



CONTINUOUS INTEGRATION

**tutorialspoint**

SIMPLY EASY LEARNING

[www.tutorialspoint.com](http://www.tutorialspoint.com)



<https://www.facebook.com/tutorialspointindia>



<https://twitter.com/tutorialspoint>

## About the Tutorial

---

Continuous Integration is a development practice that calls upon development teams to ensure that a build and subsequent testing is conducted for every code change made to a software program.

This concept was meant to remove the problem of finding the late occurrences of issues in the build lifecycle. Instead of the developers working in isolation and not integrating enough, continuous integration was introduced to ensure that the code changes and builds were never done in isolation.

## Audience

---

Continuous integration has become a very integral part of any software development process. It will help the software testing professionals who would like to learn how to build and test their projects continuously in order to help the developers integrate the changes to the project as quickly as possible and obtain fresh builds.

## Prerequisites

---

This is a preliminary tutorial that covers some of the most fundamental concepts of Continuous Integration. Any professional having a good understanding of Software Development should benefit from this tutorial.

## Copyright & Disclaimer

---

© Copyright 2016 by Tutorials Point (I) Pvt. Ltd.

All the content and graphics published in this e-book are the property of Tutorials Point (I) Pvt. Ltd. The user of this e-book is prohibited to reuse, retain, copy, distribute or republish any contents or a part of contents of this e-book in any manner without written consent of the publisher.

We strive to update the contents of our website and tutorials as timely and as precisely as possible, however, the contents may contain inaccuracies or errors. Tutorials Point (I) Pvt. Ltd. provides no guarantee regarding the accuracy, timeliness or completeness of our website or its contents including this tutorial. If you discover any errors on our website or in this tutorial, please notify us at [contact@tutorialspoint.com](mailto:contact@tutorialspoint.com)

## Table of Contents

---

<b>About the Tutorial.....</b>	i
<b>Audience .....</b>	i
<b>Prerequisites .....</b>	i
<b>Copyright &amp; Disclaimer.....</b>	i
<b>Table of Contents.....</b>	ii
<b>1. CI – OVERVIEW .....</b>	1
<b>Why Continuous Integration? .....</b>	1
<b>Workflow .....</b>	1
<b>2. CI – SOFTWARE .....</b>	4
<b>Installing Git.....</b>	5
<b>Configuring Git .....</b>	13
<b>Continuous Integration Server .....</b>	14
<b>Installing TeamCity.....</b>	15
<b>Configuring TeamCity .....</b>	23
<b>The Build Tool .....</b>	28
<b>Database Server .....</b>	29
<b>Web Server .....</b>	39
<b>3. CI – REDUCING RISKS.....</b>	51
<b>Risk 1 – Lack of Deployable Software .....</b>	51
<b>Risk 2 – Discovering Defects Late in the Lifecycle .....</b>	52
<b>Risk 3 – Lack of Project Visibility.....</b>	52
<b>Risk 4 – Low Quality Software .....</b>	53
<b>4. CI – VERSION CONTROL.....</b>	54
<b>Purpose of the Version Control System .....</b>	54
<b>Working with Git for Source Code Versioning Control System.....</b>	55

Moving Source Code to Git.....	56
5. CI – FEATURES .....	58
6. CI – REQUIREMENTS.....	59
7. CI – BUILDING A SOLUTION .....	61
Building a Solution in .Net.....	61
8. CI – BUILD SCRIPTS .....	63
9. CI – BUILDING ON THE SERVER.....	65
10. CI – CHECKING IN SOURCE CODE.....	70
11. CI – CREATING A PROJECT IN TEAMCITY.....	72
12. CI – DEFINING TASKS .....	87
13. CI – BUILD FAILURE NOTIFICATIONS.....	95
14. CI – DOCUMENTATION AND FEEDBACK.....	106
Metrics.....	106
Detailed View of Build Metrics .....	110
15. CI – CONTINUOUS TESTING .....	111
16. CI – CONTINUOUS INSPECTION .....	126
Download and Install NCover .....	126
Configure the Project in TeamCity to Use NCover .....	131
17. CI – CONTINUOUS DATABASE INTEGRATION .....	139
18. CI – CONTINUOUS DEPLOYMENT .....	157
19. CI – BEST PRACTICES.....	182

# 1. CI – Overview

Continuous Integration was first introduced in the year 2000 with the software known as **Cruise Control**. Over the years, Continuous Integration has become a key practice in any software organization. This is a development practice that calls upon development teams to ensure that a build and subsequent testing is conducted for every code change made to a software program. This concept was meant to remove the problem of finding late occurrences of issues in the build lifecycle. Instead of the developers working in isolation and not integrating enough, Continuous Integration was introduced to ensure that the code changes and builds were never done in isolation.

## Why Continuous Integration?

---

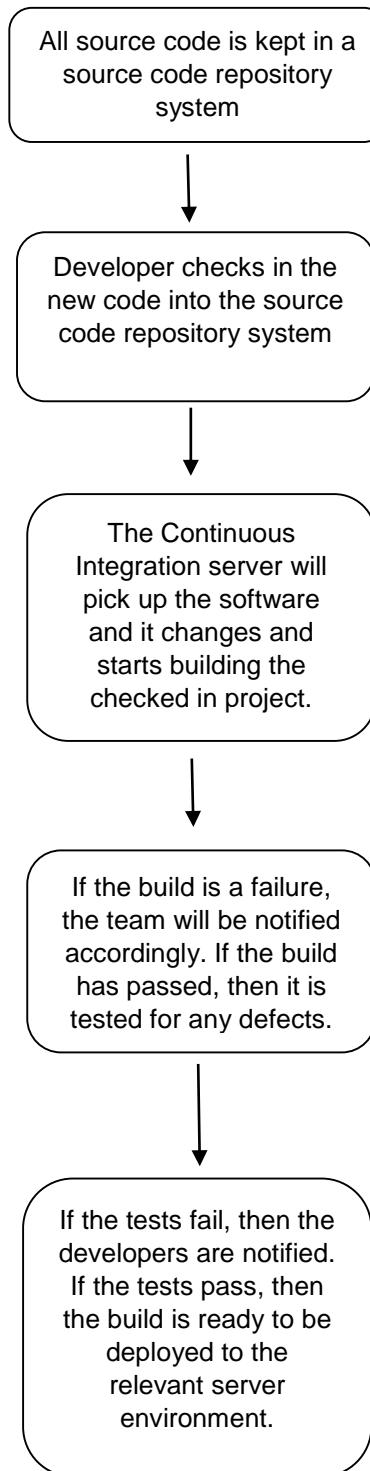
Continuous integration has become a very integral part of any software development process. The continuous Integration process helps to answer the following questions for the software development team.

- Do all the software components work together as they should? – Sometimes systems can become so complex that there are multiple interfaces for each component. In such cases, it's always critical to ensure that all the software components work seamlessly with each other.
- Is the code too complex for integration purposes? – If the continuous integration process keeps on failing, there could be a possibility that the code is just too complex. And this could be a signal to apply proper design patterns to make the code lesser complex and more maintainable.
- Does the code adhere to the established coding standards? – Most of the test cases will always check that the code is adhering to the proper coding standards. By doing an automated test after the automated build, this is a good point to check if the code meets all the desired coding standards.
- How much code is covered by automated tests? – There is no point in testing code if the test cases don't cover the required functionality of the code. So it's always a good practice to ensure that the test cases written should cover all the key scenarios of the application.
- Were all the tests successful after the latest change? – If a test fails, then there is no point in proceeding with the deployment of the code, so this is a good point to check if the code is ready to move to the deployment stage or not.

## Workflow

---

The following image shows a quick workflow of how the entire Continuous Integration workflow works in any software development project. We will look at this in detail in the subsequent chapters.



So, based on the above workflow, this is generally how the continuous integration process works.

- First, a developer commits the code to the version control repository. Meanwhile, the Continuous Integration server on the integration build machine polls source code repository for changes (e.g., every few minutes).
- Soon after a commit occurs, the Continuous Integration server detects that changes have occurred in the version control repository, so the Continuous

Integration server retrieves the latest copy of the code from the repository and then executes a build script, which integrates the software.

- The Continuous Integration server generates feedback by e-mailing build results to the specified project members.
- Unit tests are then carried out if the build of that project passes. If the tests are successful, the code is ready to be deployed to either the staging or production server.
- The Continuous Integration server continues to poll for changes in the version control repository and the whole process repeats.

## 2. CI – Software

The software part is the most important aspect of any Continuous Integration process. This chapter focusses on the software which will be needed for the entire Continuous Integration process.

### Source Code Repository

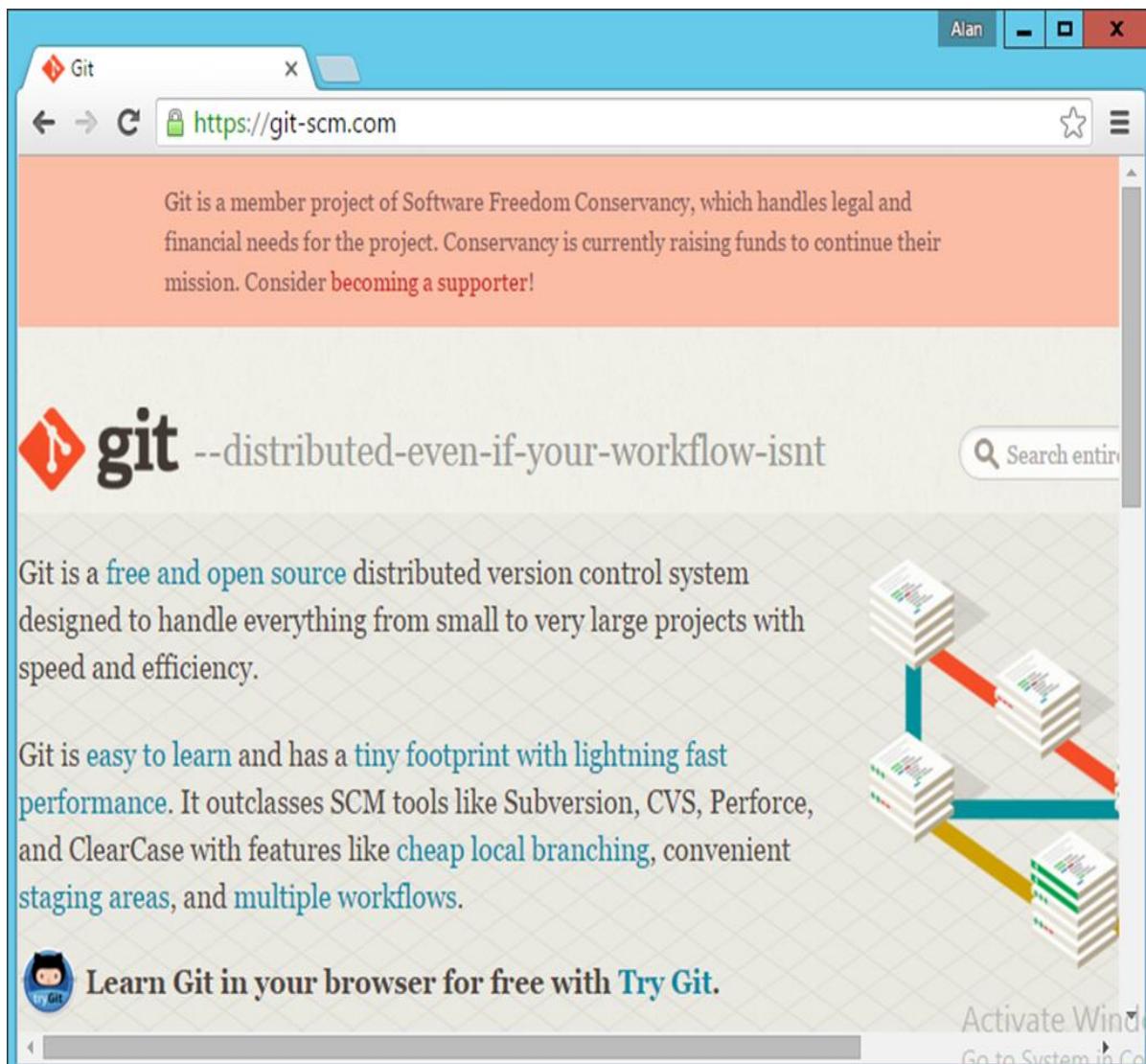
The source code repository is used for maintaining all the source code and all the changes made to it. The two most popular ones for source code repository management is subversion and Git with Git being the most recent popular system. We will now look at how to get Git installed on the system.

