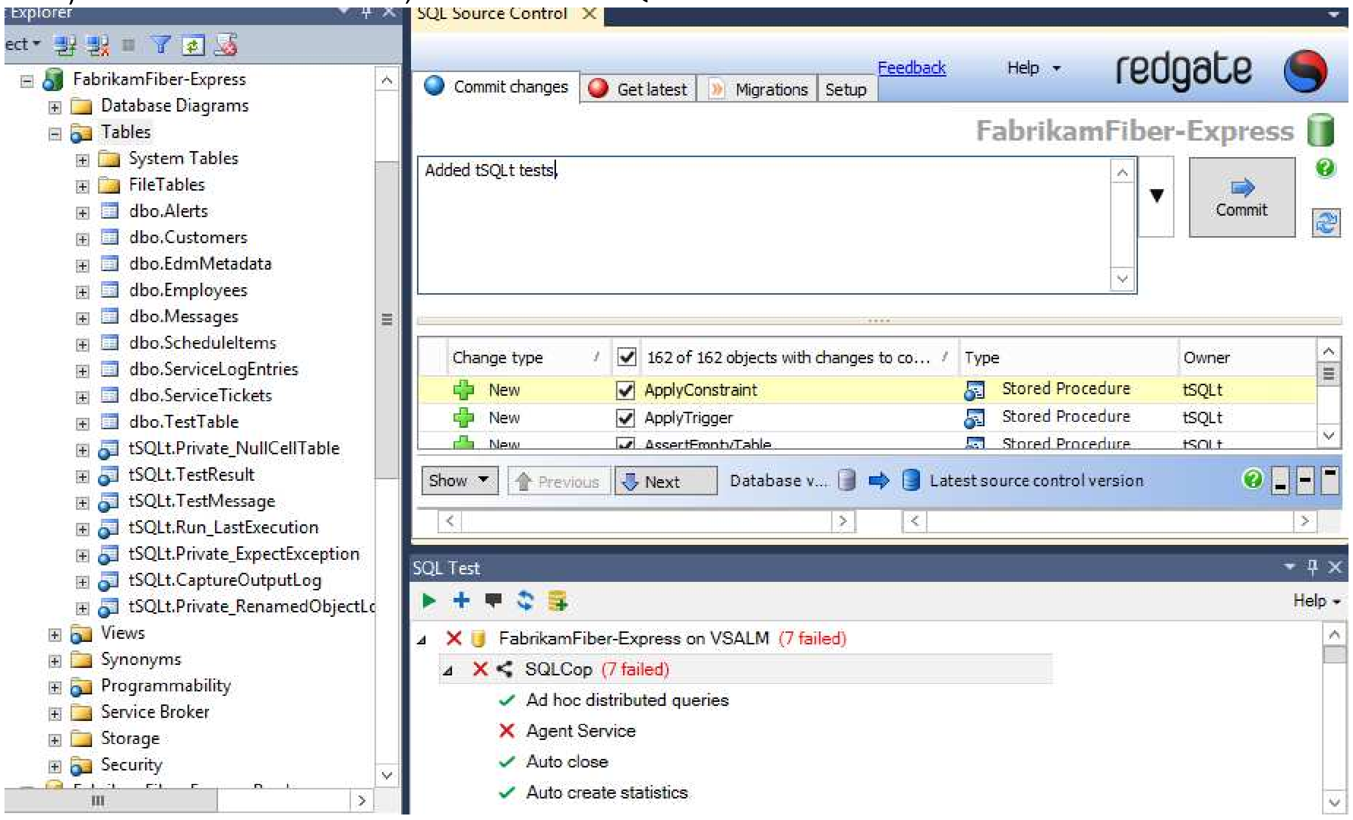
5) The next dialog will ask to install the tSQLt Framework. Click **OK**. Note that it also offers to install SQLCOP tests by default. These are tests that verify certain T-SQL coding standards, e.g. all tables have a primary key.
6) Click **OK** on the confirmation dialog.
7) SQL Test window is now populated with 48 tests. Click on the **green arrow**, and the tests will run. You will see 7 fail. The FabrikamFiber database does not meet the SQLCop requirements. We will deal with these failures later.


SQL Test

- ✗ FabrikamFiber-Express on VSALM (7 failed)
  - ✗ SQLCop (7 failed)
    - ✓ Ad hoc distributed queries
    - ✗ Agent Service
    - ✓ Auto close
    - ✓ Auto create statistics
    - ✓ Auto Shrink

Let's now check these in to source control. If you press the Refresh button on SQL Source Control, lots of objects need to be checked in.

8) Enter a commit comment, such as 'Added tSQLt tests' and click **Commit**.



FabrikamFiber-Express
- Database Diagrams
- Tables
  - System Tables
  - FileTables
  - dbo.Alerts
  - dbo.Customers
  - dbo.EdmMetadata
  - dbo.Employees
  - dbo.Messages
  - dbo.ScheduleItems
  - dbo.ServiceLogEntries
  - dbo.ServiceTickets
  - dbo.TestTable
  - tSQLt.Private_NullCellTable
  - tSQLt.TestResult
  - tSQLt.TestMessage
  - tSQLt.Run_LastExecution
  - tSQLt.Private_ExpectException
  - tSQLt.CaptureOutputLog
  - tSQLt.Private_RenamedObjectLo
- Views
- Synonyms
- Programmability
- Service Broker
- Storage
- Security

Added tSQLt tests

Change type	✓ 162 of 162 objects with changes to co...	Type	Owner
New	✓ ApplyConstraint	Stored Procedure	tSQLt
New	✓ ApplyTrigger	Stored Procedure	tSQLt
New	✓ AssertEmptyTable	Stored Procedure	tSQLt

FabrikamFiber-Express on VSALM (7 failed)
- SQLCop (7 failed)
  - ✓ Ad hoc distributed queries
  - ✗ Agent Service
  - ✓ Auto close
  - ✓ Auto create statistics

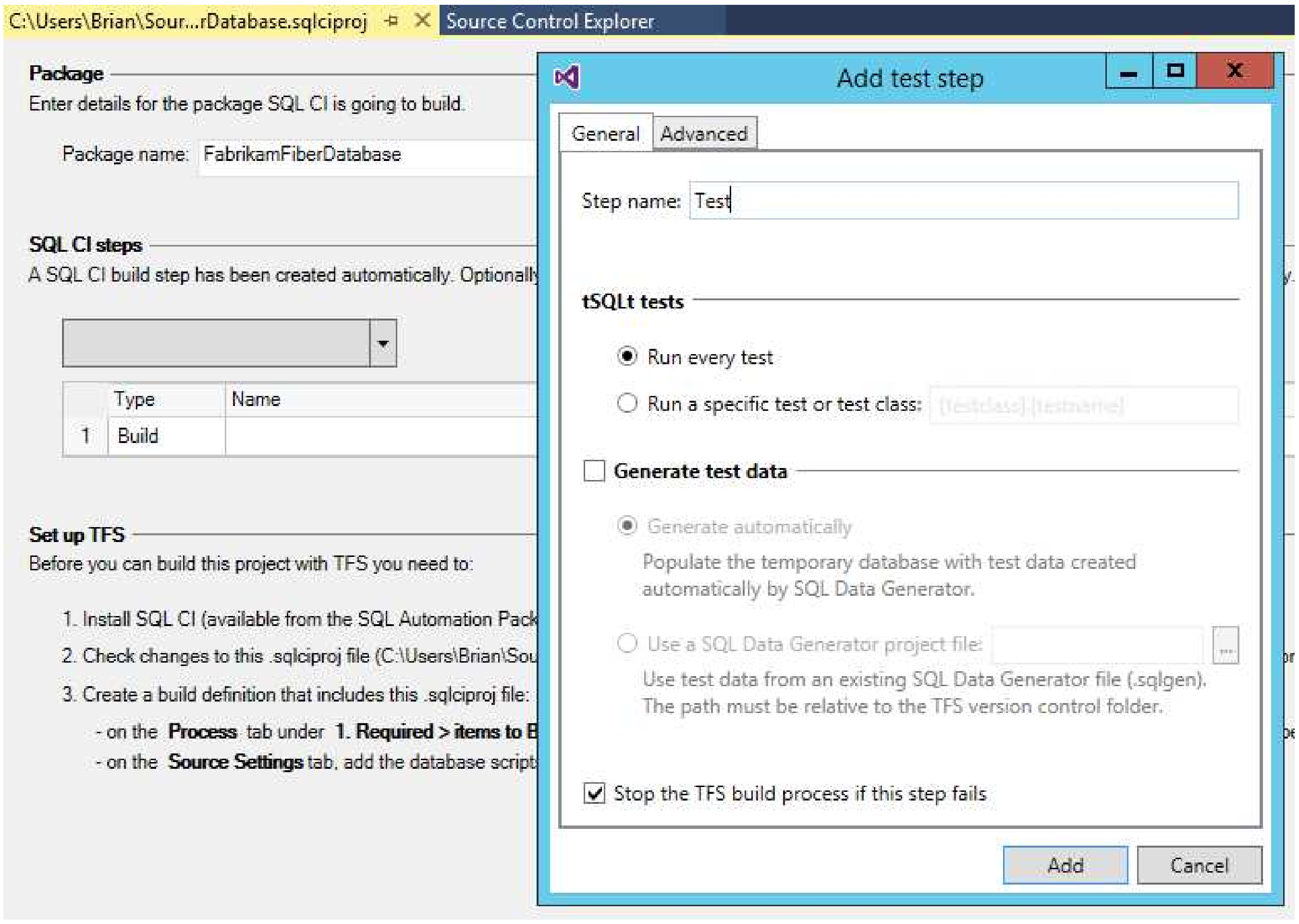
The tSQLt tests are now added to source control. Since we have database CI setup, a build will be automatically triggered. If you open TFS Team Build, you should see it build successfully.

Part 2 – Running the tests in CI

We just ran a DB build successfully. It would be great if we could run the tSQLt tests on every DB build, and fail the build if there are test failures. This can be achieved using Redgate's SQL CI.

First, we need to open the project file that we created in Exercise 2:

1) In Visual Studio, got to **Team Explorer > Source Control Explorer.**
2) Navigate to **vsalm** > **FabrikamFiber** > **Dev** > **Database**, and open the SQL CI Project file, **FabrikamFiberDatabase**.**sqlciproj**.
3) Under SQL CI Steps, click **Add a step** and choose **Test.**
4) The 'Add test step' dialog will appear. Enter 'Test' as the step name, and click **Add.**


C:\Users\Brian\Sour...rDatabase.sqlciproj | Source Control Explorer

Package
Enter details for the package SQL CI is going to build.
Package name: FabrikamFiberDatabase

SQL CI steps
A SQL CI build step has been created automatically. Optionally

	Type	Name
1	Build	

Set up TFS
Before you can build this project with TFS you need to:

1. Install SQL CI (available from the SQL Automation Pack
2. Check changes to this .sqlciproj file (C:\Users\Brian\Sou
3. Create a build definition that includes this .sqlciproj file:
   - on the **Process** tab under **1. Required > items to B**
   - on the **Source Settings** tab, add the database script

Add test step

General | Advanced

Step name: Test

tSQLt tests

(●) Run every test
( ) Run a specific test or test class: [testclass].[testname]

[ ] **Generate test data**

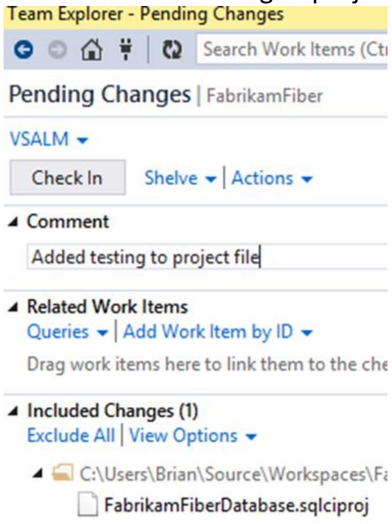
(●) Generate automatically
Populate the temporary database with test data created automatically by SQL Data Generator.

( ) Use a SQL Data Generator project file:
Use test data from an existing SQL Data Generator file (.sqlgen). The path must be relative to the TFS version control folder.

[✓] Stop the TFS build process if this step fails

Add | Cancel

5) Press **CTRL + S** to save your build definition.
6) To check-in the change to the project file, go to **Team Explorer > Pending Changes**, add a comment like 'Added testing to project file', and click **Check In.**


Team Explorer - Pending Changes

Search Work Items (Ct

Pending Changes | FabrikamFiber

VSALM

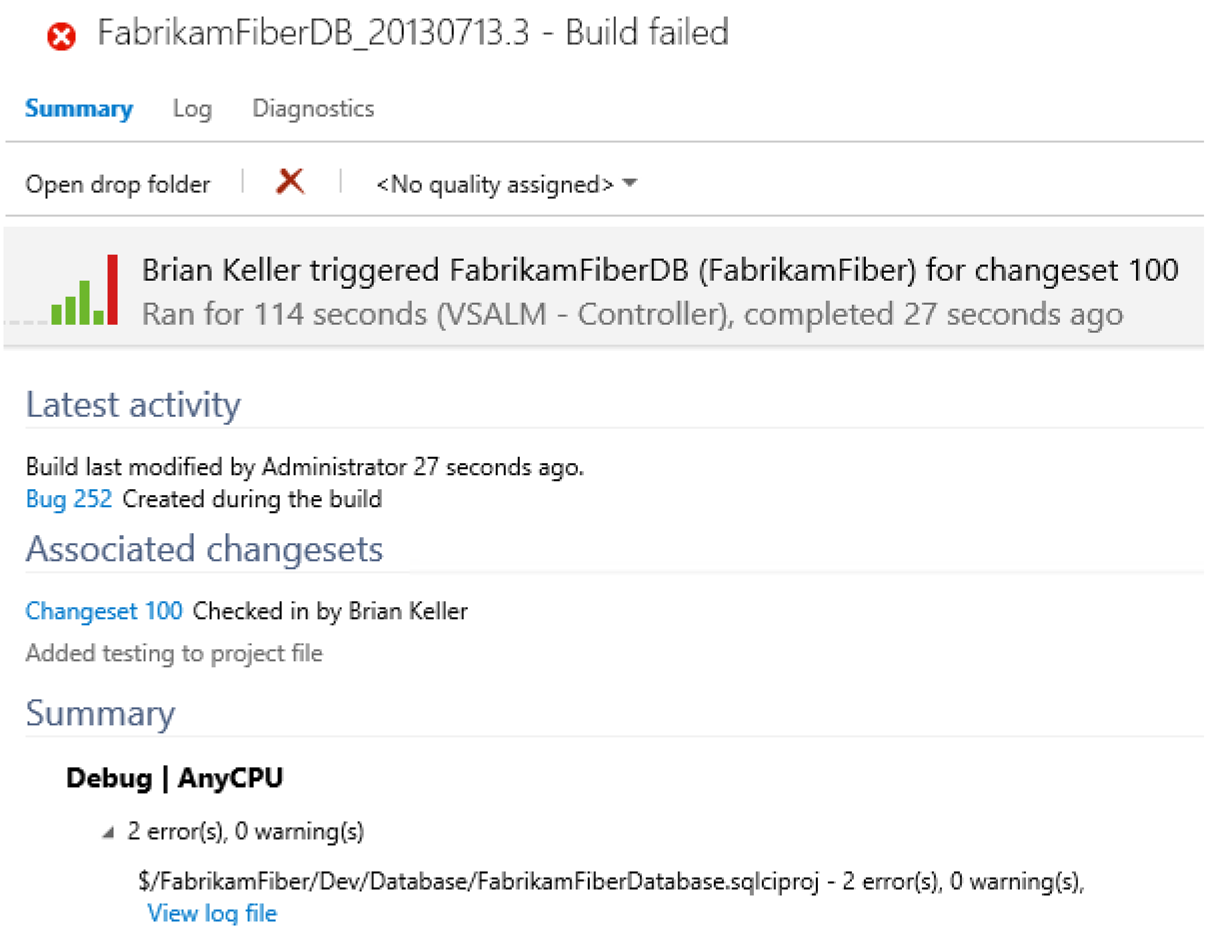
Check In | Shelve | Actions

Comment
Added testing to project file

Related Work Items
Queries | Add Work Item by ID
Drag work items here to link them to the che

Included Changes (1)
Exclude All | View Options
C:\Users\Brian\Source\Workspaces\F
FabrikamFiberDatabase.sqlciproj

7) The project file change will trigger a build. The build will fail because as in SSMS, the FabrikamFiber database doesn't pass all the SQLCop tests.