### System Requirements

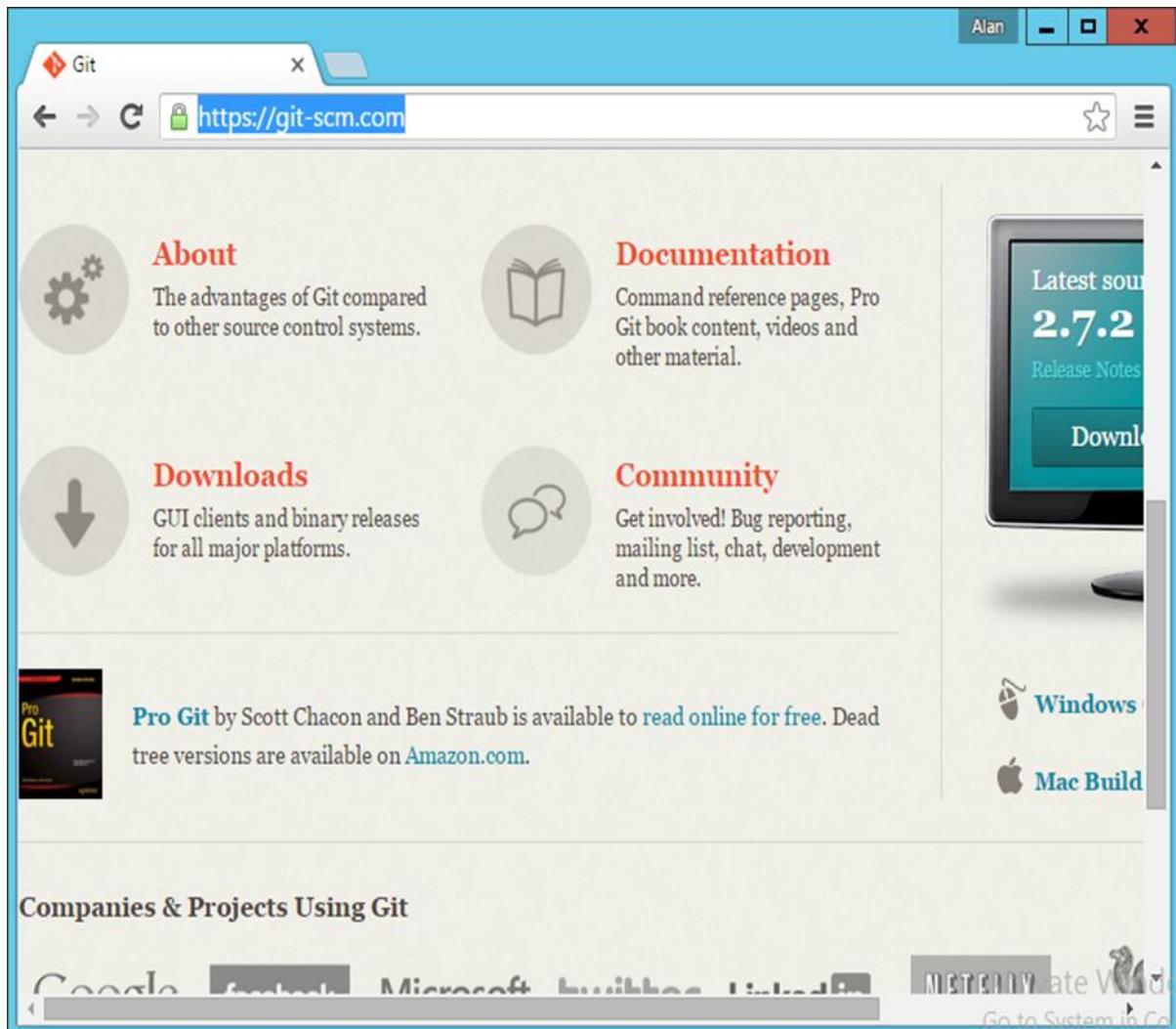
Memory	2 GB RAM (recommended)
Disk Space	200 MB HDD for the installation. Additional storage is required to store the project source code and this is dependent on the source code being added.
Operating System Version	Can be installed on Windows, Ubuntu/Debian, Red Hat/Fedora/CentOS, Mac OS X.

## Installing Git

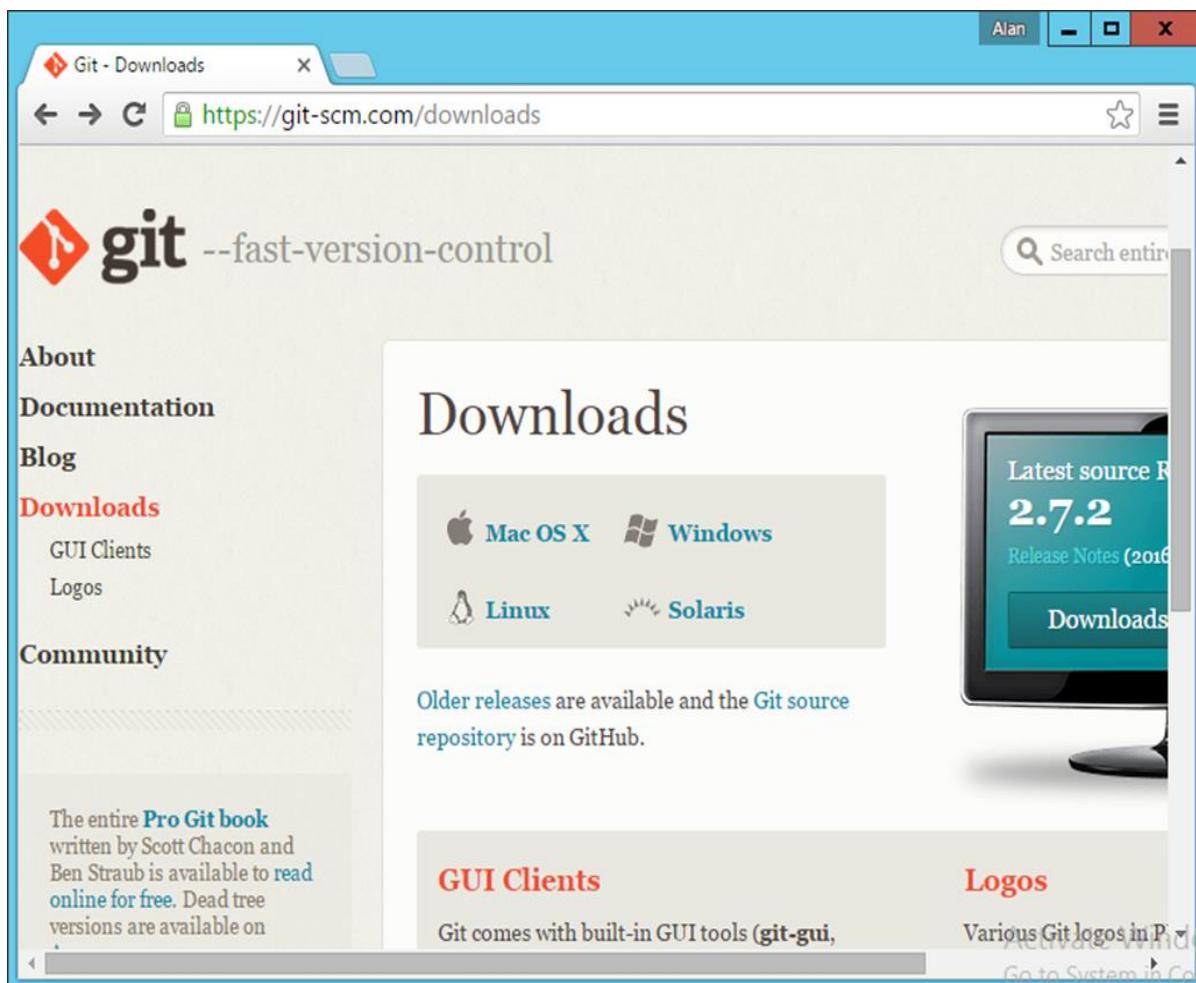
**Step 1:** The official website for Git is <https://git-scm.com/>. If you click on the link, you will get to the home page of the Git official website as shown in the following screenshot.



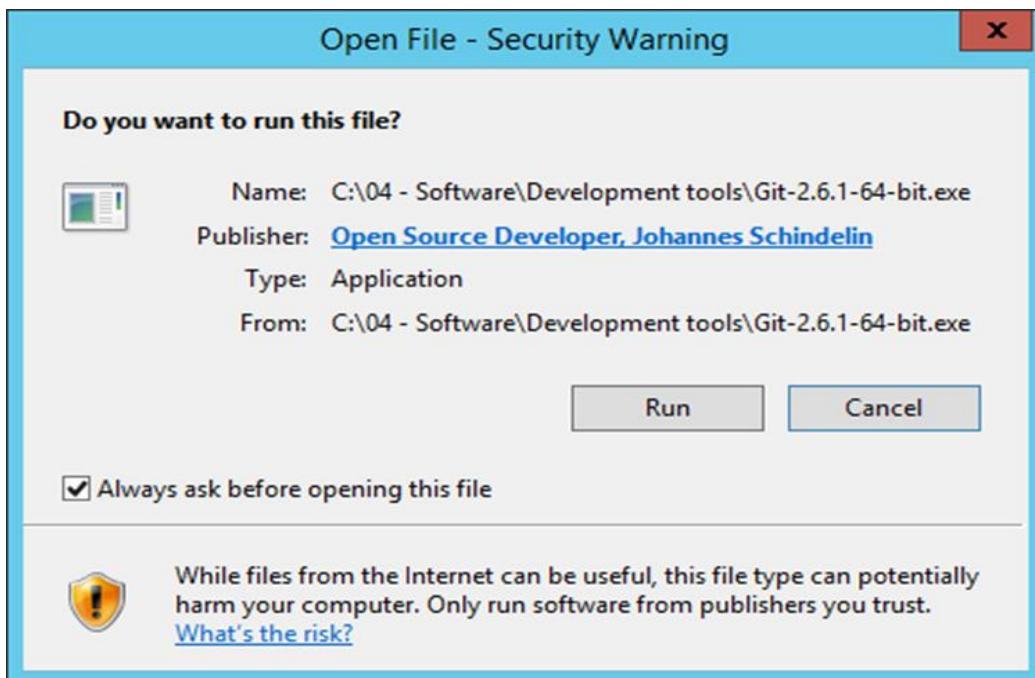
**Step 2:** To download Git, just scroll down the screen and go to the Downloads section and click Downloads.



**Step 3:** Click the Windows link and the download for Git will begin automatically.



**Step 4:** Click the downloaded .exe file for Git. In our case, we are using the Git-2.6.1-64-bit.exe file. Click Run which comes appears on the next screen.



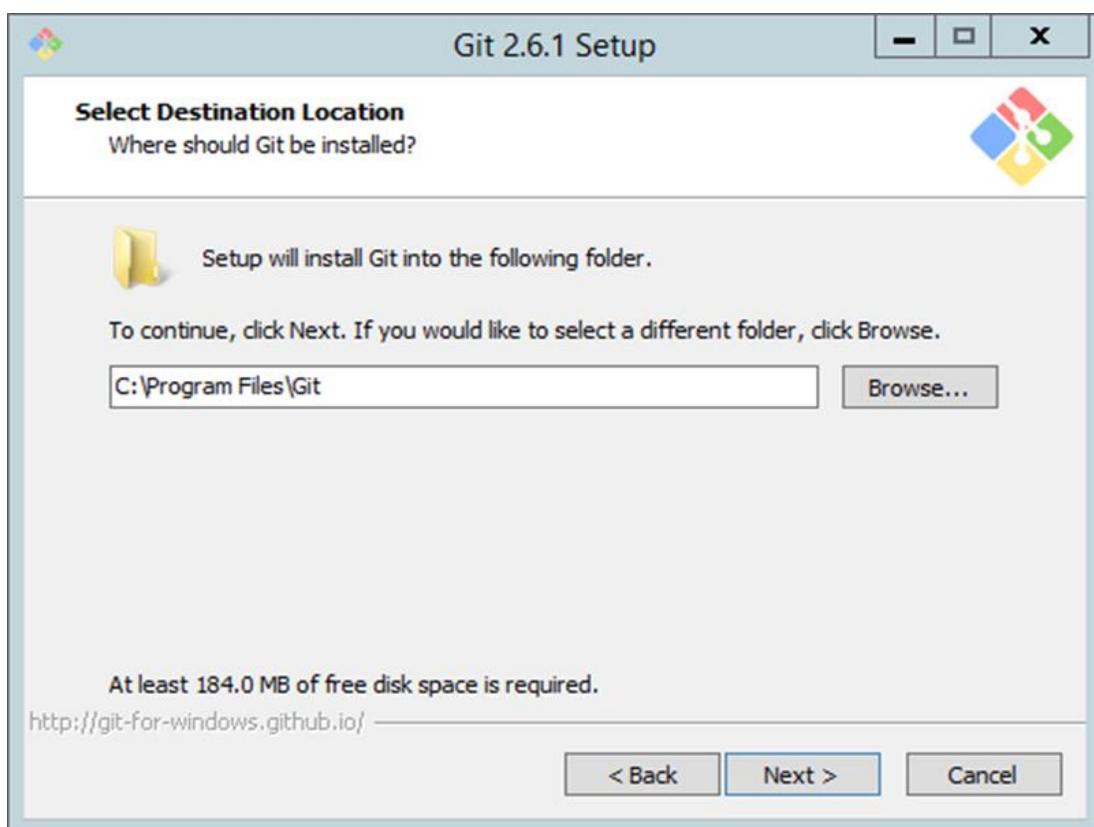
**Step 5:** Click the Next button that appears on the following screen.



**Step 6:** Click Next in the following screen to accept the General License agreement.



**Step 7:** Choose the location for your Git installation.



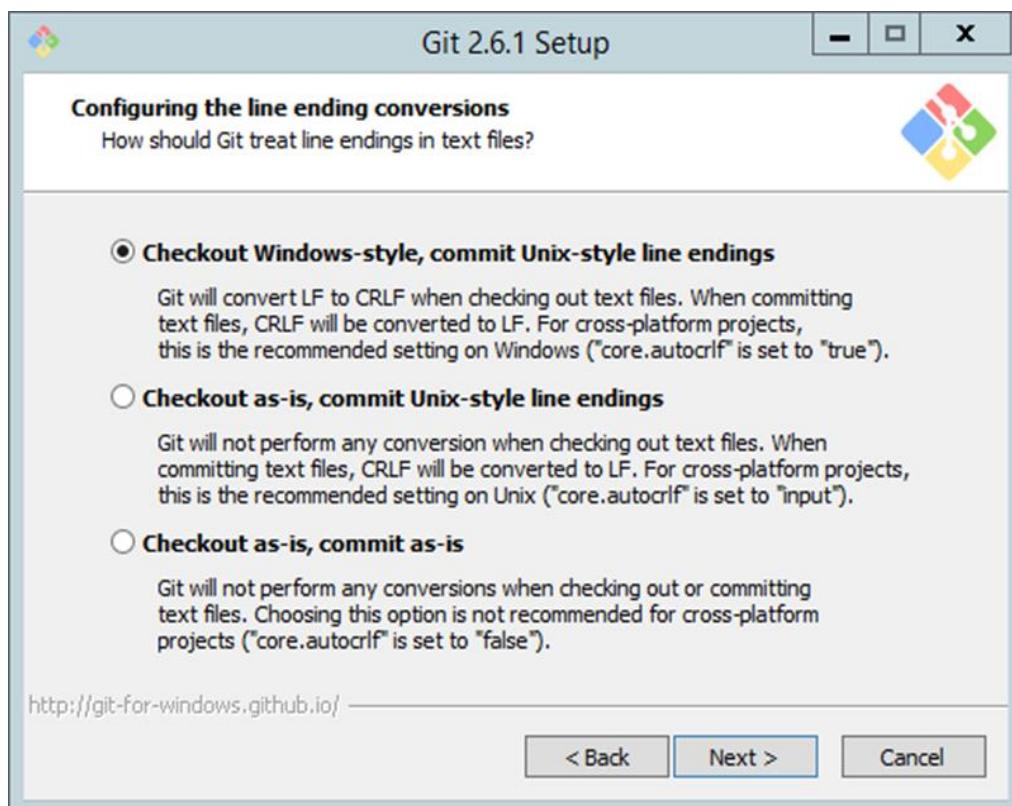
**Step 8:** Click Next to accept the default components that are need to be installed.



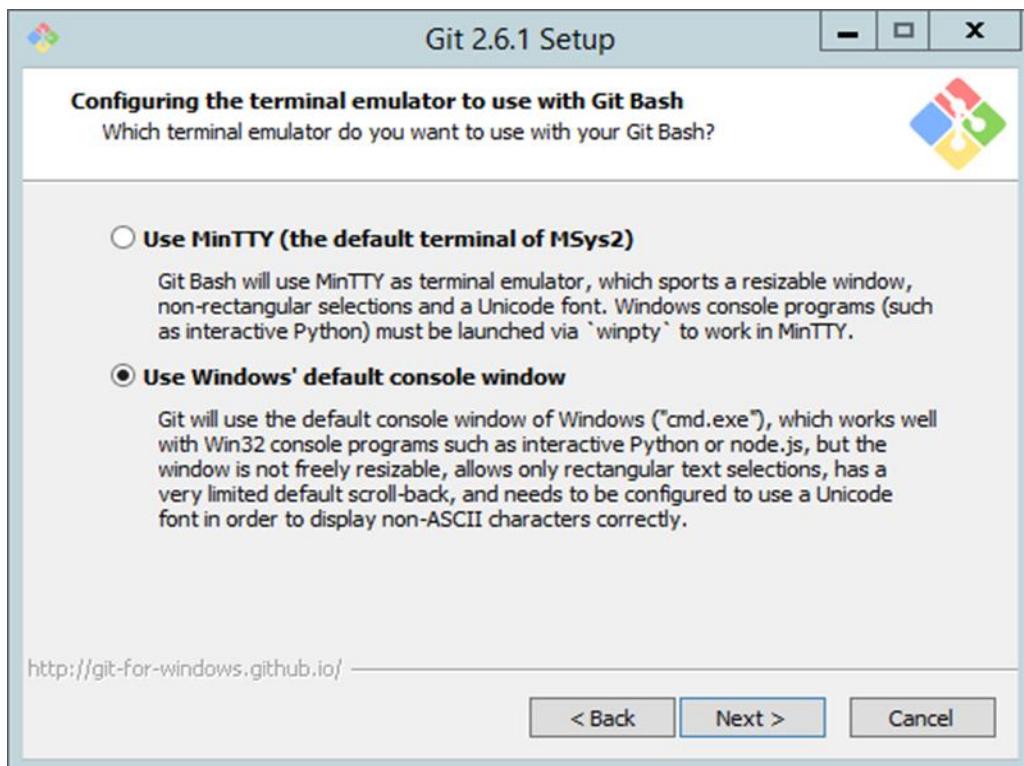
**Step 9:** Choose the option of 'Use Git from the Windows command prompt' since we are going to be using Git from Windows.



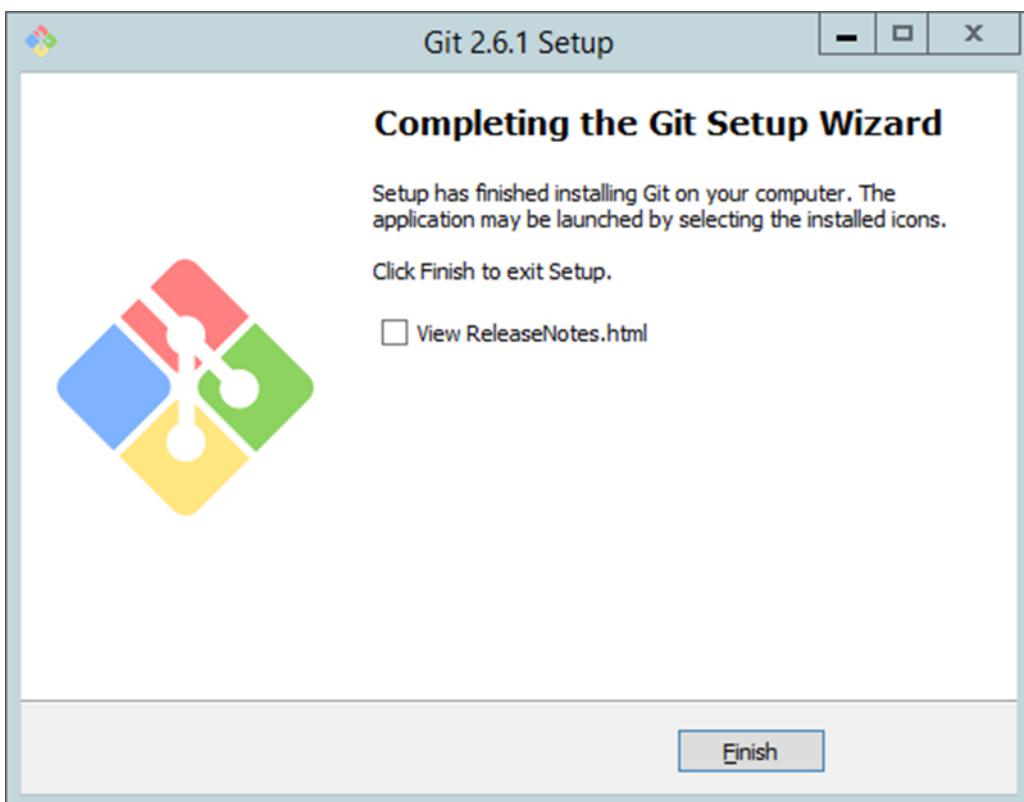
**Step 10:** In the following screen, accept the default setting of 'Checkout Windows-style, commit Unix-style line endings' and click Next.



**Step 11:** In the following screen, choose the option of 'Use Windows default console window', since we are using Windows as the system for installation of Git.



The installation will now start, and the subsequent steps can be followed for configuring Git, once the installation is complete.