FabrikamFiberDB_20130713.3 - Build failed

Brian Keller triggered FabrikamFiberDB (FabrikamFiber) for changeset 100
Ran for 114 seconds (VSALM - Controller), completed 27 seconds ago

Latest activity

Build last modified by Administrator 27 seconds ago.
Bug 252 Created during the build

Associated changesets

Changeset 100 Checked in by Brian Keller
Added testing to project file

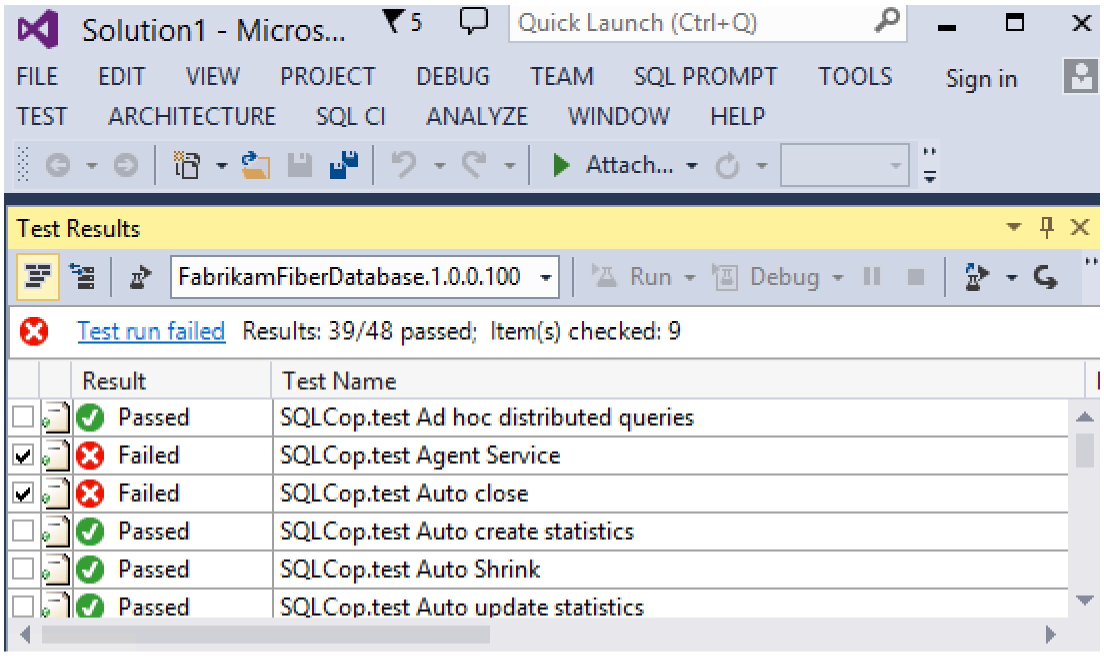
Summary

Debug | AnyCPU

2 error(s), 0 warning(s)

$/FabrikamFiber/Dev/Database/FabrikamFiberDatabase.sqlciproj - 2 error(s), 0 warning(s),
View log file

8) In TFS Team Build, it won't show us exactly which tests failed. This is something we are looking to improve. To see a test breakdown, click **Open drop folder**, and double-click on the trx file. For me, this is called FabrikamFiberDatabase.1.0.0.100.trx.

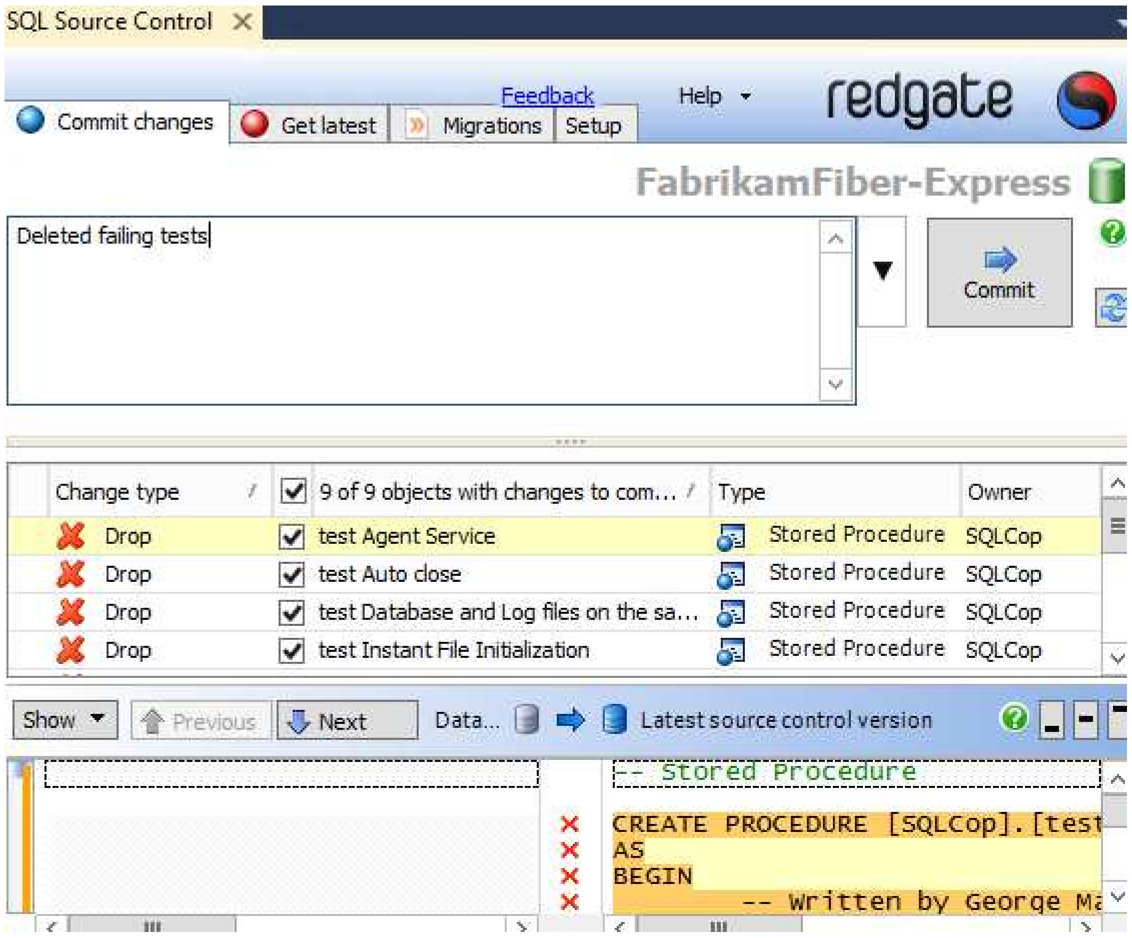

Solution1 - Micros...
Quick Launch (Ctrl+Q)
FILE EDIT VIEW PROJECT DEBUG TEAM SQL PROMPT TOOLS Sign in
TEST ARCHITECTURE SQL CI ANALYZE WINDOW HELP
Attach...
Test Results
FabrikamFiberDatabase.1.0.0.100
Run
Debug

Test run failed Results: 39/48 passed; Item(s) checked: 9

	Result	Test Name
☐	Passed	SQLCop.test Ad hoc distributed queries
☑	Failed	SQLCop.test Agent Service
☑	Failed	SQLCop.test Auto close
☐	Passed	SQLCop.test Auto create statistics
☐	Passed	SQLCop.test Auto Shrink
☐	Passed	SQLCop.test Auto update statistics

Most of these 9 failures are due to poor database design and administration, and some are safe to ignore. In reality you should investigate and fix failed tests, but let's just delete the failing tests:

9) Go to **SSMS**, and the **SQL Test** window.
10) Delete the 9 tests which failed. To delete a test, click on it and press the DELETE key**,** or use the right-click context menu. Note that in SSMS you may only see 7 test failures, because the build has more test failures than SQL Test within SSMS. You should be sure to delete the "Max degree of parallelism" and "Auto close" tests.
11) Now, check in these test deletions into source control. Add a comment, such as 'Deleted failing tests'.



Deleted failing tests

Change type	✓ 9 of 9 objects with changes to com...	Type	Owner
Drop	✓ test Agent Service	Stored Procedure	SQLCop
Drop	✓ test Auto close	Stored Procedure	SQLCop
Drop	✓ test Database and Log files on the sa...	Stored Procedure	SQLCop
Drop	✓ test Instant File Initialization	Stored Procedure	SQLCop

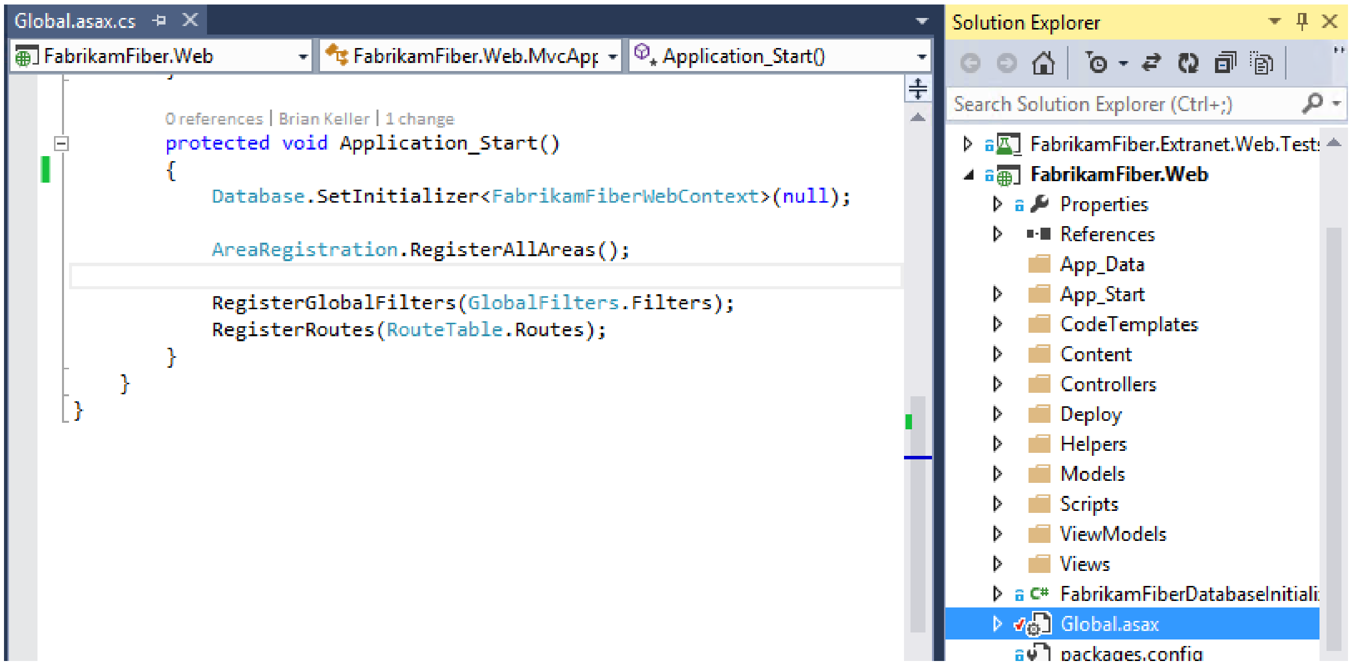
```
-- Stored Procedure

CREATE PROCEDURE [SQLCop].[test
AS
BEGIN
        -- Written by George Ma
```

12) A build will now be triggered, which will succeed with no test failures.

Exercise 4 – Making the FabrikamFiber Website not recreate the database every time

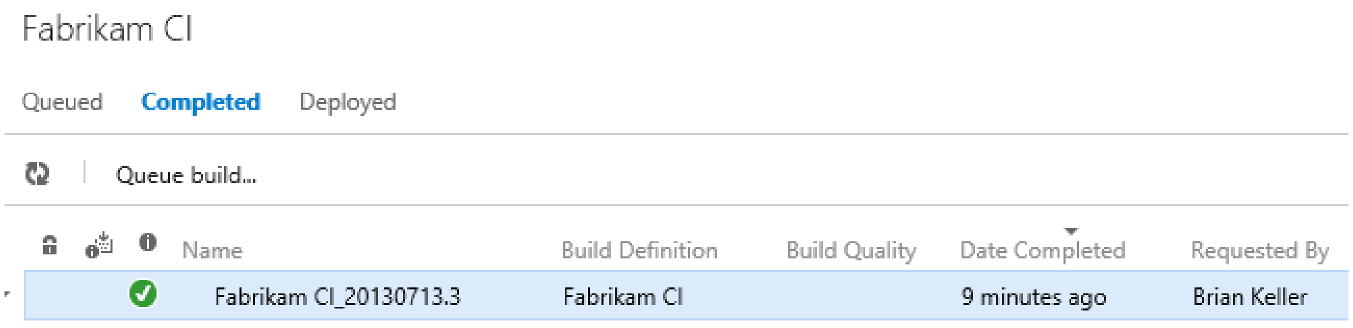
The FabrikamFiber Website uses Entity Framework and will automatically delete and recreate the database on every run. This is undesirable for this lab – we want to be able to work on the database. Let's make a change to suppress this behavior.