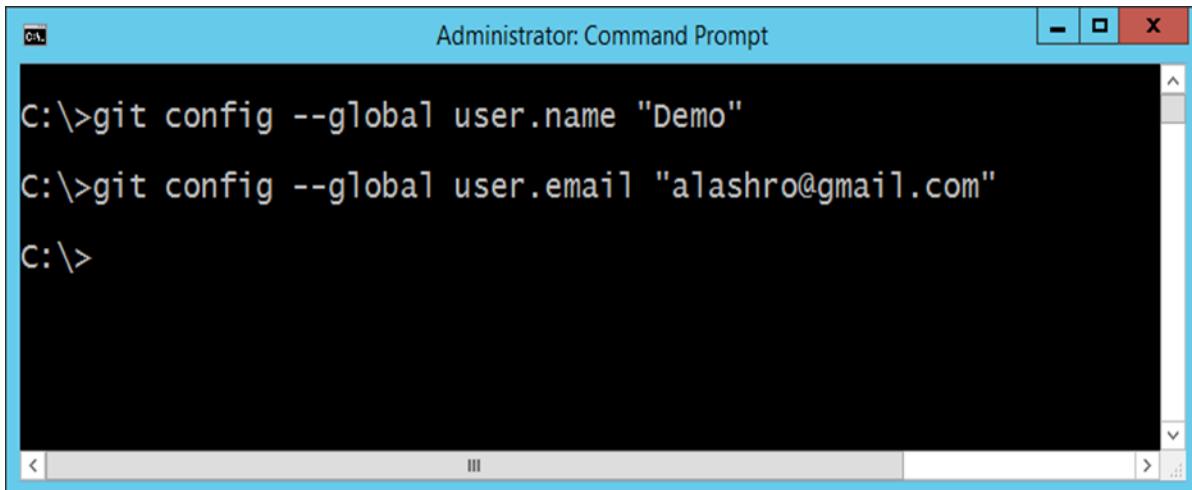
## Configuring Git

Once Git has been installed, the configuration steps need to be carried out for the initial configuration of Git.

The first thing that needs to be done is to configure the identity in Git and then to configure a user name and email. This is important because every **Git commit** uses this information, and it's immutably baked into the commits you start creating. One can do this by opening the command prompt and then enter the following commands –

```
git config -global user.name "Username"  
git config -global user.email "emailid"
```

The following screenshot is an example for better understanding.



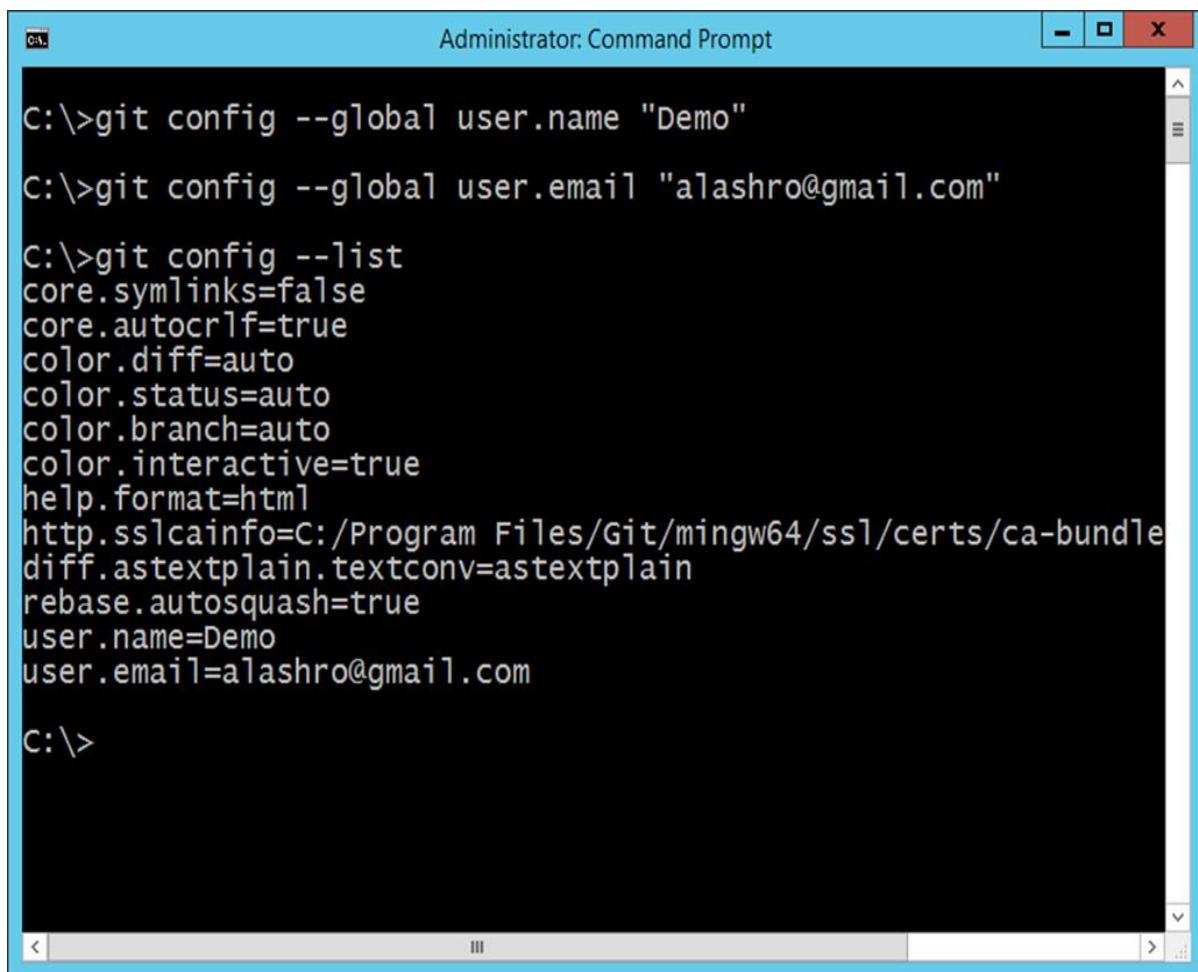
A screenshot of a Windows Command Prompt window titled "Administrator: Command Prompt". The window shows the following text being typed:

```
C:\>git config --global user.name "Demo"  
C:\>git config --global user.email "alashro@gmail.com"  
C:\>
```

These commands will actually change the configuration file of Git accordingly. To ensure your settings have taken effect, you can list down the settings of the Git configuration file by using issuing the following command.

```
git config --list
```

An example of the output is shown in the following screenshot.



```
C:\>git config --global user.name "Demo"
C:\>git config --global user.email "alashro@gmail.com"
C:\>git config --list
core.symlinks=false
core.autocrlf=true
color.diff=auto
color.status=auto
color.branch=auto
color.interactive=true
help.format=html
http.sslcainfo=C:/Program Files/Git/mingw64/ssl/certs/ca-bundle
diff.astextplain.textconv=astextplain
rebase.autosquash=true
user.name=Demo
user.email=alashro@gmail.com

C:\>
```

## Continuous Integration Server

The next crucial software required for the entire continuous integration pipeline is the Continuous Integration software itself. Following are the most commonly used Continuous Integration softwares used in the industry –

- **Jenkins** – This is an open source Continuous Integration software which is used by a lot of development communities.
- **JetBrains TeamCity** – This is one of the most popular commercial Continuous Integration software's available and most companies use this for their Continuous Integration needs.
- **Atlassian Bamboo** – This is another popular Continuous Integration software provided by a company called Atlassian Pvt. Ltd.

All of the softwares mentioned above, work on the same model for Continuous Integration. For the purpose of this tutorial, we will look at **Jetbrains TeamCity** for the Continuous Integration server.

## Installing TeamCity

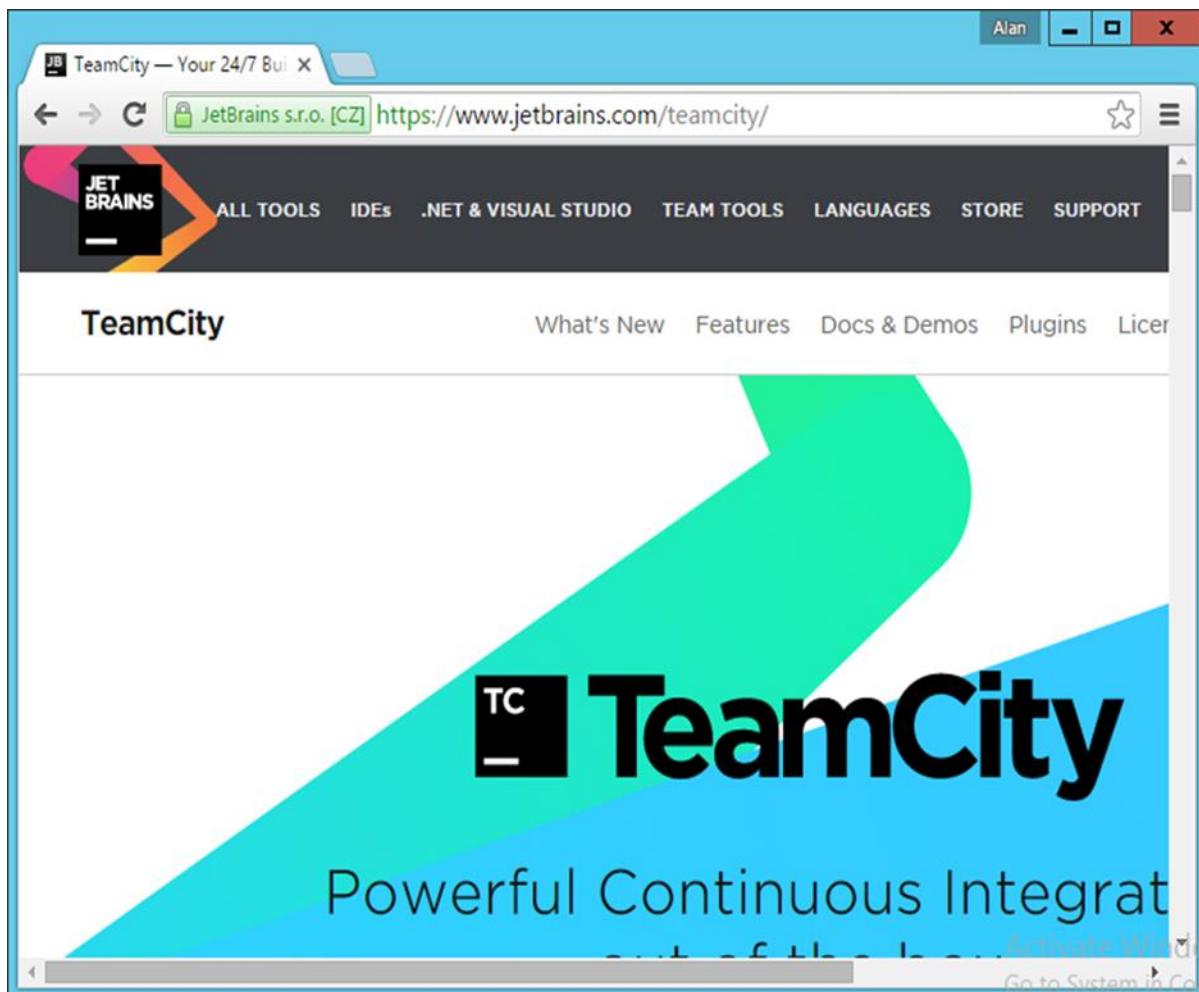
Following are the steps and the system requirements for installing Jet Brains TeamCity in your computer.

### System Requirements

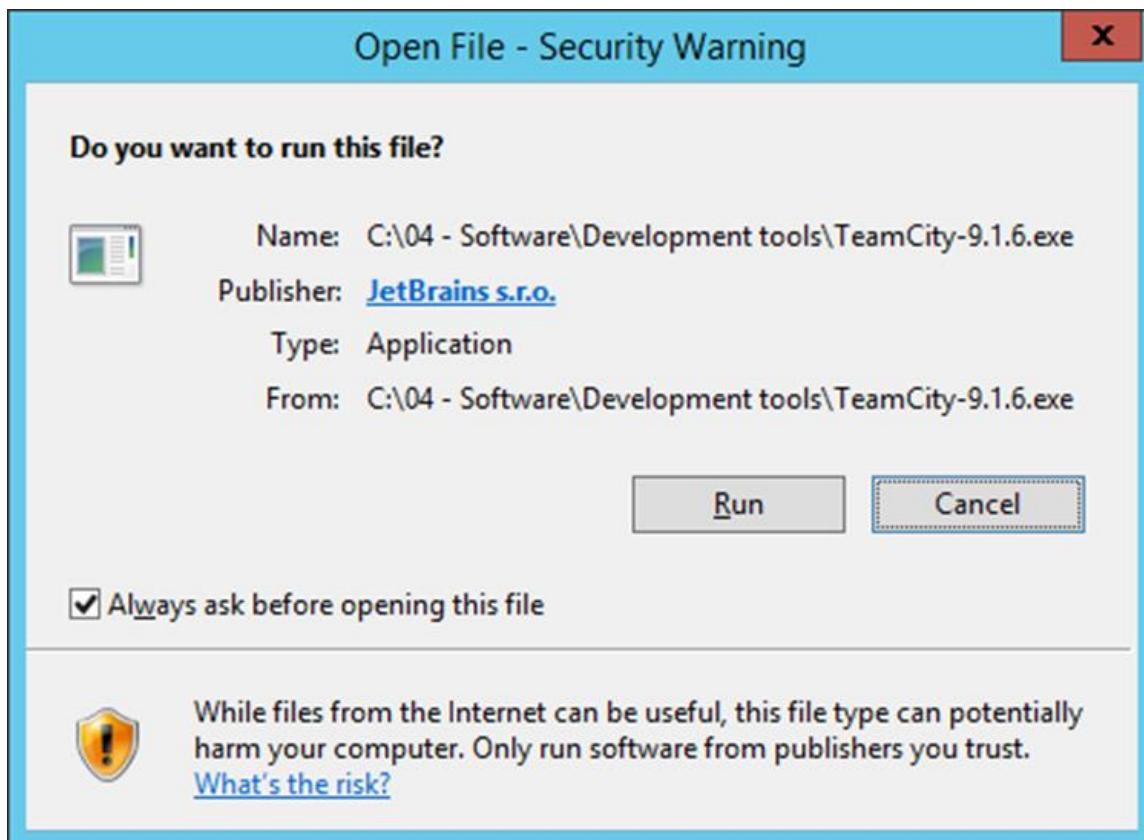
Memory	4 GB RAM (recommended)
Disk Space	1 GB HDD for the installation. Additional storage is required to store the build workspace for each project.
Operating System Version	Can be installed on Windows, Linux, Mac OS X.

### Installation

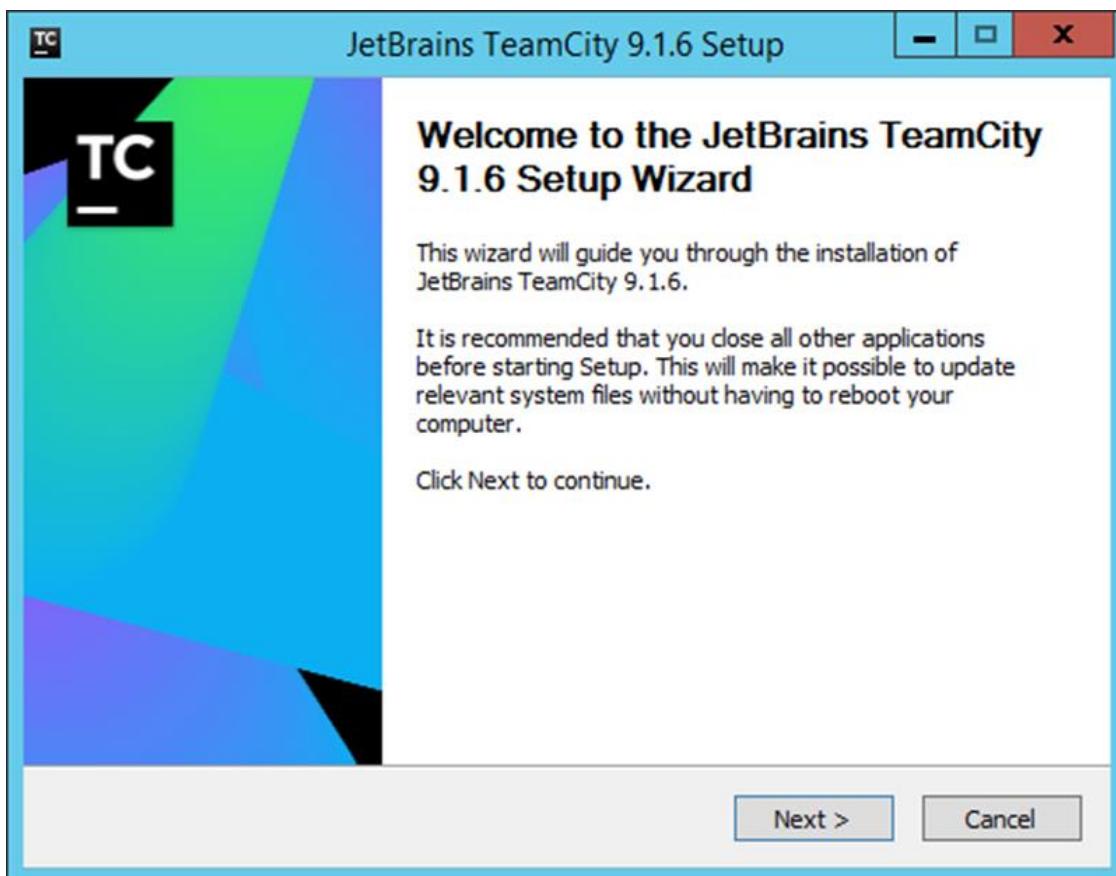
**Step 1:** The official website for TeamCity is <https://www.jetbrains.com/teamcity/>. If you click the given link, you will go to the home page of the TeamCity official website as shown in the following screenshot. You can browse the page to download the required software for TeamCity.



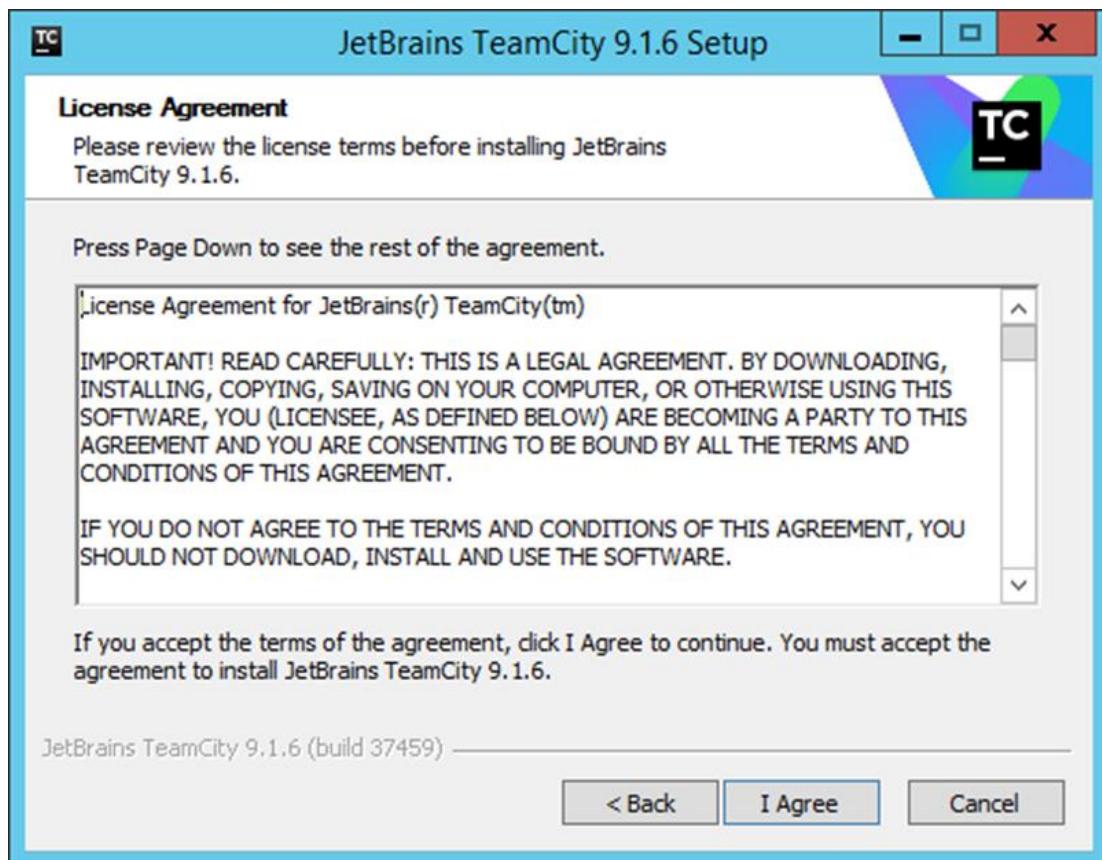
**Step 2:** The downloaded .exe is being used for the purpose of executing **TeamCity-9.1.6.exe**. Double-click the executable and then click Run in the next screen that pops up.



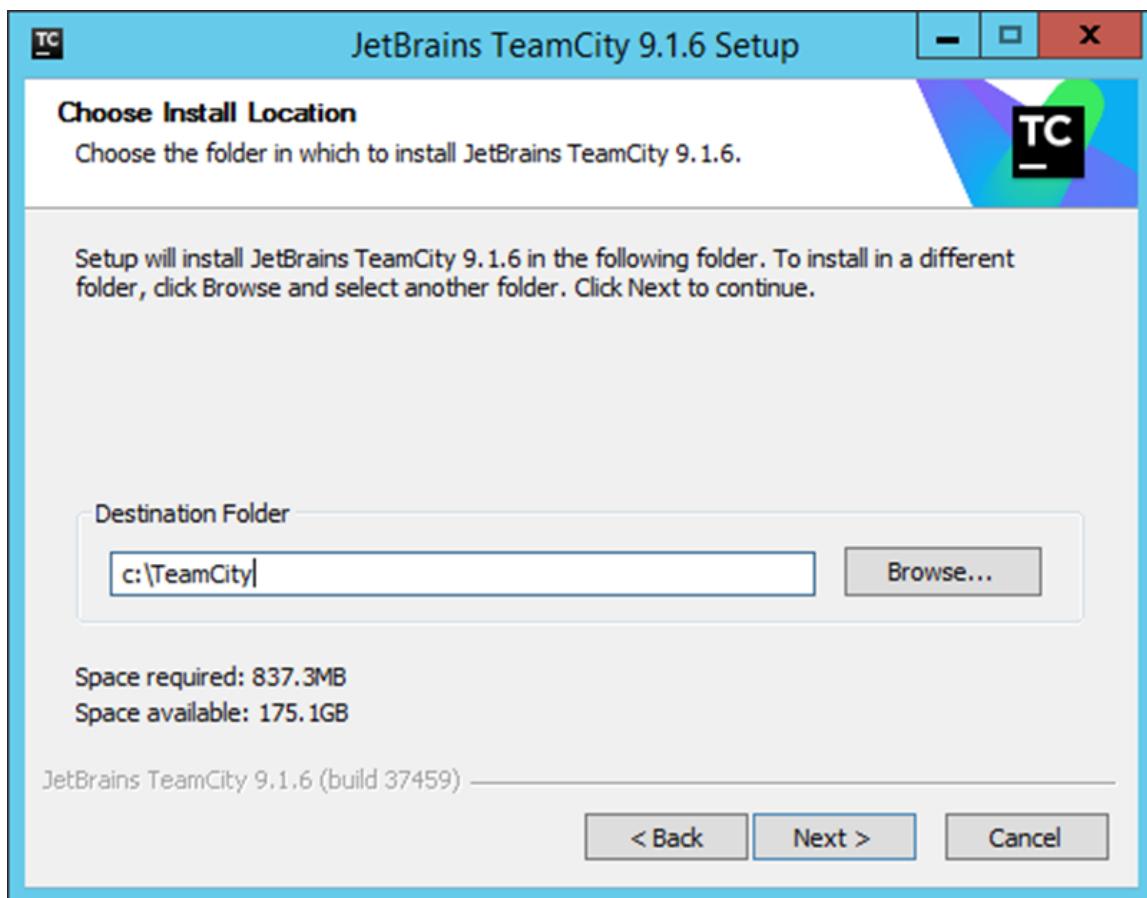
**Step 3:** Click Next to start the setup.