1) In the **FabrikamFiber.Web** project, in the file **Global.asax.cs**, find the line `Database.SetInitializer<FabrikamFiberWebContext>(new FabrikamFiberDatabaseInitializer());`
2) Replace this with `Database.SetInitializer<FabrikamFiberWebContext>(null);`



```
0 references | Brian Keller | 1 change
protected void Application_Start()
{
    Database.SetInitializer<FabrikamFiberWebContext>(null);

    AreaRegistration.RegisterAllAreas();

    RegisterGlobalFilters(GlobalFilters.Filters);
    RegisterRoutes(RouteTable.Routes);
}
```

3) Press **CTRL + Shift + B** to invoke a build. It should build successfully.
4) Press **F5** to run a debug build. The website should work as usual.
5) Go to **Team Explorer** > **Pending Changes** and click **Check In** to check in this change. Add a comment, for instance 'Suppressed database creation'.
6) If you go to TFS Team Build, you'll see it start to build automatically. It should finish successfully in just a few seconds.



Name	Build Definition	Build Quality	Date Completed	Requested By
Fabrikam CI_20130713.3	Fabrikam CI		9 minutes ago	Brian Keller

Exercise 5 – Microsoft Release Management: Adding the SQL Release tools

Introduction

In order to perform a DB release, we need to add Redgate SQL Release to MS RM. We do this by writing PowerShell scripts that use the SQL Release cmdlets, add an invocation of this script in MS RM as a *Tool*, and then use this *Tool* from *Actions* or *Components*.

SQL Release provides PowerShell cmdlets for releasing databases safely. Internally, the engine is using Redgate SQL Compare. For instance, running the below in PowerShell will release a database defined in c:\db\DatabaseUpdate to a Staging server:

```
$staging = New-DatabaseConnection -ServerInstance 'staging01\sql2012' -Database 'Staging'
Import-DatabaseUpdate -Path 'C:\db\DatabaseUpdate' | Publish-DatabaseUpdate -Target $staging
```

In this exercise, we'll add tools and actions to MS RM so these commands can be used easily from within MS RM.

The Scripts

We will be using 4 scripts:
PublishDbFromPackage.ps1: This simply deploys from a Nuget package to a database.
RedeploytoStagingFromProduction.ps1: To do a realistic trial deployment against a staging db, we need the staging db to look like the production db. This script makes the staging db have the same schema as the production db.
CreateDbUpdateResources.ps1: Given a production database, staging database and a Nuget package, this confirms that staging matches the schema of production and creates a SQL script that will update the production database to the schema defined in the Nuget package. The script is not invoked - it is just stored to disk. Also, it produces a log of changes and warnings, which should be reviewed prior to proceeding with the deployment.
PublishDbFromArtifacts.ps1: This performs a deployment from the update resources created by the CreateDbUpdateResources script. It checks for drift in the target database from when the update resources were created and then runs the SQL update script.

Feel free to edit these scripts to meet your own requirements. Learn more about the cmdlets in the SQL Release Documentation.

PublishDbFromPackage.ps1

```
param
(
  [Parameter(Mandatory=$true)]
  [string]$PackageFilePathPattern = $null,
  [Parameter(Mandatory=$true)]
  [string]$DatabaseServer = $null,
  [Parameter(Mandatory=$true)]
  [string]$DatabaseName = $null,
  [Parameter(Mandatory=$false)]
  [string]$DatabaseUserName = $null,
  [Parameter(Mandatory=$false)]
  [string]$DatabasePassword = $null
)
```

```
$ErrorActionPreference = "Stop"

$possiblePackageFiles = (Get-ChildItem $PackageFilePathPattern)
if ($possiblePackageFiles.Count -ne 1)
{
    [string]$message = "Error: " + $possiblePackageFiles.Count.ToString() + " files match. Precisely 1 
file should match pattern."
    throw [System.Exception] $message
}
$PackageFilePath = $possiblePackageFiles.FullName

$DbConnectionString = "Data Source=$DatabaseServer;Initial Catalog=$DatabaseName"

# If DatabaseUserName is null or empty, use Windows Authentication. Otherwise, add username and password
if($DatabaseUserName)
{
   $DbConnectionString    += ";User ID=$DatabaseUserName;Password=$DatabasePassword"
}

$update = New-DatabaseUpdate -BeforeUpdate $DbConnectionString -AfterUpdate $PackageFilePath
Publish-DatabaseUpdate $update -Target $DbConnectionString -SkipPostUpdateSchemaCheck
```

PublishDbFromArtifacts.ps1

```
param
(
  [Parameter(Mandatory=$true)]
  [string]$DatabaseUpdateResourcesPath = $null,
  [Parameter(Mandatory=$true)]
  [string]$DatabaseServer = $null,
  [Parameter(Mandatory=$true)]
  [string]$DatabaseName = $null,
  [Parameter(Mandatory=$false)]
  [string]$DatabaseUserName = $null,
  [Parameter(Mandatory=$false)]
  [string]$DatabasePassword = $null
)

$ErrorActionPreference = "Stop"

$DbConnectionString = "Data Source=$DatabaseServer;Initial Catalog=$DatabaseName"

# If DatabaseUserName is null or empty, use Windows Authentication. Otherwise, add username and password
if($DatabaseUserName)
{
   $DbConnectionString    += ";User ID=$DatabaseUserName;Password=$DatabasePassword"
}

Import-DatabaseUpdate $DatabaseUpdateResourcesPath | Publish-DatabaseUpdate -Target $DbConnectionString
```

CreateDbUpdateResources.ps1

```
param
(
  [Parameter(Mandatory=$true)]
  [string]$PackageFilePathPattern = $null,
  [Parameter(Mandatory=$true)]
  [string]$DatabaseServer = $null,
  [Parameter(Mandatory=$true)]
  [string]$StagingDatabaseName = $null,
  [Parameter(Mandatory=$true)]
  [string]$ProductionDatabaseName = $null,
  [Parameter(Mandatory=$false)]
  [string]$DatabaseUserName = $null,
  [Parameter(Mandatory=$false)]
  [string]$DatabasePassword = $null,
  [Parameter(Mandatory=$true)]
  [string]$DatabaseUpdateResourcesPath = $null
)

$ErrorActionPreference = "Stop"

# Match the package file name. Only one file should match
$possiblePackageFiles = (Get-ChildItem $PackageFilePathPattern)
```

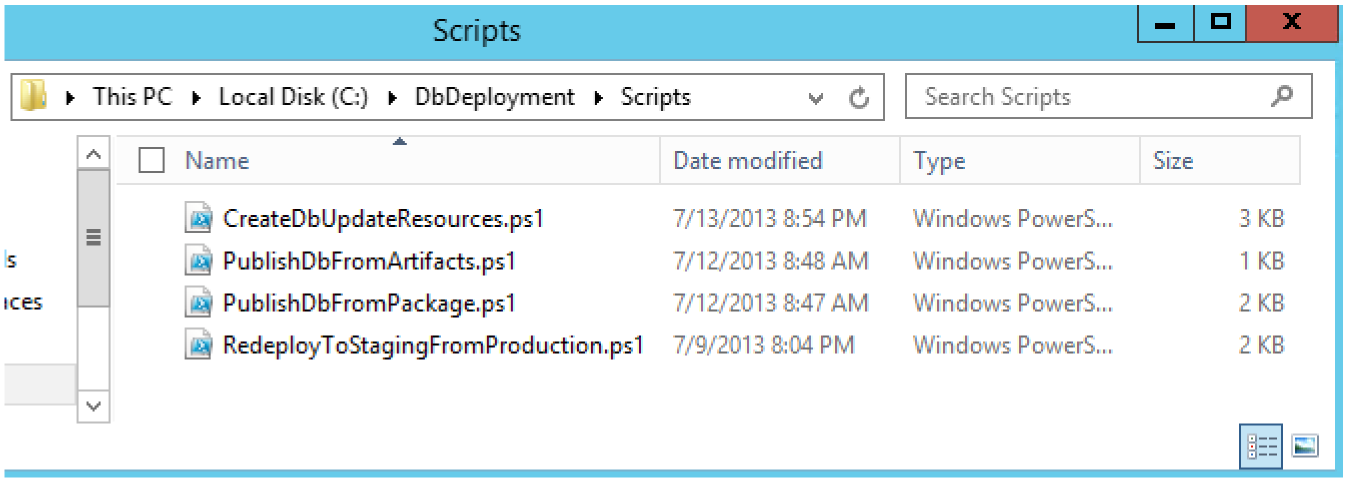
```
if ($possiblePackageFiles.Count -ne 1)
{
    [string]$message = "Error: " + $possiblePackageFiles.Count.ToString() + " files match. Precisely 1
file should match pattern."
    throw [System.Exception] $message
}
$PackageFilePath = $possiblePackageFiles.FullName

# Makes sure the directory we're about to create doesn't already exist.
If (Test-Path $DatabaseUpdateResourcesPath) {
    rmdir $DatabaseUpdateResourcesPath -Recurse -Force
}

# Sets up connection strings for the staging and production databases.
$stagingDatabase = "Data Source=$DatabaseServer; `
                    Initial Catalog=$StagingDatabaseName; `
                    User ID=$DatabaseUserName;Password=$DatabasePassword"
$productionDatabase = "Data Source=$DatabaseServer; `
                       Initial Catalog=$ProductionDatabaseName; `
                       User ID=$DatabaseUserName;Password=$DatabasePassword"

# If DatabaseUserName is null or empty, use Windows Authentication. Otherwise, add username and password
if($DatabaseUserName)
{
   $stagingDatabase    += ";User ID=$DatabaseUserName;Password=$DatabasePassword"
   $productionDatabase += ";User ID=$DatabaseUserName;Password=$DatabasePassword"
}

# Creates the DatabaseUpdateResources directory.
$databaseUpdate = New-DatabaseUpdate -BeforeUpdate @($stagingDatabase, $productionDatabase) `
                   -AfterUpdate $PackageFilePath -Verbose

# Stop on any warnings
$highWarnings = $databaseUpdate.Warnings | Where { $_.Severity -eq "High" }
if($highWarnings.Count -gt 0)
{
    [string]$highWarningsDetails = ""
    $highWarnings | ForEach-Object { $highWarningsDetails = $highWarningsDetails + $_.Details + "`n" }
    throw [System.Exception] $highWarningsDetails
}

# Export the update to disk
Export-DatabaseUpdate $databaseUpdate -Path $DatabaseUpdateResourcesPath
```

RedeployToStagingFromProduction.ps1

```
param
(
  [Parameter(Mandatory=$true)]
  [string]$DatabaseServer = $null,
  [Parameter(Mandatory=$true)]
  [string]$StagingDatabaseName = $null,
  [Parameter(Mandatory=$true)]
  [string]$ProductionDatabaseName = $null,
  [Parameter(Mandatory=$false)]
  [string]$DatabaseUserName = $null,
  [Parameter(Mandatory=$false)]
  [string]$DatabasePassword = $null
)

$ErrorActionPreference = "Stop"

# Sets up connection strings for the staging and production databases.
$stagingDatabase = "Data Source=$DatabaseServer; Initial Catalog=$StagingDatabaseName"
$productionDatabase = "Data Source=$DatabaseServer; Initial Catalog=$ProductionDatabaseName "

# If DatabaseUserName is null or empty, use Windows Authentication. Otherwise, add username and password
if($DatabaseUserName)
{
   $stagingDatabase    += ";User ID=$DatabaseUserName;Password=$DatabasePassword"
   $productionDatabase += ";User ID=$DatabaseUserName;Password=$DatabasePassword"
}

# Update staging DB to be like production DB
$databaseUpdate = New-DatabaseUpdate -BeforeUpdate $stagingDatabase -AfterUpdate $productionDatabase
Publish-DatabaseUpdate $databaseUpdate -Target $stagingDatabase -SkipPreUpdateSchemaCheck -
SkipPostUpdateSchemaCheck
```

Adding the tools to MS RM

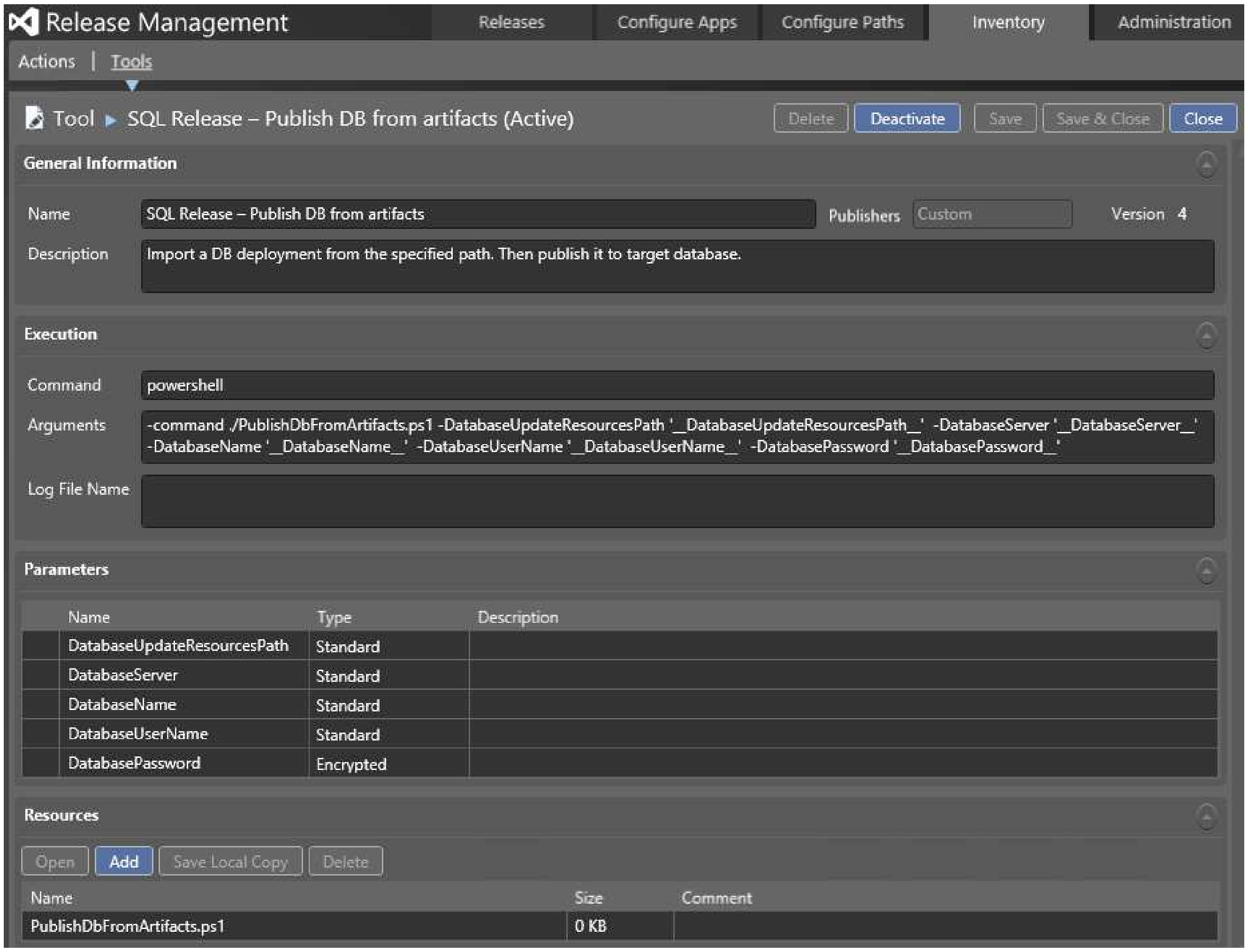
1) Download and install the SQL Release (beta).
2) Create a directory called C:\DbDeployment\Scripts\.
3) Add the four files defined above to this directory.



Name	Date modified	Type	Size
CreateDbUpdateResources.ps1	7/13/2013 8:54 PM	Windows PowerS...	3 KB
PublishDbFromArtifacts.ps1	7/12/2013 8:48 AM	Windows PowerS...	1 KB
PublishDbFromPackage.ps1	7/12/2013 8:47 AM	Windows PowerS...	2 KB
RedeployToStagingFromProduction.ps1	7/9/2013 8:04 PM	Windows PowerS...	2 KB

4) Open MS RM, go to **Inventory > Tools.**
5) Click **New**, and fill in these details:
   a) **Name**: 'SQL Release - Publish DB from artifacts'.
   b) **Description:** 'Import a DB deployment from the specified path. Then publish it to target database'.
   c) **Command**: 'powershell'.
   d) **Arguments**: '-command ./PublishDbFromArtifacts.ps1 -DatabaseUpdateResourcesPath '__DatabaseUpdateResourcesPath__' -DatabaseServer '__DatabaseServer__' -DatabaseName '__DatabaseName__' -DatabaseUserName '__DatabaseUserName__' -DatabasePassword '__DatabasePassword__' '.
   e) In **Resources**, click **Add**, and add PublishDbFromArtifacts.ps1.
5) In **Parameters**, select **Standard** next to **DatabasePassword** and change this to **Encrypted.**

6) Click **Save** to save this tool.



Tool ▸ SQL Release – Publish DB from artifacts (Active)

General Information

Name: SQL Release – Publish DB from artifacts
Publishers: Custom
Version: 4
Description: Import a DB deployment from the specified path. Then publish it to target database.

Execution

Command: powershell
Arguments: -command ./PublishDbFromArtifacts.ps1 -DatabaseUpdateResourcesPath '__DatabaseUpdateResourcesPath__' -DatabaseServer '__DatabaseServer__' -DatabaseName '__DatabaseName__' -DatabaseUserName '__DatabaseUserName__' -DatabasePassword '__DatabasePassword__'
Log File Name:

Parameters

Name	Type	Description
DatabaseUpdateResourcesPath	Standard	
DatabaseServer	Standard	
DatabaseName	Standard	
DatabaseUserName	Standard	
DatabasePassword	Encrypted	

Resources

Open | Add | Save Local Copy | Delete

Name	Size	Comment
PublishDbFromArtifacts.ps1	0 KB	

7) Repeat this procedure for the other 3 files, according to this table:

Name	SQL Release – Create DB Update Resources
Description	Work out what is needed to make the production database look like the database defined in the nuget package. Export this deployment to the specified path
Command	powershell
Arguments	-command ./CreateDbUpdateResources.ps1 -PackageFilePath '__PackageFilePath__' -DatabaseServer '__DatabaseServer__' -ProductionDatabaseName '__ProductionDatabaseName__' -StagingDatabaseName '__StagingDatabaseName__' -DatabaseUserName '__DatabaseUserName__' -DatabasePassword '__DatabasePassword__' -DatabaseUpdateResourcesPath '__DatabaseUpdateResourcesPath__'
Resources	CreateDbUpdateResources.ps1

Name	SQL Release - Publish DB from package
Description	Publishes a DB from a Nuget package
Command	powershell

Arguments	-command ./PublishDbFromPackage.ps1 -PackageFilePathPattern '__PackageFilePathPattern__' -DatabaseServer '__DatabaseServer__' -DatabaseName '__DatabaseName__' -DatabaseUserName '__DatabaseUserName__' -DatabasePassword '__DatabasePassword__'
Resources	PublishDbFromPackage.ps1

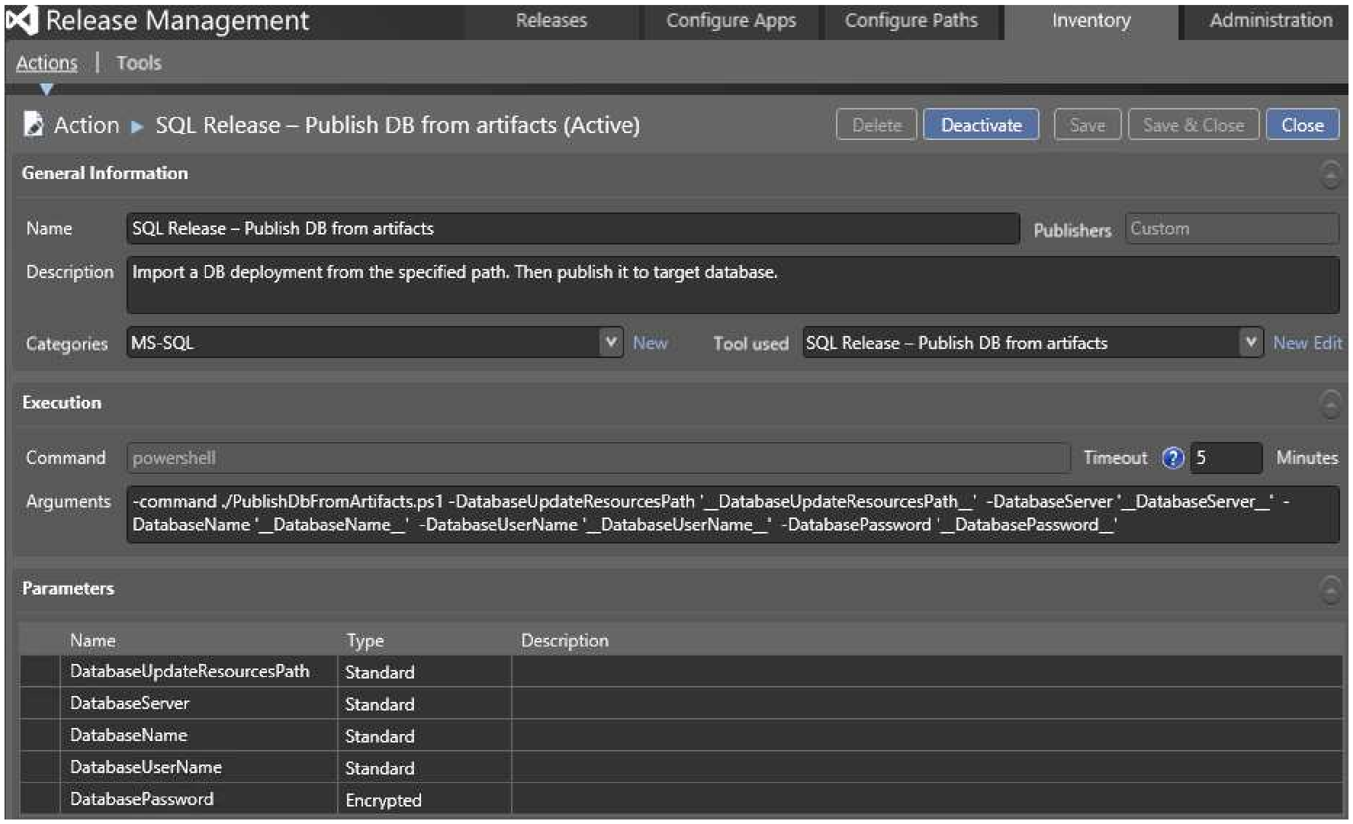
Name	SQL Release - Redeploy to staging from production
Description	Make the staging DB the same as the production DB
Command	powershell
Arguments	-command ./RedeployToStagingFromProduction.ps1 -DatabaseServer '__DatabaseServer__' -ProductionDatabaseName '__ProductionDatabaseName__' -StagingDatabaseName '__StagingDatabaseName__' -DatabaseUserName '__DatabaseUserName__' -DatabasePassword '__DatabasePassword__'
Resources	RedeployToStagingFromProduction.ps1

We now need to add actions and components corresponding to these tools. This will be done in the next two exercises.

Exercise 6 – Microsoft Release Management: Creating database actions

In order to invoke tools in a release template, we need to create *actions* or *components* that invoke those tools. We will create actions corresponding to the **SQL Release – Publish DB from artifacts** and S**QL Release – Redeploy to staging from production** tools**.**

1) In MS RM, go to **Inventory > Actions.**
2) Click **New.**
3) In **Name**, enter 'SQL Release – Publish DB from artifacts'.
4) In **Description**, enter 'Import a DB deployment from the specified path. Then publish it to target database'.
5) In **Categories**, choose MS-SQL.
6) In **Tool used**, choose 'SQL Release – Publish DB from artifacts'.
7) See that **Arguments** and **Parameters** are auto-populated.
8) Click **Save & Close.**



Name	SQL Release – Publish DB from artifacts
Publishers	Custom
Description	Import a DB deployment from the specified path. Then publish it to target database.
Categories	MS-SQL
Tool used	SQL Release – Publish DB from artifacts
Command	powershell
Timeout	5 Minutes

Arguments
```
-command ./PublishDbFromArtifacts.ps1 -DatabaseUpdateResourcesPath '__DatabaseUpdateResourcesPath__' -DatabaseServer '__DatabaseServer__' -
DatabaseName '__DatabaseName__' -DatabaseUserName '__DatabaseUserName__' -DatabasePassword '__DatabasePassword__'
```

Name	Type	Description
DatabaseUpdateResourcesPath	Standard	
DatabaseServer	Standard	
DatabaseName	Standard	
DatabaseUserName	Standard	
DatabasePassword	Encrypted	

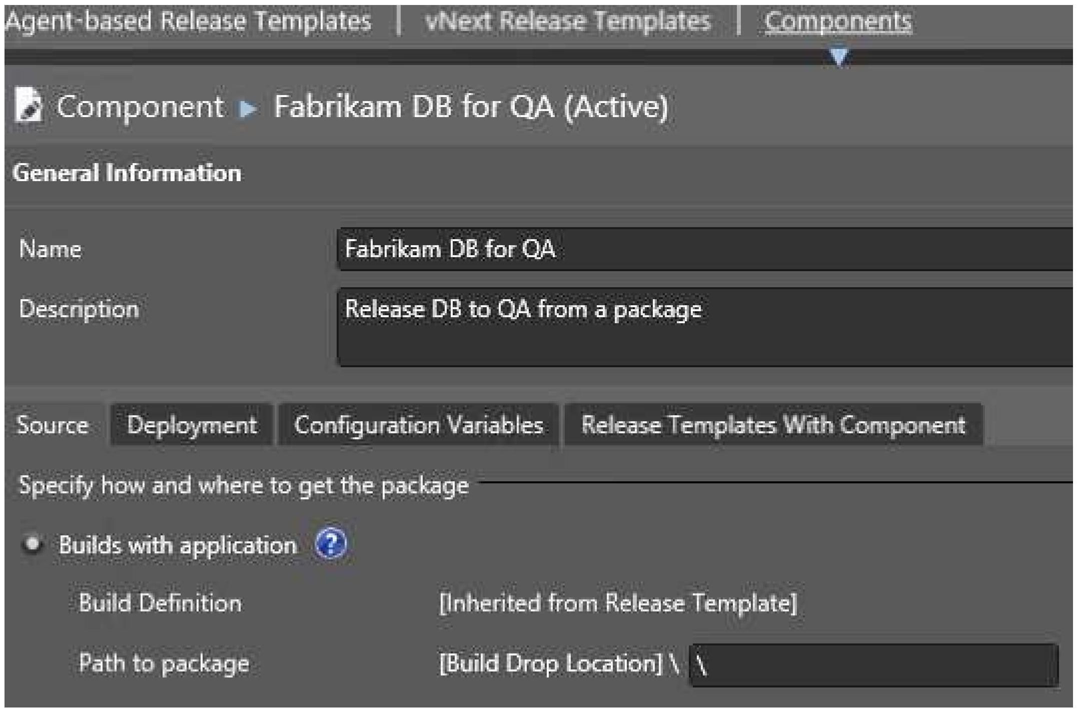
9) Repeat this process, using these parameters:
   a) **Name**: SQL Release - Redeploy to staging from production.
   b) **Description**: Make the staging DB the same as the production DB.
   c) **Categories**: MS-SQL.
   d) **Tool used**: SQL Release – Redeploy to staging from production.

Exercise 7 – Microsoft Release Management: Creating components for database release

We now need to create components corresponding to releasing the database. A *component* is a build definition with an accompanying deployment mechanism. For instance, a component could be a web-app build and XCOPY.