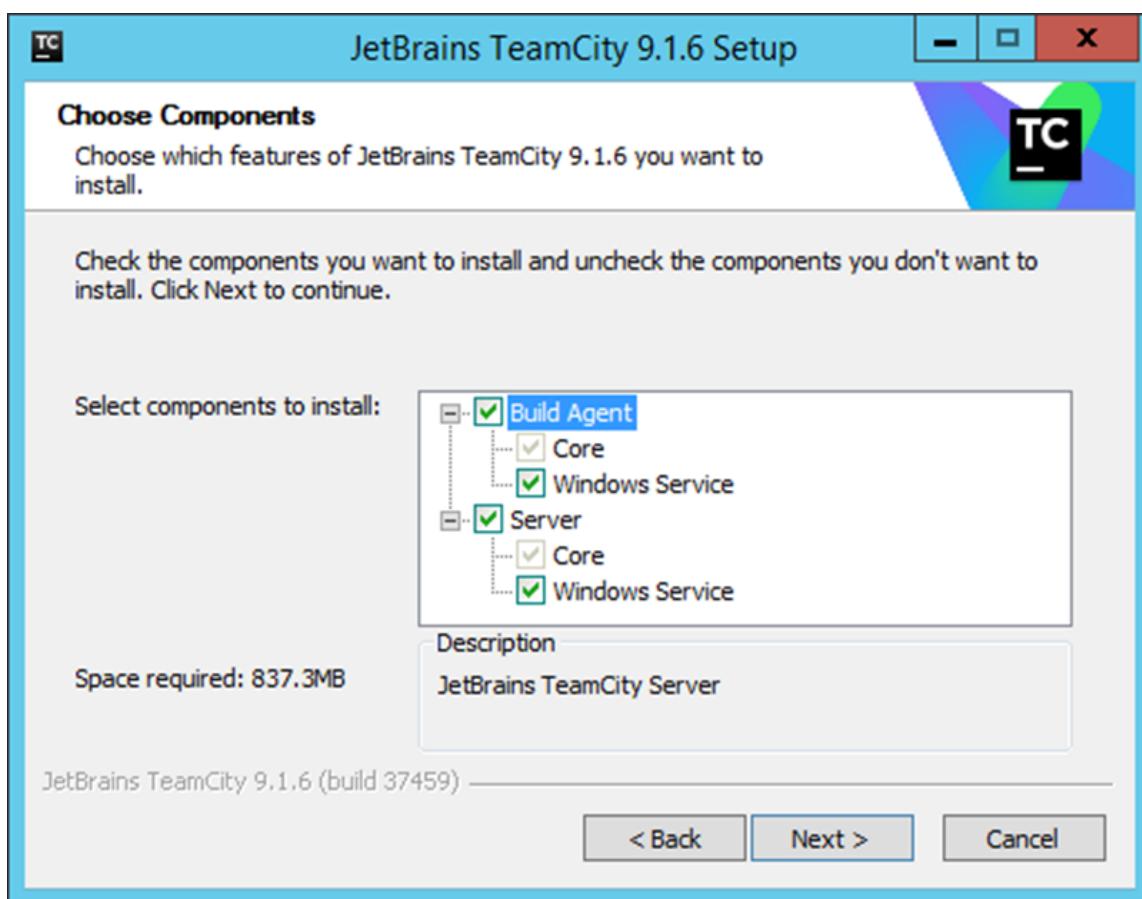
**Step 4:** Click the 'I Agree' button to accept the license agreement and proceed with the installation.



**Step 5:** Choose the location for the installation and click Next.

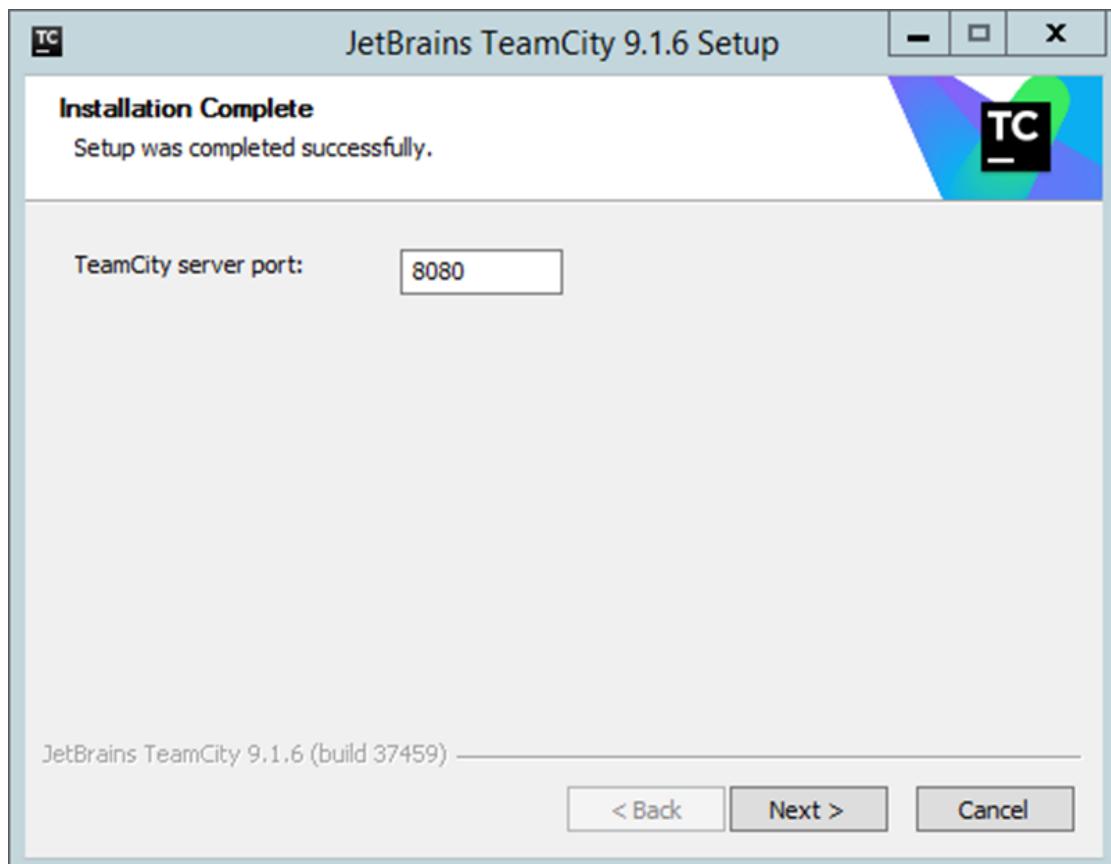


**Step 6:** Choose the default components for the installation and click Next.

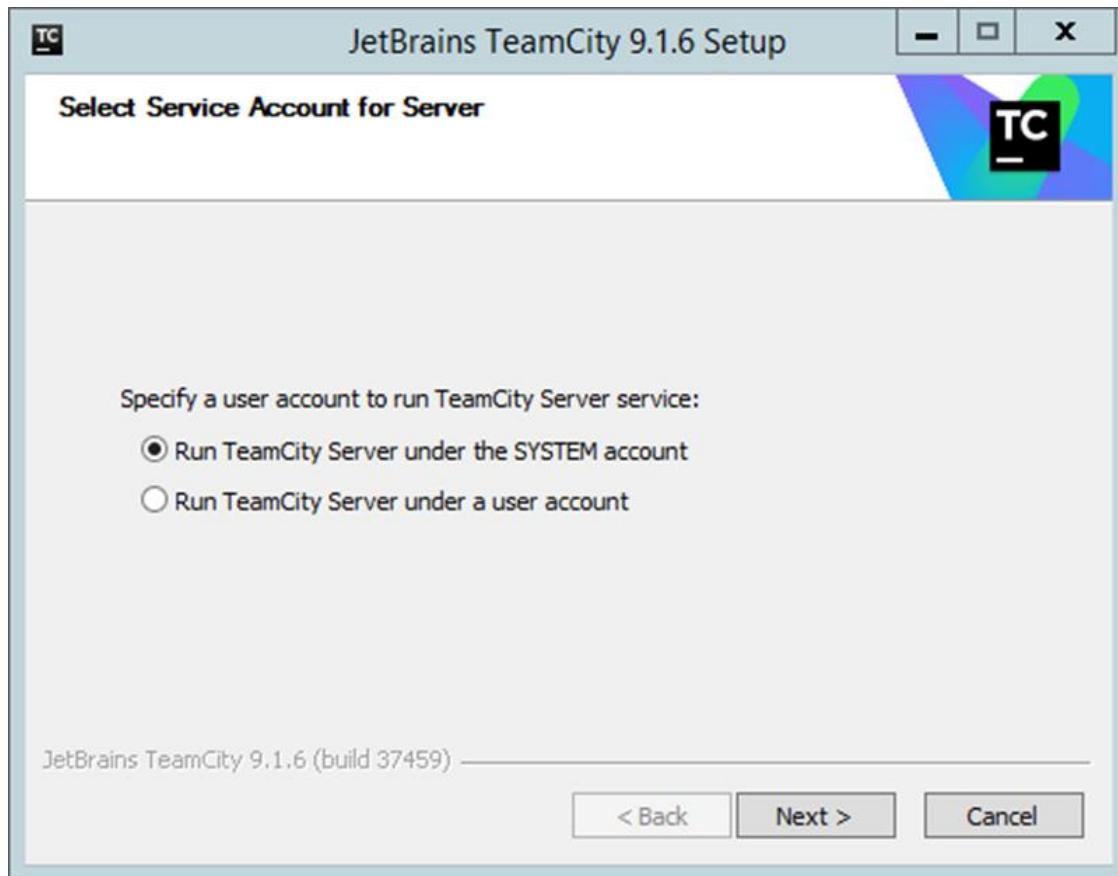


This will start the installation process. Once completed the configuration process will follow.