Firstly, we have a **QA** environment. Here, we can deploy the database directly from the package to the server. We create a *component* that does this as so:

1) Go to **Configure Apps > Components.**
2) Click **New.**
3) For **Name**, enter 'Fabrikam DB for QA'.
4) For **Description**, enter 'Release DB to QA from a package'.
5) Choose **Builds with application**, and under **Path to package** enter '\' (i.e. just a backslash).
6) In the **Deployment** tab, set the **Tool** as **SQL Release – Publish DB from Package**.
7) Click **Save & Close.**



Component ▸ Fabrikam DB for QA (Active)

General Information

Name: Fabrikam DB for QA

Description: Release DB to QA from a package

Specify how and where to get the package

● Builds with application

Build Definition: [Inherited from Release Template]

Path to package: [Build Drop Location] \ \

Secondly we have a **Staging** environment. This component is deployed using a different mechanism: an update script is created from the package, which is then reviewed by a DBA before being executed. This component is defined as follows:

1) Go to **Configure Apps > Components.**
2) Click **New.**
3) For **Name**, enter 'Fabrikam DB for release'.
4) For **Description**, enter 'Build deployment artifacts from Nuget package'.
5) Choose **Builds with application**, and under **Path to package** enter '\'.
6) In the **Deployment** tab, set the **Tool** as **SQL Release – Create DB Update Resources.**
7) Click **Save & Close.**

Exercise 8 – Creating a Release Path

In MS RM, we need to create a release path. This defines the stages in a release. We want three stages: QA, Staging and Production

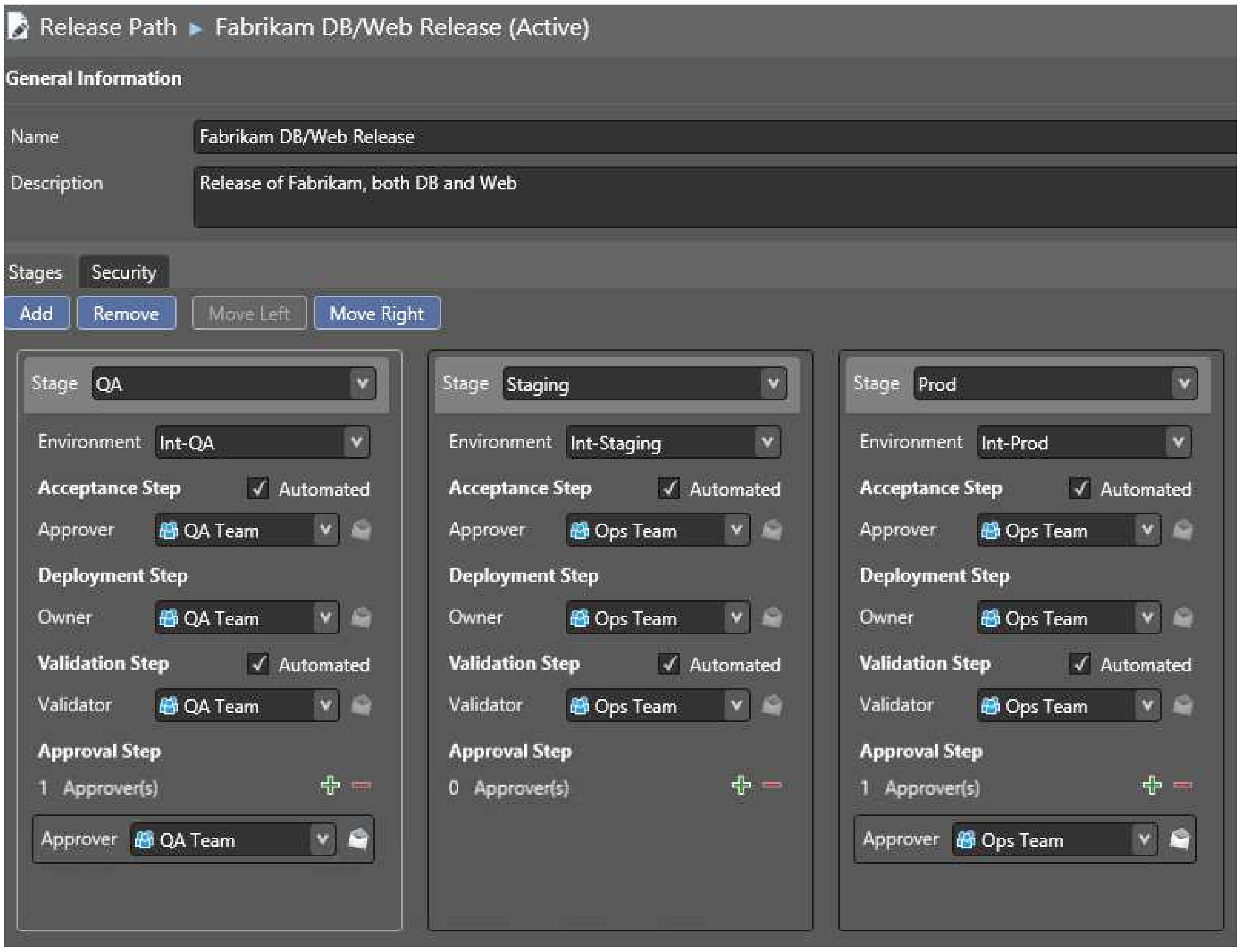
1) In **MS RM**, go **to Configure Paths > Agent-based Release Paths**.
2) Click **New.**
3) Enter 'Fabrikam DB/Web Release' as the **Name**
4) Enter 'Release of Fabrikam, both DB and Web' as the **Description.**
5) Use **Add** to add 3 stages.
6) In **Stage**, from left-to-right, set them as being **QA, Staging** and **Prod.**
7) Set the stages according to this table:

Stage	QA	Staging	Prod
Environment	Int-QA	Int-Staging	Int-prod
Approver	QA Team	Ops Team	Ops Team
Owner	QA Team	Ops Team	Opt Team
Validator	QA Team	Ops Team	Ops Team

Note: You may not have an 'Int-Staging' environment. If not, go to **Configure Paths -> Environments** and click **New** to add one. Click **Link Existing** to link it to our server *VSALM*.

8) Click **Automated** wherever available.
9) Click the **green add symbol** to add an Approver to the QA stage. Set it as **QA Team.**
10) Click the **green add symbol** to add an Approver to the Prod stage. Set it as **Ops Team.**
11) Click the **envelope icon** next to the Approval Step approvers. This will set it to send notification emails at these stages.
12) Click Save & Close.

It should look like this:



Release Path ▸ Fabrikam DB/Web Release (Active)

General Information

Name	Fabrikam DB/Web Release
Description	Release of Fabrikam, both DB and Web

Stages | Security

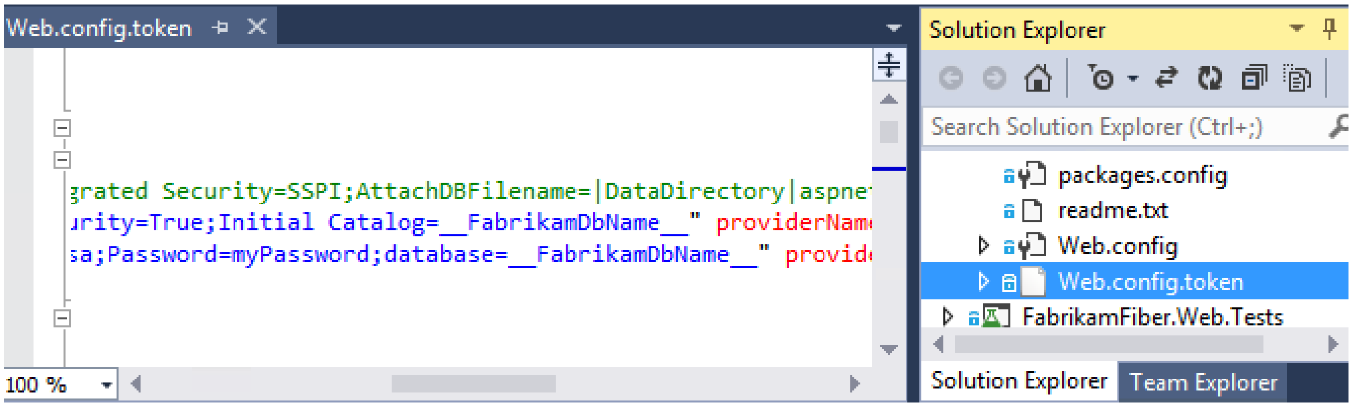
Add | Remove | Move Left | Move Right

Stage	QA	Staging	Prod
Environment	Int-QA	Int-Staging	Int-Prod
Acceptance Step	✓ Automated	✓ Automated	✓ Automated
Approver	QA Team	Ops Team	Ops Team
Deployment Step			
Owner	QA Team	Ops Team	Ops Team
Validation Step	✓ Automated	✓ Automated	✓ Automated
Validator	QA Team	Ops Team	Ops Team
Approval Step	1 Approver(s)	0 Approver(s)	1 Approver(s)
Approver	QA Team		Ops Team

Exercise 9 – Microsoft Release Management: Tokenizing the web.config

FabrikamFiber has a web.config where the connectionString is hard-coded against a single database: FabrikamFiber-Express. We need to have different environments referencing different websites: a QA website that uses a QA database, a Staging website that uses a Staging database, etc. To achieve this, we can use the *tokenization* and *variable replacement* features of MS RM.

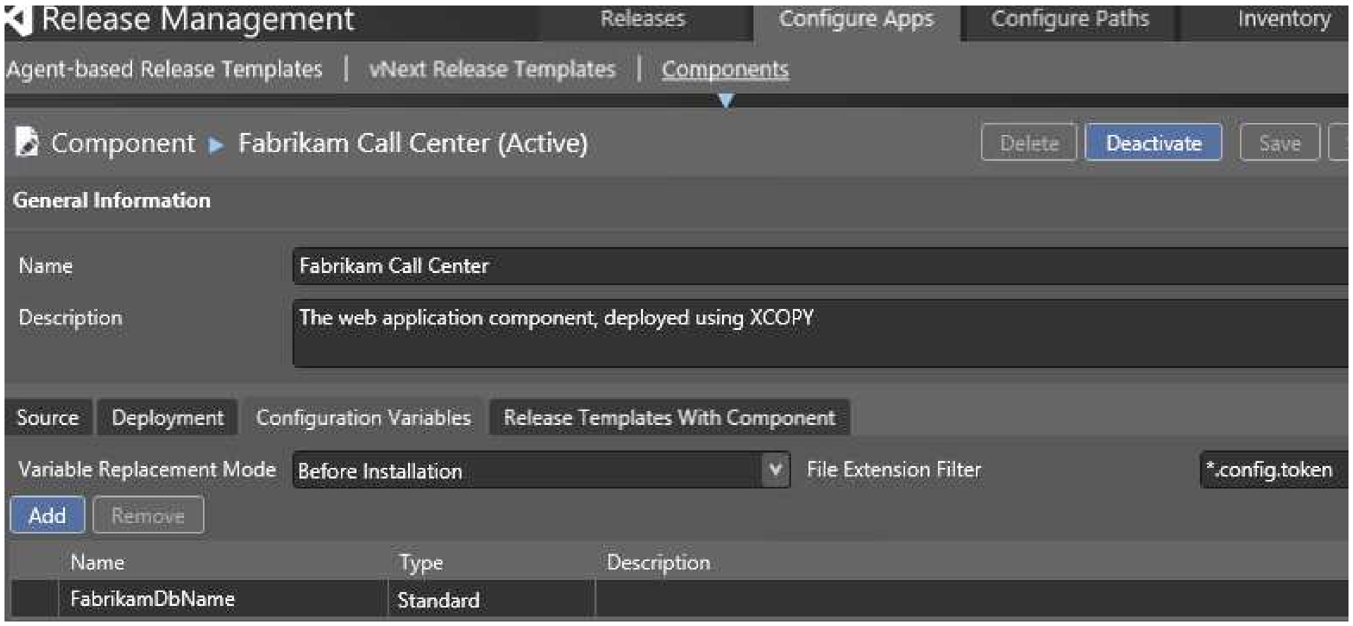
1) Go to Visual Studio, and open the **FabrikamFiber.CallCenter** solution and **FabrikamFiber.Web** project.
2) Copy **Web.config** to **Web.config.token**.
3) Change the *Build action* of Web.config.token to content.
4) In **Web.config.token**, replace the 'FabrikamFiber-Express' string in the connectionString with '__FabrikamDbName__' (note the *double* underscore) in lines 9 and 10.
5) Commit this change to TFS.
6) Check that the build output in the drop folder contains Web.config.token. It should be a file like: C:\ffdrops\Fabrikam CI\Fabrikam CI_20130714.2_PublishedWebsites\FabrikamFiber.Web\Web.config.token.



```
grated Security=SSPI;AttachDBFilename=|DataDirectory|aspne
urity=True;Initial Catalog=__FabrikamDbName__" providerNam
sa;Password=myPassword;database=__FabrikamDbName__" provid
```

We now need to edit the component in MS RM to do variable replacement in web.config:

1) Open the **Fabrikam Call Center** component in MS RM in **Configure Apps > Components.**
2) Go to the **Configuration Variables** tab.
3) Choose **Variable Replacement Mode** as **Before Installation**.
4) Click **Add** and enter 'FabrikamDbName' in **Name**.
5) Under **File Extension Filter**, enter '*.config.token'. This means the only files ending .config.token will have this variable-replacement applied.
6) Click **Save & Close.**



Name: Fabrikam Call Center

Description: The web application component, deployed using XCOPY

Variable Replacement Mode: Before Installation

File Extension Filter: *.config.token

Name	Type	Description
FabrikamDbName	Standard	

The final step is to copy the web.config.token over web.config during the release. This is part of building the release process, and will be covered later.

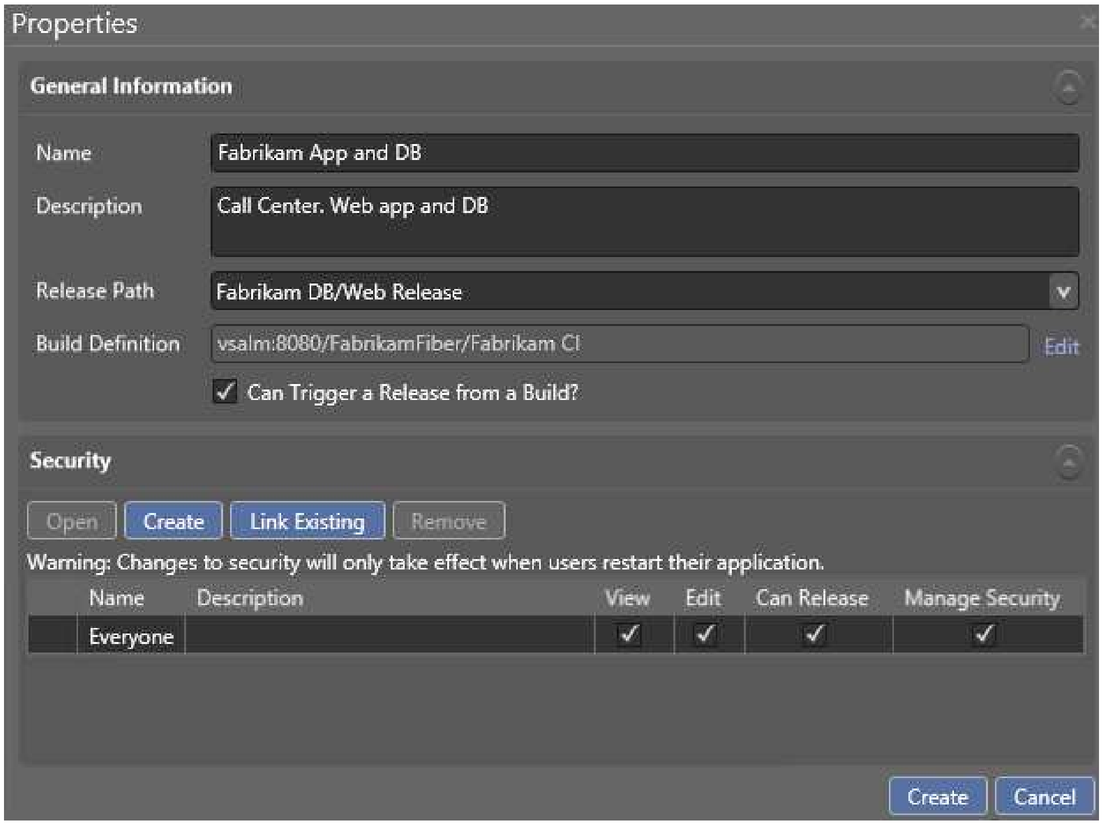
Exercise 10 – Microsoft Release Management: Creating the release template

Now we want to create the Release Template that defines the exact procedure of doing a release of our web-app/database. This will largely be irrelevant if you already have a release template you want to add a database to. If this is the case, skim this exercise and skip to Exercise 11.

We will be basing our template on the existing ***Fabrikam Call Center*** template. You may wish to browse this template before continuing with the exercise.

To create our new release template:

1) Go to **Configure Apps > Agent-based Release Templates.**
2) Click on **Fabrikam Call Center**, and then **Copy.**
3) In the resulting Properties dialog, choose these options:
    a) **Name**: 'Fabrikam App and DB'.
    b) **Description**: 'Call Center. Web app and DB'.
    c) **Release Path**: **Fabrikam DB/Web Release.**
    d) **Build definition**: **http://vsalm:8080/tfs/FabrikamFiber/Fabrikam CI.**
    e) **Can Trigger a Release from a Build?** Tick yes.
    f) Ignore the Security section and Click **Create.**


Properties

General Information

Name: Fabrikam App and DB

Description: Call Center. Web app and DB

Release Path: Fabrikam DB/Web Release

Build Definition: vsalm:8080/FabrikamFiber/Fabrikam CI

✓ Can Trigger a Release from a Build?

Security

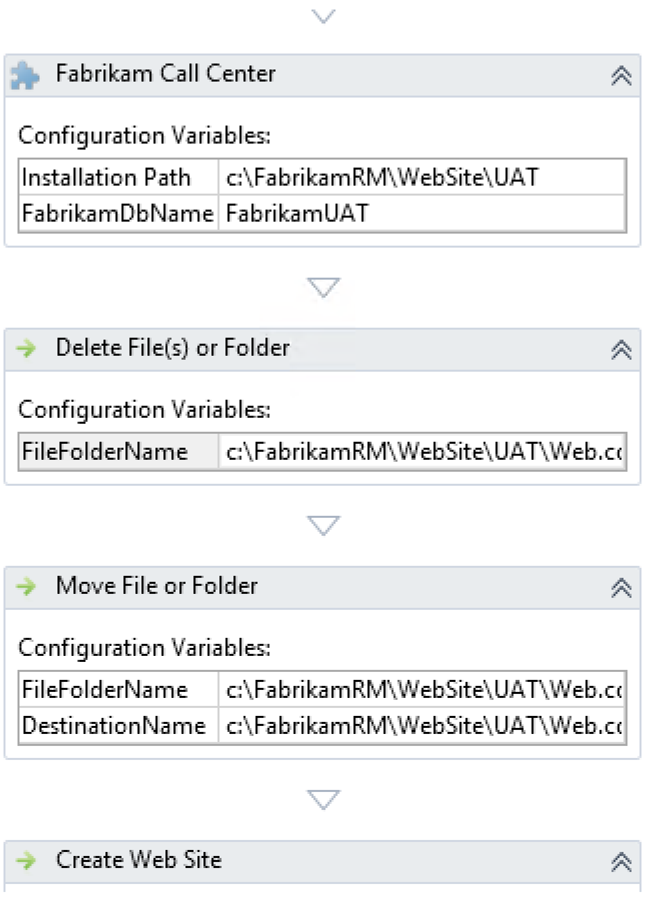
Warning: Changes to security will only take effect when users restart their application.

Name	Description	View	Edit	Can Release	Manage Security
Everyone		✓	✓	✓	✓

This will create a copy with nearly all web application deployment information filled in for us. Have a look around the **QA**, **Staging** and **Prod** stages. You will see that the **QA** and **Prod** stages have lots of steps, but the

Staging stage is empty. We will fill it in momentarily. First, let's add the web.config configuration to the **QA** and **Prod** stages:

1) Go to the **QA** stage.
2) After the **Fabrikam Call Center** component, add a ***Delete File(s) or Folder*** action. To do this, simply drag it from the toolbox on the left, it is under **Windows OS**.
3) Set **FileFolderName** as 'c:\FabrikamRM\WebSite\UAT\Web.config'.
4) Immediately after this action, drag in a **Move File or Folder** action**.**
5) Set **FileFolderName** as 'c:\FabrikamRM\WebSite\UAT\Web.config.token', and **DestinationName** as 'c:\FabrikamRM\WebSite\UAT\Web.config'.



Fabrikam Call Center

Configuration Variables:

Installation Path	c:\FabrikamRM\WebSite\UAT
FabrikamDbName	FabrikamUAT

Delete File(s) or Folder

Configuration Variables:

FileFolderName	c:\FabrikamRM\WebSite\UAT\Web.c

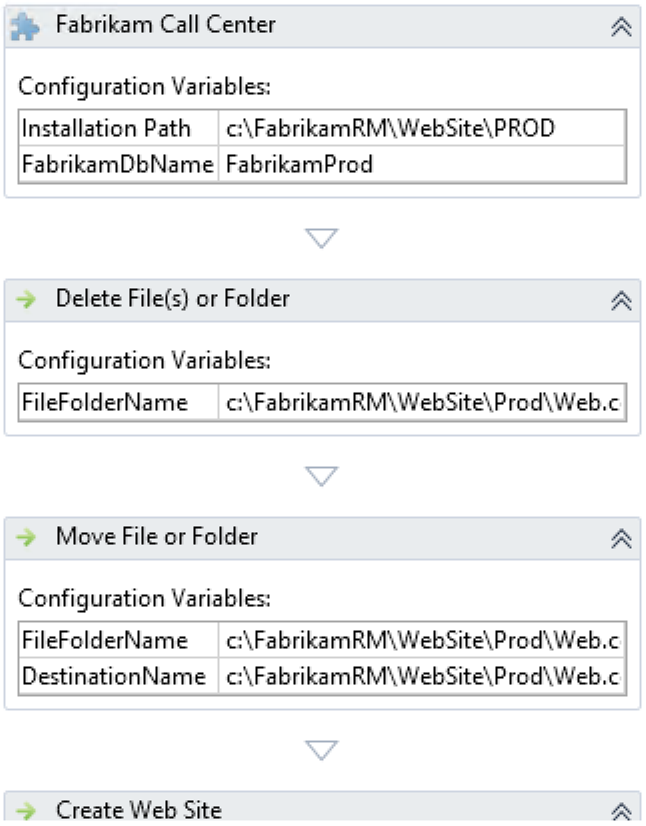
Move File or Folder

Configuration Variables:

FileFolderName	c:\FabrikamRM\WebSite\UAT\Web.c
DestinationName	c:\FabrikamRM\WebSite\UAT\Web.c

Create Web Site

6) Do exactly the same in the **Prod** Stage, except replace **UAT** with **Prod** in the folder names:



Configuration Variables:

Installation Path	c:\FabrikamRM\WebSite\PROD
FabrikamDbName	FabrikamProd

Configuration Variables:

FileFolderName	c:\FabrikamRM\WebSite\Prod\Web.c

Configuration Variables:

FileFolderName	c:\FabrikamRM\WebSite\Prod\Web.c
DestinationName	c:\FabrikamRM\WebSite\Prod\Web.c

Now, let's fill it in the **Staging** stage:

1) Right-click on **Prod** in the deployment sequence at the top.
2) Click **Copy Deployment Sequence.**
3) Right-click on **Staging** in the deployment sequence.
4) Click **Paste Deployment Sequence.**
5) In the **Paste Deployment Sequence** dialog that appears, choose **Paste.**

The **Staging** stage will now be populated.

The staging stage will have various things we need to correct. Basically, we need to replace every instance of 'PROD' with 'Staging', and set the port number to 8001:

1) In **Remove Web Site**, set 'FabrikamPROD' to 'FabrikamStaging'.
2) In **Copy File or Folder**:
    a) Set **SourceFileFolder** as 'C:\FabrikamRM\Website\Staging'.
    b) Set **DestinationFileFolder** as 'C:\FabrikamRM\Backup\Staging'.
3) In **Fabrikam Call Center**, set **Installation Path** to 'C:\FabrikamRM\WebSite\Staging'.
4) In **Delete File(s) or Folder**, set **FileFolderPath** to 'C:\FabrikamRM\WebSite\Staging\Web.config'.
5) In **Move Folder or Folder**:
    a) Set **FileFolderPath** to 'C:\FabrikamRM\WebSite\Staging\Web.config.token'.
    b) Set **DetsinationName** to 'C:\FabrikamRM\WebSite\Staging\Web.config'.
6) In **Create Web Site:**
    a) Set **SiteName** to 'FabrikamStaging'.
    b) Set **PortNumber** to '8001'.

    c) Set **PhysicalPath** to ‘C:\FabrikamRM\WebSite\Staging’.
7) In **Rollback**, **Copy File or Folder**, set:
    a) **SourceFileFolder** to ‘C:\FabrikamRM\Backup\Staging’.
    b) **SourceFileFolder** to ‘C:\FabrikamRM\Website\Staging’.
8) In **Rollback, Create Web Site:**
    a) Set **SiteName** to ‘FabrikamStaging’.
    b) Set **PortNumber** to ‘8001’.
    c) Set **PhysicalPath** to ‘C:\FabrikamRM\WebSite\Staging’.

When you’re done, click **Save & Close.**

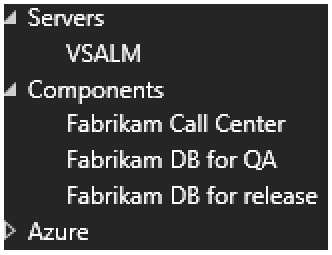
Exercise 11 – Microsoft Release Management: Adding the DB to the release template

We now want to add the database operations to the release template. This will largely be a matter of dragging items from the toolbox to the template, and entering some fields.

Adding the components to the toolbox

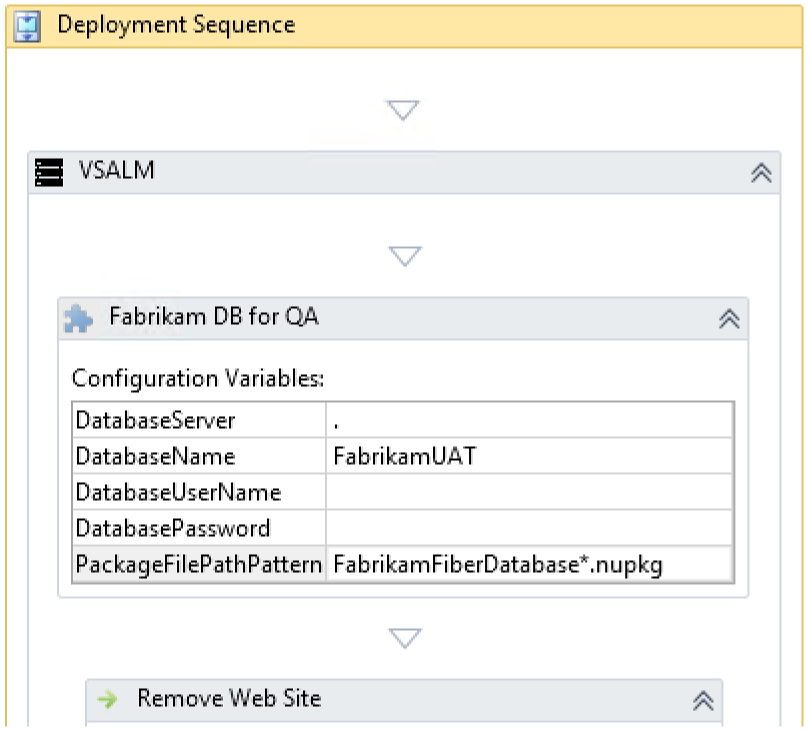
1) Go to **Configure Apps > Agent-based Release Templates > Fabrikam App and DB.**
2) In the **Toolbox**, right click on **Components.**
3) Click **Add.**
4) From the Components dialog, select both components and click **Link.**

All three components will now be listed in the toolbox:



Adding the component to QA

1) Go to the QA stage.
2) Drag **Fabrikam DB for QA** to the sequence, as the first step.
3) Set:
    a) Set **DatabaseServer** as '.' (i.e. the local server).
    b) Set **DatabaseName** as 'FabrikamUAT'.
    c) Leave **DatabaseUserName** and **DatabasePassword** blank. This means we will use Windows Authentication to connect to the database. If they are filled-in, SQL Authentication will be used.
    d) Set **PackageFilePathPattern** as 'FabrikamFiberDatabase*.nupkg'.



Configuration Variables:

DatabaseServer	.
DatabaseName	FabrikamUAT
DatabaseUserName	
DatabasePassword	
PackageFilePathPattern	FabrikamFiberDatabase*.nupkg

Now, as the first step of a QA deployment, SQL Release will deploy from the NuGet package in the build output to the database **FabrikamUAT**, which is used by the UAT web-application.

Note: UAT stands for *User Acceptance Testing*, a type of QA. For our purposes, it is synonymous with QA.

Setting-up the Staging Sequence

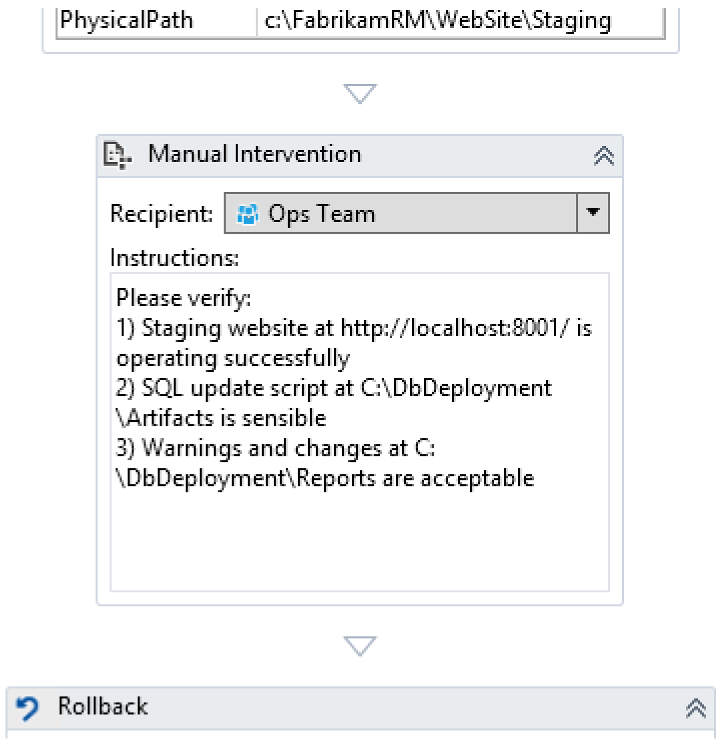
The **Staging** sequence is the most complicated. Four database-related steps are required:

1) Make the Staging DB schema identical to the Production DB schema.
2) Create deployment resources. That is: create the upgrade script needed to perform the deployment to production. Other resources (for instance, a list of warnings) are also produced.
3) Test this upgrade script by applying it to the Staging DB.
4) Insist on *Manual Intervention* by the user. I.e., the user must check the resources and authorize continuation.