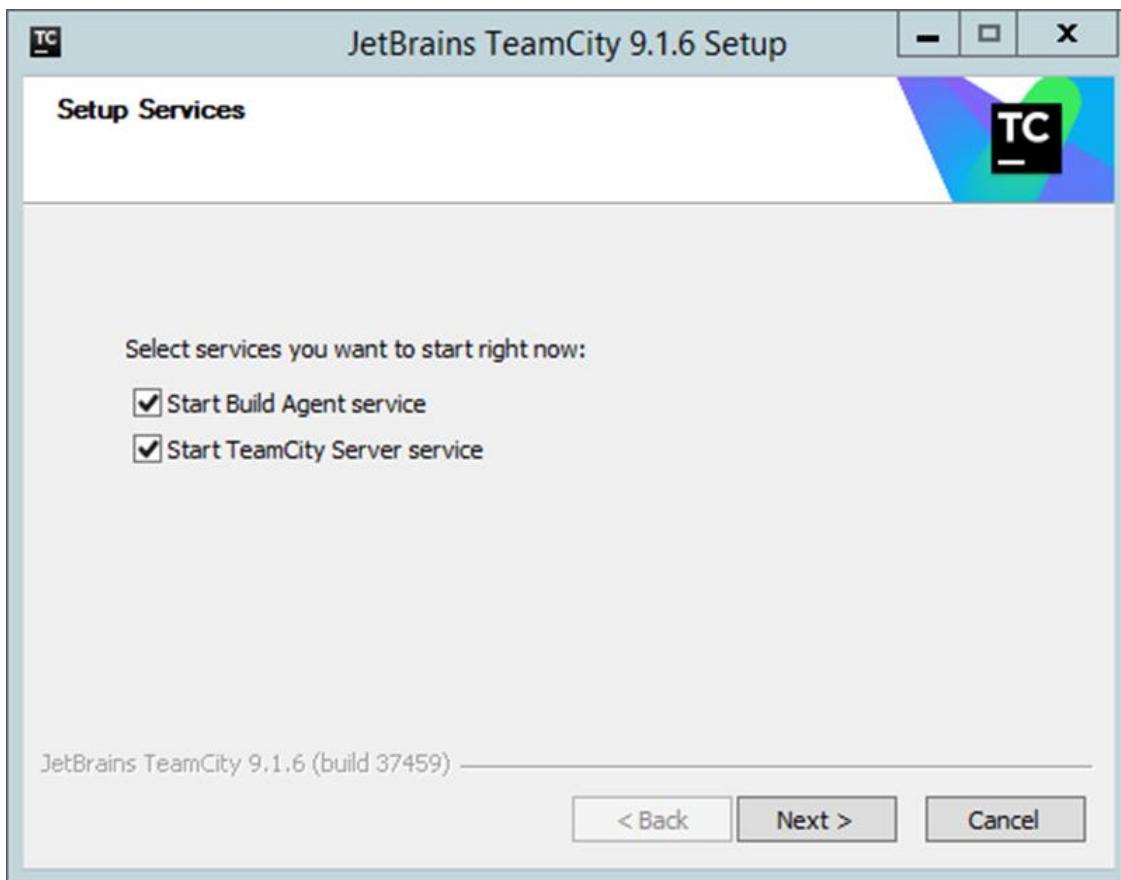
**Step 7:** Choose a port number for the server to run. Best is to use a different port such as **8080**.



**Step 8:** Next it will ask for which account TeamCity needs to run as. Choose the SYSTEM account and Click Next.



**Step 9:** Next it will ask for the services which needs to be started. Accept the default ones and then click Next.

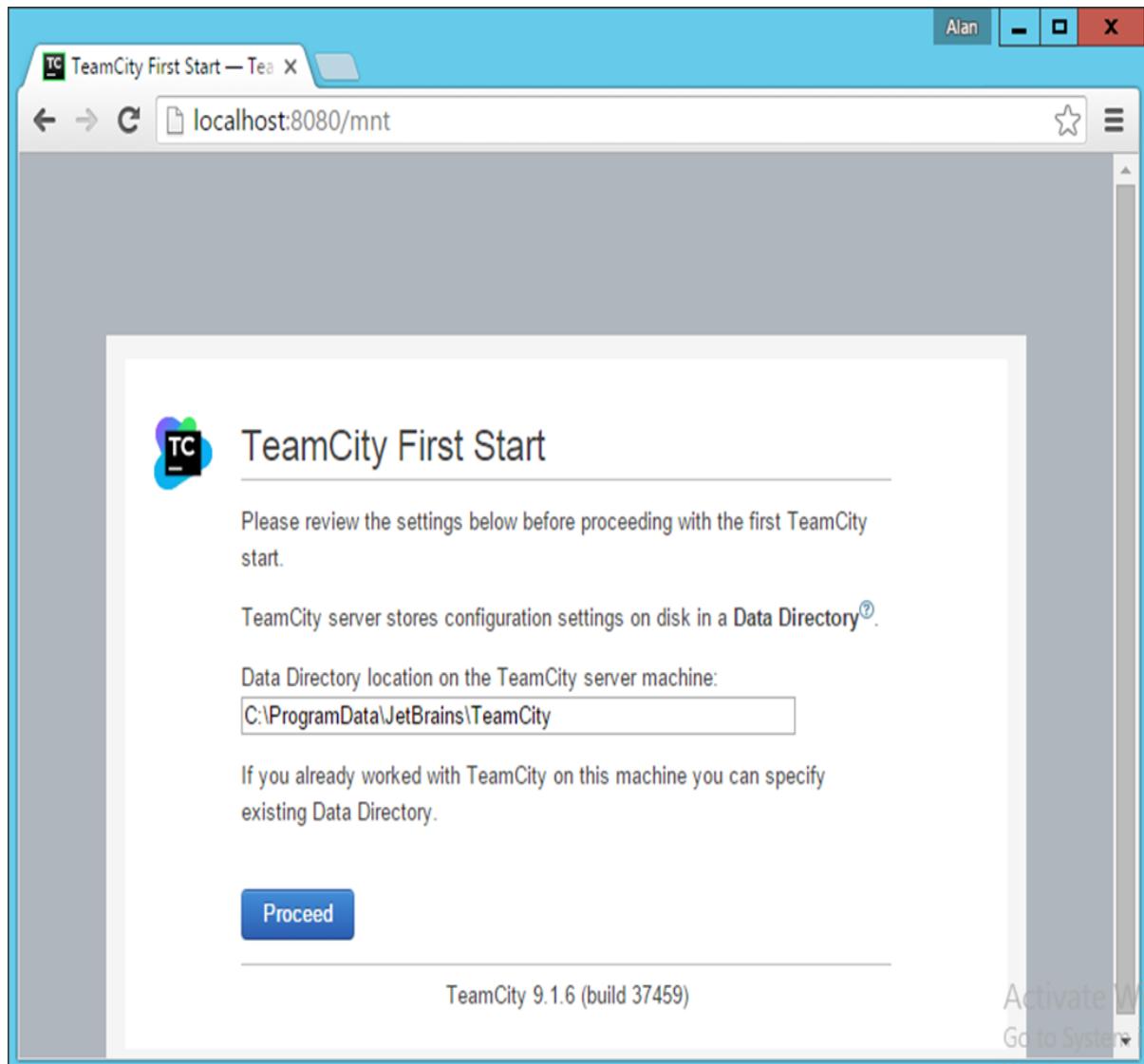


## Configuring TeamCity

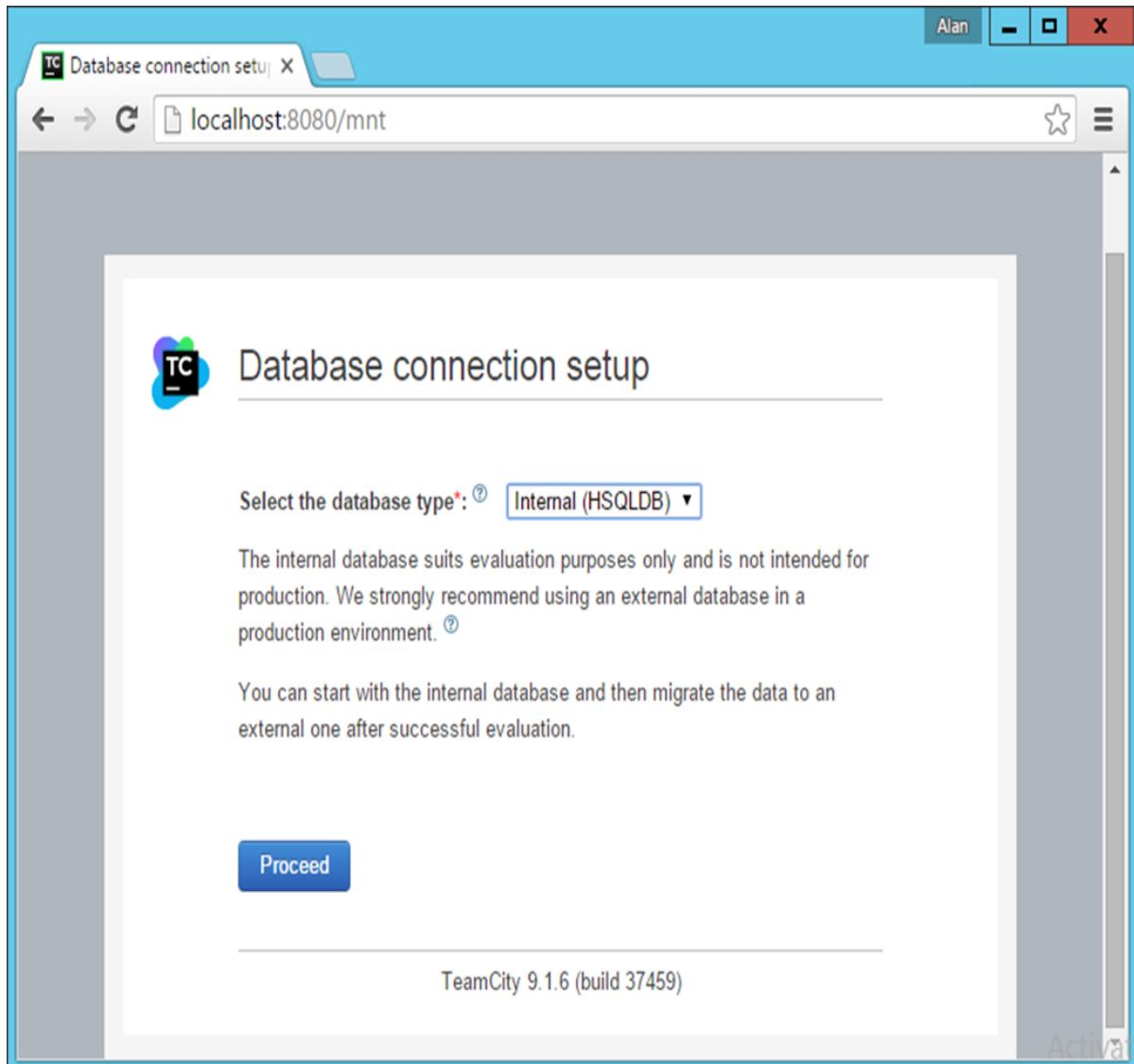
Once the installation is complete, the next step is the configuration of TeamCity. This software can be opened by browsing on the following URL in the browser –

<http://localhost:8080>

**Step 1:** The first step is to provide the location of the builds, which will be carried out by TeamCity. Choose the desired location and click the Proceed button.



**Step 2:** The next step is to specify the database for storing all the TeamCity artefacts. For the purpose of the tutorial, one can choose the **Internal (HSQLDB)**, which is an internal database that is best suited when using products for testing purposes.