Let's set this up now:

1) Go to the **Staging** sequence.
2) Drag **SQL Release – Redeploy to staging from Production** as the first item in the sequence. It will be in the **Toolbox** under **MS-SQL.** Set:
    a) **DatabaseServer** as '.'.
    b) **ProductionDatabaseName** as 'FabrikamProd'.
    c) **StagingDatabaseName** as 'FabrikamStaging'.
    d) **DatabaseUserName** and **DatabasePassword** can be left blank.

3) Drag **Fabrikam DB for release** as the second item in the sequence. It will be in **Components** in the **Toolbox.** Set:
    a) **PackageFilePath** as 'FabrikamFiberDatbase*.nupkg'.
    b) **DatabaseServer** as '.'.
    c) **ProductionDatabaseName** as 'FabrikamProd'.
    d) **StagingDatabaseName** as 'FabrikamStaging'.
    e) **DatabaseUserName** and **DatabasePassword** can be left blank.
    f) **DatabaseUpdateResourcesPath** as 'C:\DbDeployment\Artifacts'.
4) Drag **SQL Release – Publish DB from artifacts** as the third item in the sequence. It will be in the **Toolbox** under **MS-SQL.** Set:
    a) **DatabaseUpdateResourcesPath** as 'C:\DbDeployment\Artifacts'.
    b) **DatabaseServer** as '.'.
    c) **DatabaseName** as 'FabrikamStaging'.
    d) **DatabaseUserName** and **DatabasePassword** can be left blank.
5) As the last item in the main sequence (i.e after 'Create Web Site' but before 'Rollback') add a **Manual Intervention**. This is under **Control Flow** in the **Toolbox**.
    a) Set the **Recipient** as **Ops Team.**
    b) Under **Instructions**, enter this text:
    'Please verify:
    1) Staging website at http://localhost:8001/ is operating successfully.
    2) SQL update script at C:\DbDeployment\Artifacts is sensible.
    3) Warnings and changes at C:\DbDeployment\Reports are acceptable.'.


PhysicalPath c:\FabrikamRM\WebSite\Staging

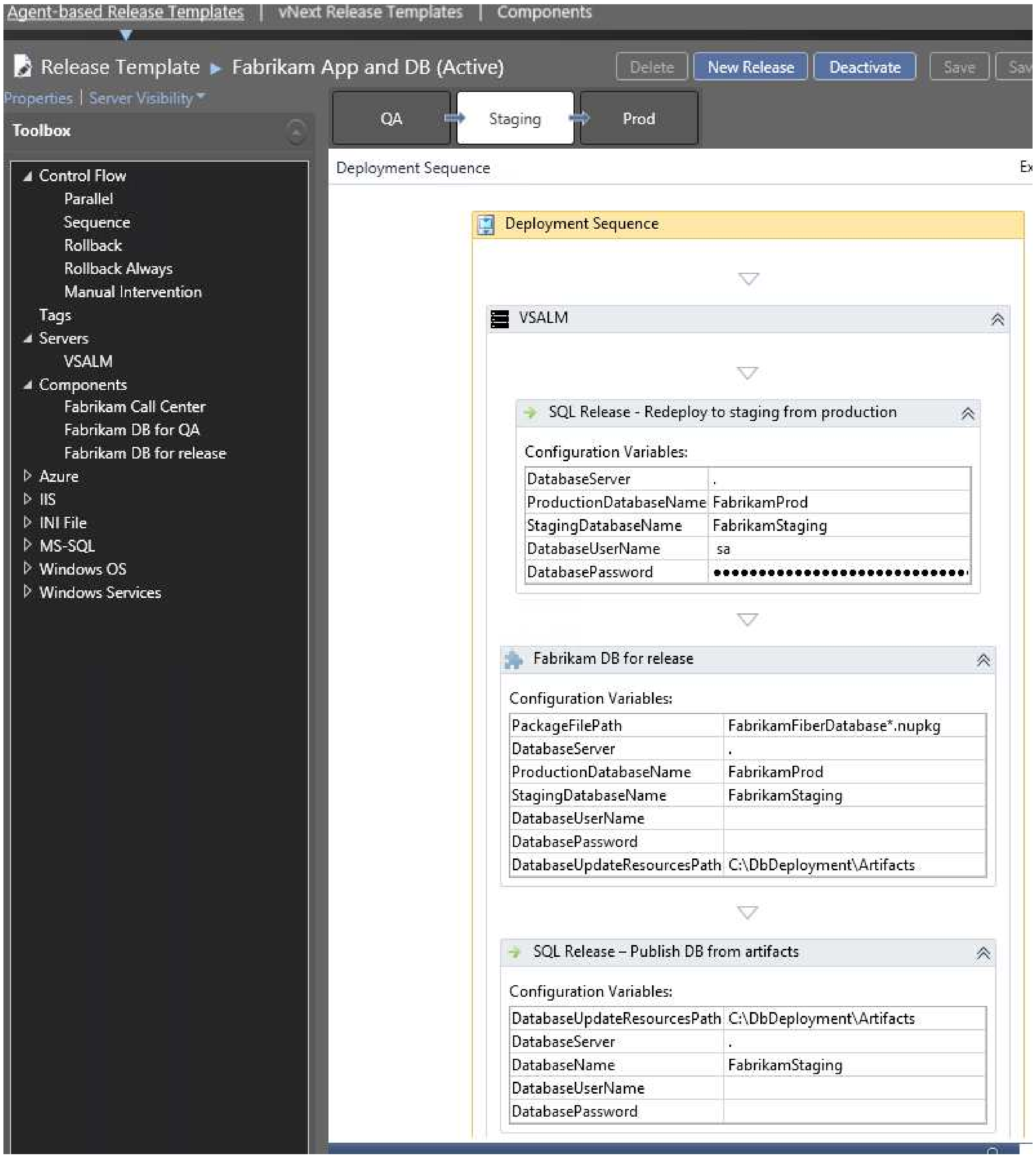
Manual Intervention

Recipient: Ops Team

Instructions:

Please verify:
1) Staging website at http://localhost:8001/ is operating successfully
2) SQL update script at C:\DbDeployment\Artifacts is sensible
3) Warnings and changes at C:\DbDeployment\Reports are acceptable

Rollback



SQL Release - Redeploy to staging from production

Configuration Variables:

DatabaseServer	.
ProductionDatabaseName	FabrikamProd
StagingDatabaseName	FabrikamStaging
DatabaseUserName	sa
DatabasePassword	••••••••••••••••••••••••••••••

Fabrikam DB for release

Configuration Variables:

PackageFilePath	FabrikamFiberDatabase*.nupkg
DatabaseServer	.
ProductionDatabaseName	FabrikamProd
StagingDatabaseName	FabrikamStaging
DatabaseUserName	
DatabasePassword	
DatabaseUpdateResourcesPath	C:\DbDeployment\Artifacts

SQL Release – Publish DB from artifacts

Configuration Variables:

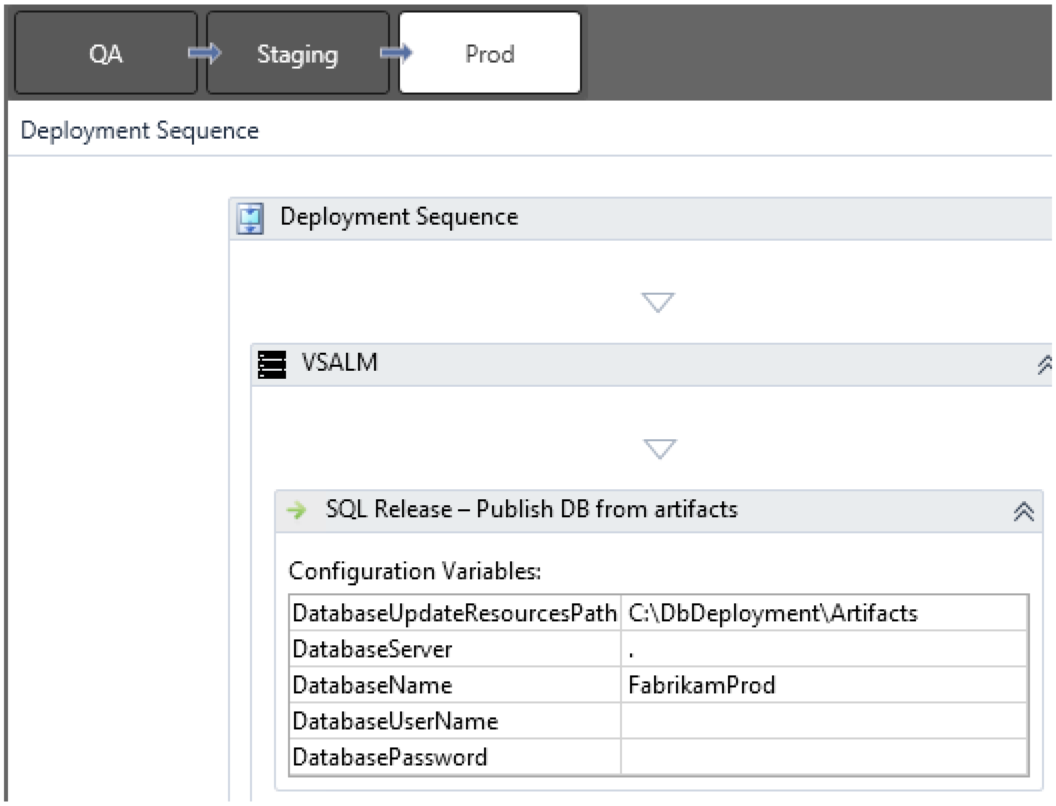
DatabaseUpdateResourcesPath	C:\DbDeployment\Artifacts
DatabaseServer	.
DatabaseName	FabrikamStaging
DatabaseUserName	
DatabasePassword	

Staging is now set up as required.

Setting-up the Production Sequence

In production, we simply need to deploy to the production database from the artifacts we verified.

1) Go to **Prod** in the sequence.
2) Drag **SQL Release – Publish DB from artifacts** as the first item in the sequence. It will be in the **Toolbox** under **MS-SQL.** Set:
    a) **DatabaseUpdateResourcesPath** as 'C:\DbDeployment\Artifacts'.
    b) **DatabaseServer** as '.'.
    c) **DatabaseName** as 'FabrikamProd'.
    d) **DatabaseUserName** and **DatabasePassword** can be left blank.
3) Click **Save & Close.**



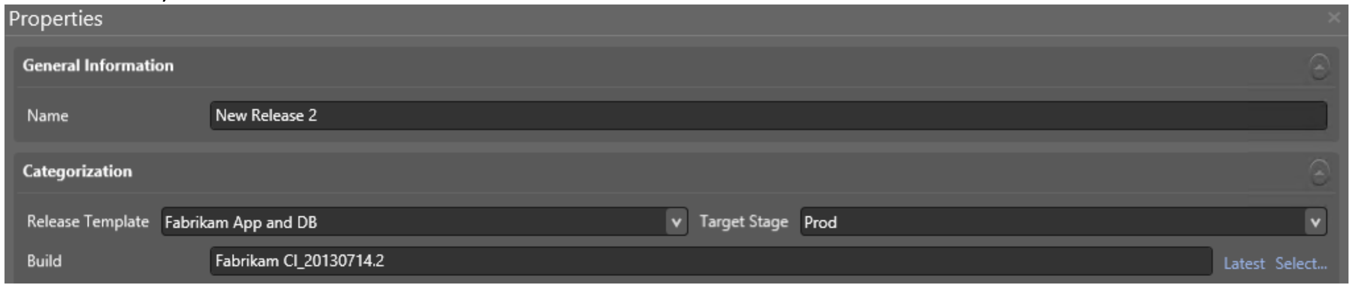
Configuration Variables:

DatabaseUpdateResourcesPath	C:\DbDeployment\Artifacts
DatabaseServer	.
DatabaseName	FabrikamProd
DatabaseUserName	
DatabasePassword	

Exercise 12 – Microsoft Release Management: Let's do a database release

We're now ready to do a release. Before we start, create empty databases for **FabrikamUAT**, **FabrikamStaging** and **FabrikamProd** on the server.

1) In **Releases > Releases**, click **New Agent-based.**
2) In the **Properties** dialog:
    a) Set **Name** as 'Test Release 01', and choose the **Release Template** 'Fabrikam App and DB'.
    b) Leave **Target Stage** as **Prod**.
    c) Click **Latest** to select the Latest build.


Properties

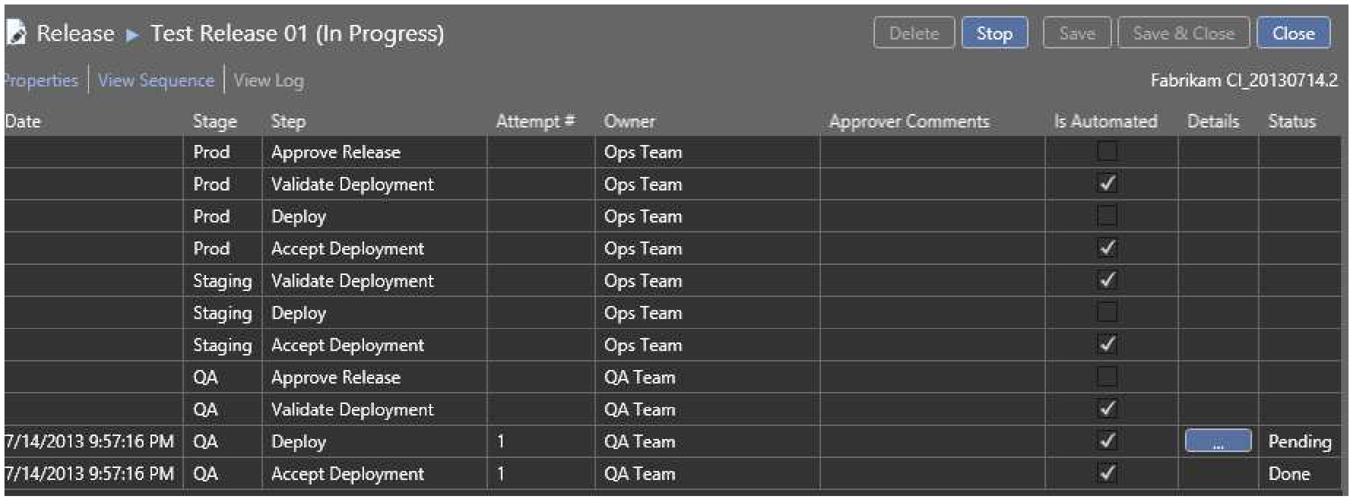
General Information

Name	New Release 2

Categorization

Release Template	Fabrikam App and DB	Target Stage	Prod
Build	Fabrikam CI_20130714.2		

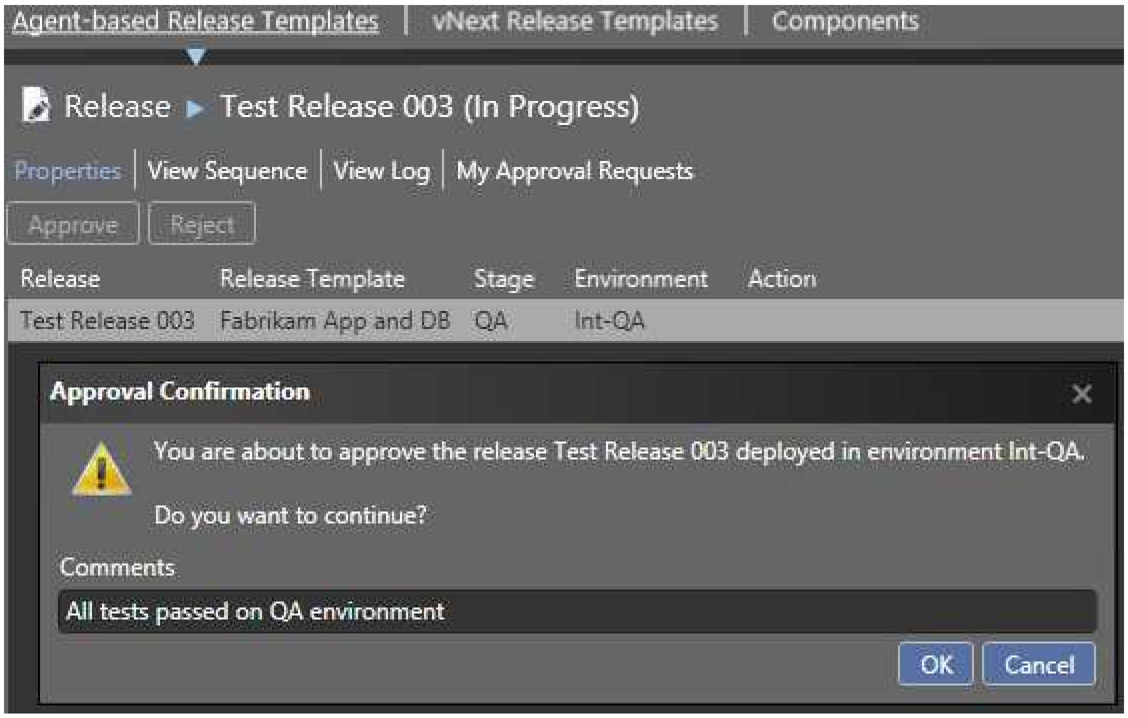
3) Click **Start** to start the release.
4) A release process like this should appear. To see future steps, click **Include Future Steps** (bottom left)**.**



Fabrikam CI_20130714.2

Date	Stage	Step	Attempt #	Owner	Approver Comments	Is Automated	Details	Status
	Prod	Approve Release		Ops Team		☐		
	Prod	Validate Deployment		Ops Team		✓		
	Prod	Deploy		Ops Team		☐		
	Prod	Accept Deployment		Ops Team		✓		
	Staging	Validate Deployment		Ops Team		✓		
	Staging	Deploy		Ops Team		☐		
	Staging	Accept Deployment		Ops Team		✓		
	QA	Approve Release		QA Team		☐		
	QA	Validate Deployment		QA Team		✓		
7/14/2013 9:57:16 PM	QA	Deploy	1	QA Team		✓	...	Pending
7/14/2013 9:57:16 PM	QA	Accept Deployment	1	QA Team		✓		Done

5) Once the QA steps have completed, you'll have a pending Approval Request. At this point, QA should check that the QA website passes the tests, and the QA database is correct. A QA representative should then:
    a) Go to **My Approval Requests**, and select the item.
    b) Click **Approve.**
    c) Enter under **Comments** "All tests passed on QA environment".


Agent-based Release Templates | vNext Release Templates | Components
Release ▸ Test Release 003 (In Progress)
Properties | View Sequence | View Log | My Approval Requests
Approve
Reject

Release	Release Template	Stage	Environment	Action
Test Release 003	Fabrikam App and DB	QA	Int-QA	

Approval Confirmation

You are about to approve the release Test Release 003 deployed in environment Int-QA.

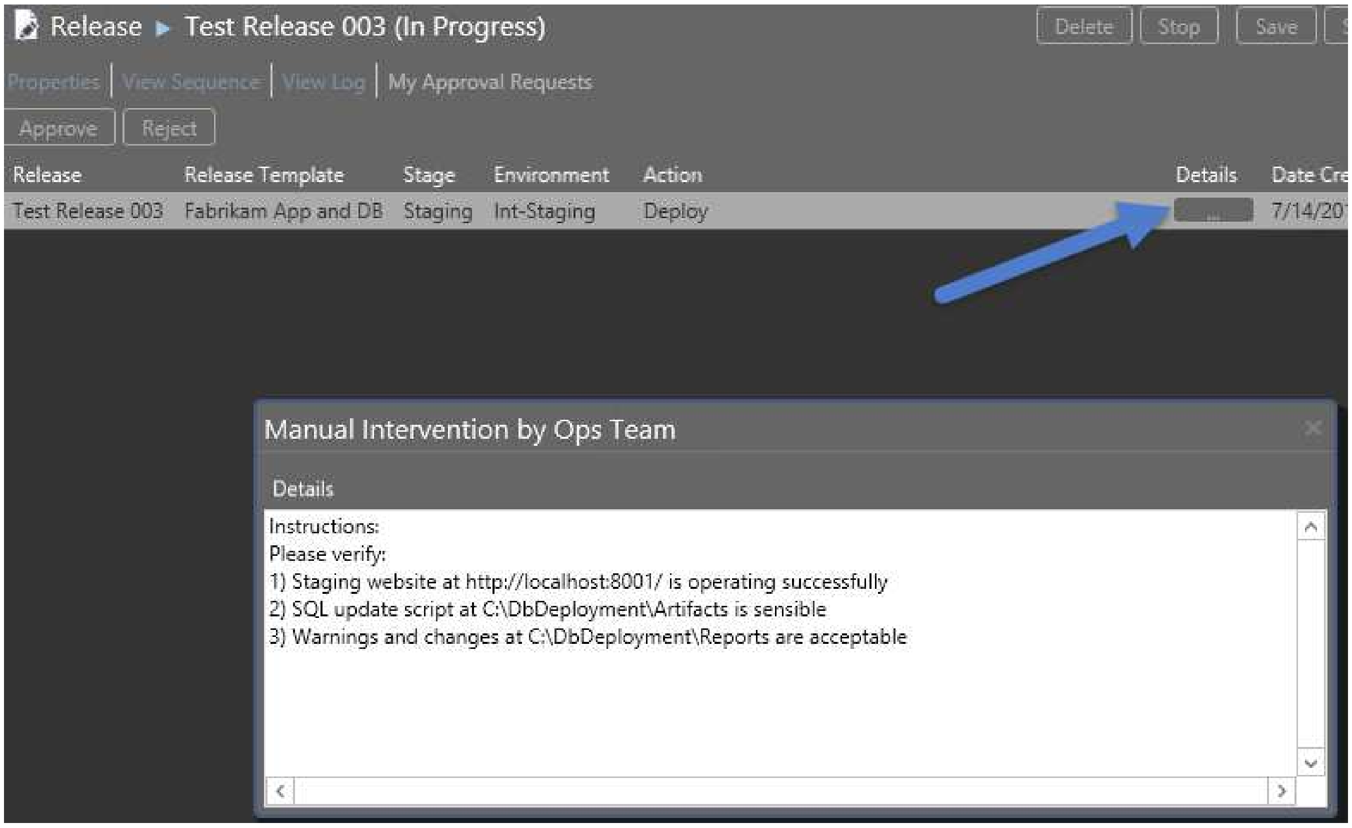
Do you want to continue?

Comments

All tests passed on QA environment

d) Click **OK.**

6) Now the deployment to **Staging** will occur. Once it's completed, you'll be prompted to approve a manual intervention step. Go to **Approval Requests** again, and click on the ... under **Details**:


Release ▸ Test Release 003 (In Progress)
Delete
Stop
Save
Properties | View Sequence | View Log | My Approval Requests
Approve
Reject

Release	Release Template	Stage	Environment	Action	Details	Date Cre
Test Release 003	Fabrikam App and DB	Staging	Int-Staging	Deploy	...	7/14/20

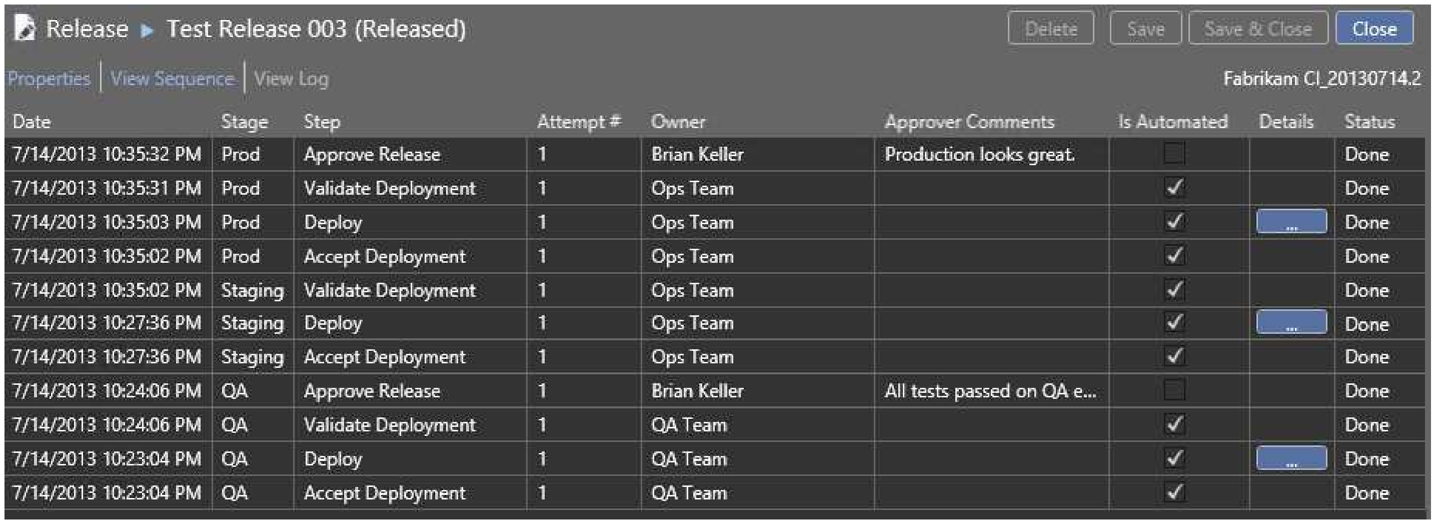
Manual Intervention by Ops Team

Details

```
Instructions:
Please verify:
1) Staging website at http://localhost:8001/ is operating successfully
2) SQL update script at C:\DbDeployment\Artifacts is sensible
3) Warnings and changes at C:\DbDeployment\Reports are acceptable
```

a) Go to **C:\DbDeployment\Artifacts** as instructed. Here you will find **Update.sql**. Open it. This will show what steps are necessary to deploy the DB changes.
b) Go to **C:\DbDeployment\Artifacts\Reports** as instructed. Here, have a look at **Changes.html** and **Warnings.html**. Respectively, these show the changes that will take place when the production deployment occurs, and any warnings you should know about (for instance, that a table is being dropped or a column is being deleted).
c) Check the staging database **FabrikamStaging** and the staging website http://localhost:8001/ . They should match what you want in production.
d) Back in MS RM, click **Approve** to continue with the deployment.
e) Enter 'Staging website looks fine. We're ready to release' as **Comments**, and click **OK.**

7) The Production deployment will now occur. Once it is done, you will again have an approval request waiting. Once you have verified the production database and website are fine, approve this as well. Overall, the deployment should look like this:



Date	Stage	Step	Attempt #	Owner	Approver Comments	Is Automated	Details	Status
7/14/2013 10:35:32 PM	Prod	Approve Release	1	Brian Keller	Production looks great.	☐		Done
7/14/2013 10:35:31 PM	Prod	Validate Deployment	1	Ops Team		✓		Done
7/14/2013 10:35:03 PM	Prod	Deploy	1	Ops Team		✓	...	Done
7/14/2013 10:35:02 PM	Prod	Accept Deployment	1	Ops Team		✓		Done
7/14/2013 10:35:02 PM	Staging	Validate Deployment	1	Ops Team		✓		Done
7/14/2013 10:27:36 PM	Staging	Deploy	1	Ops Team		✓	...	Done
7/14/2013 10:27:36 PM	Staging	Accept Deployment	1	Ops Team		✓		Done
7/14/2013 10:24:06 PM	QA	Approve Release	1	Brian Keller	All tests passed on QA e...	☐		Done
7/14/2013 10:24:06 PM	QA	Validate Deployment	1	QA Team		✓		Done
7/14/2013 10:23:04 PM	QA	Deploy	1	QA Team		✓	...	Done
7/14/2013 10:23:04 PM	QA	Accept Deployment	1	QA Team		✓		Done

Exercise 13 - DLM Dashboard

The DLM Dashboard DLM Dashboard (previously called SQL Lighthouse) provides a visual overview of the state of all your databases across your environments and monitors schema changes. It can be configured to send an alert – both for notifications of unexpected changes (drift) and confirmation that intended changes happened successfully. It also provides a history of SQL changes line by line, who made the changes, and when.

1) Download and install the DLM Dashboard (beta).

What's next?

The next thing to do is to implement this on your own infrastructure. Obviously, your environment is different to that described above, and you may want to make some changes/omissions/additions.

Here's a few things you might want to do:

Invoke a release from Team Foundation Server

You can make MS RM automatically do a release when a build occurs. This requires a couple of steps – search the internet to find out more.

Customize the release process

There's probably a lot you want to change about the release process. In particular, notifications may be sent to various email addresses.

Deploy to multiple databases

You can do this by adding multiple components/actions to your release process. Or you can adapt the PowerShell to deploy to multiple targets.

Working with data

The above process ignores data. You may require ***Static Data*** in your databases. Search Redgate SQL Source Control help for topics on this. Also, you may wish to use Redgate SQL Data Generator to generate test data for your QA (and possibly Staging) stages. Search Redgate SQL CI help for information on how to make this part of your CI process.

Make two builds

In this lab, we have one build for both the application and the database. This is correct for most people, because:

- You want one united build of your whole application.
- It's easier to talk about one build and one build number.
- Application-level automated tests may exercise database code/schema, so it's important to run those when the database changes.

However, you may wish to create a separate build definition just for the db. This may be desirable if multiple applications use a single db.

Feedback

If you have any feedback on this guide or would like to talk to us about database development and deployment processes, please email us at dlm@red-gate.com.