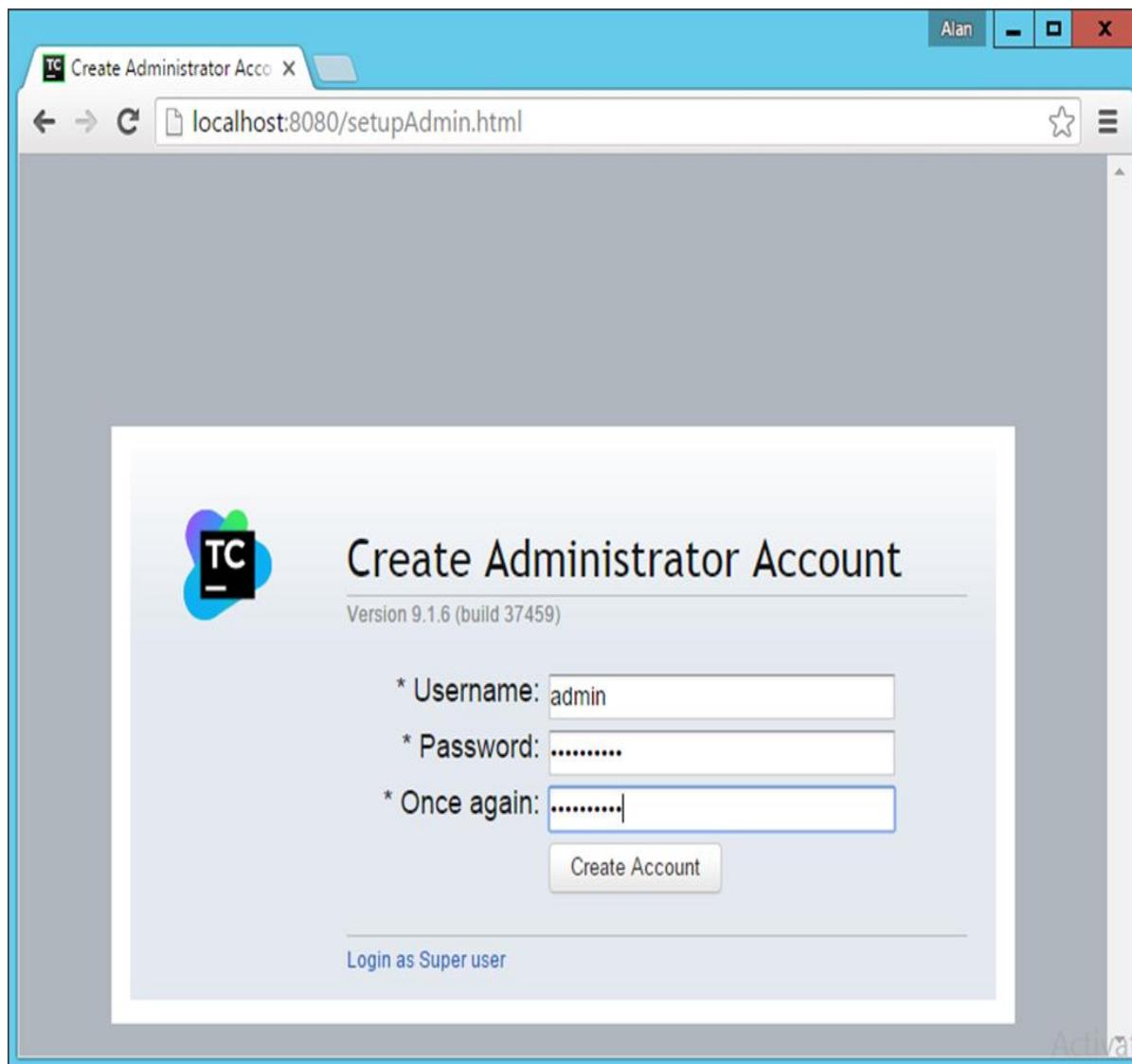


TeamCity will then process all the necessary steps to get it up and running.

**Step 3:** Next you will be requested to Accept the license agreement. Accept the same and click Continue.

The screenshot shows a web browser window titled "License Agreement — TeamCity". The URL in the address bar is "localhost:8080/showAgreement.html". The page content discusses license limitations for Enterprise Server and Professional Server Licenses, including restrictions on the number of Build Agents and build configurations. It also specifies limitations for Open Source Development, noting that the purpose of use shall be restricted to non-commercial open source projects. A message at the bottom states, "You must accept the license agreement to proceed." Below this, there are two checkboxes: one checked ("Accept license agreement") and one unchecked ("Send anonymous usage statistics to TeamCity development team (can be turned off at any time)"). A "Continue »" button is located at the bottom left of the form area.

**Step 4:** You need to create an administrator account that will be used to log into the TeamCity software. Enter the required details and click the 'Create Account' button.



You will now be logged into TeamCity.

The screenshot shows the TeamCity administration interface with the URL `localhost:8080/admin/admin.html?item=projects`. The main navigation bar includes 'Projects' (selected), 'Changes', 'Agents 1', 'Build Queue 0', and user 'admin'. Below the navigation is the breadcrumb 'Administration > Projects'. On the left, a sidebar lists 'Project-related Settings' with 'Projects' selected, and other options like 'Build Time', 'Disk Usage', 'Server Health', and 'Audit'. To the right, there are buttons for '+ Create project' and '+ Create project from URL'. A message states 'You have 2 active projects with 1 build configuration. You can have a maximum of 20 builds per day.' Below this is a 'Filter' input field and a 'Show archived' checkbox. The main content area shows a tree view of projects: '<Root project>' (Contains all other projects) expanded to show 'Demo'.

## The Build Tool

The Build tool is a tool which ensures that the program is built in a particular way. The tool will normally carry out a list of tasks, which are required for the program to be built in a proper manner. Since in our example, we are going to be looking at a **.Net program**, we will be looking at **MSBuild** as the build tool. The MSBuild tool looks at a build file which contains a list of tasks that are used to build the project. Let's look at a typical Build file for a web configuration project.

Following are the key sections of the Build file, which need to be considered.

### IIS Settings

Following settings are used to determine which is the port number, what is the path on the web server and what type of authentication is required when the application is run. These are important settings, which will be changed via the MSBuild command when we learn how the deployment will be carried out later on in the tutorial.

```
<UseIIS>True</UseIIS>
<AutoAssignPort>True</AutoAssignPort>
<DevelopmentServerPort>61581</DevelopmentServerPort>
<DevelopmentServerVPath>/</DevelopmentServerVPath>
<IISUrl>http://localhost:61581/</IISUrl>
<NTLMAuthentication>False</NTLMAuthentication>
```

## ItemGroup

This is used to tell the Build server what are all the dependent binaries that are required to run this project.

```
<ItemGroup>
  <Reference Include="System.Web.ApplicationServices" />
  <Reference Include="System.ComponentModel.DataAnnotations" />
```

```
<ItemGroup>
  <Compile Include="App_Start\BundleConfig.cs" />
  <Compile Include="App_Start\FilterConfig.cs" />
```

## .Net Framework Version

The **TargetFrameworkVersion** tells which is the version of .Net that needs to be present for the project to work. This is absolutely required because if the build server does not have this in place, the build will fail.

```
<TargetFrameworkVersion>v4.5</TargetFrameworkVersion>
```

## Deployment Environment – Amazon

For the purpose of this tutorial, we will ensure our Continuous Integration server has the ability to deploy our application to Amazon. For this, we need to ensure the following artefacts are in place.

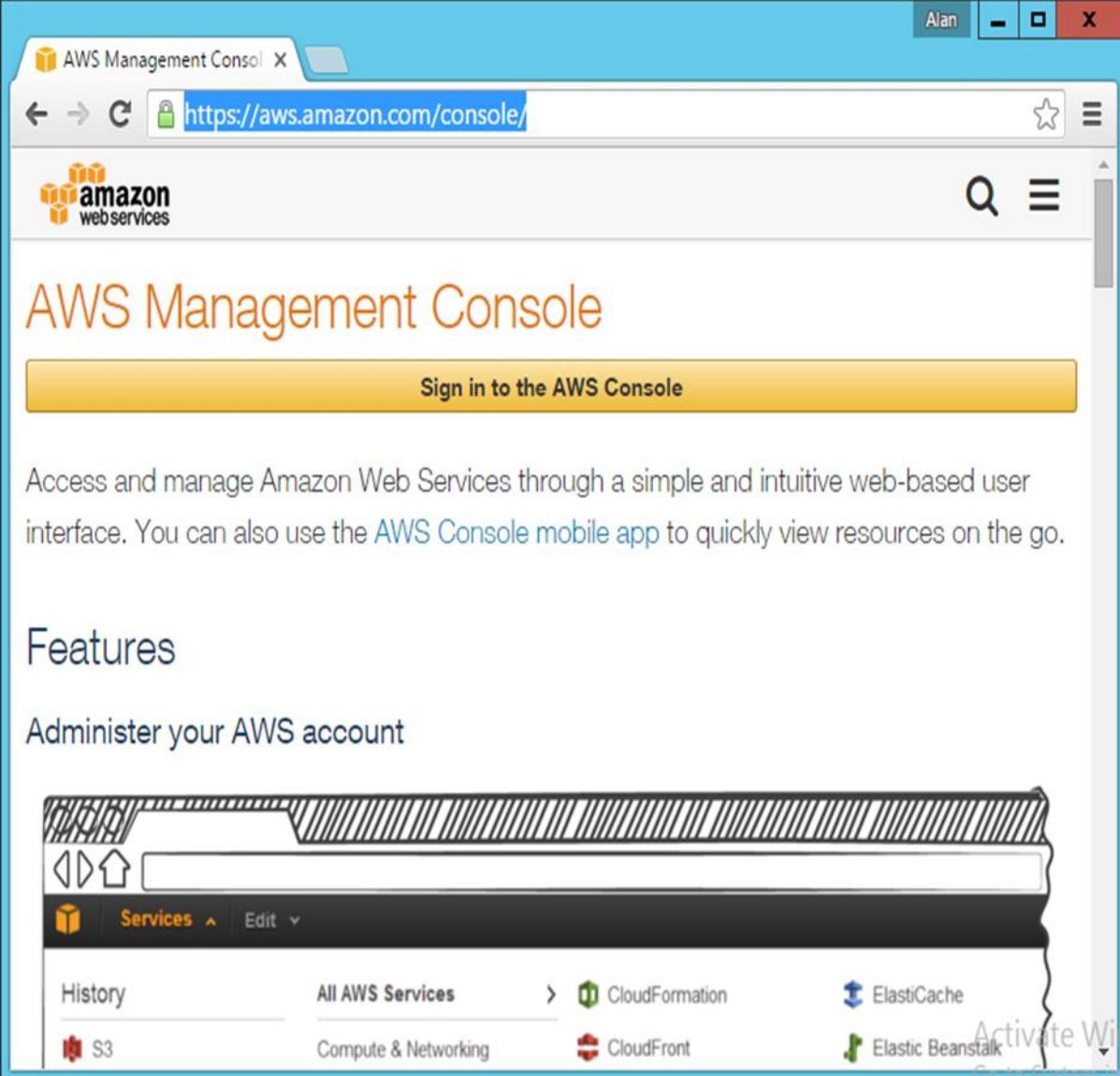
## Database Server

---

Perform the following steps to ensure that the database server is in place in Amazon for the deployment.

**Step 1:** Go to the Amazon Console - <https://aws.amazon.com/console/>

Login with your credentials. Note that you can apply for a free id on the amazon site, which will allow you to have a free tier that allows you to use some of the resources on Amazon free of cost.



The screenshot shows the AWS Management Console login page. At the top, there's a blue header bar with the AWS logo and the text "AWS Management Console". Below it is a browser-style interface with a back/forward button, a search bar containing the URL "https://aws.amazon.com/console/", and a magnifying glass icon. The main content area has a light orange header with the text "AWS Management Console" and a yellow button labeled "Sign in to the AWS Console". Below this, there's a descriptive paragraph about managing AWS services through the console or mobile app. The bottom section features a navigation bar with links for "Services", "Edit", "History", "All AWS Services", "CloudFormation", "ElastiCache", "S3", "Compute & Networking", "CloudFront", and "Elastic Beanstalk". A watermark for "Activate Wi-Fi" is visible on the right side of the page.

**Step 2:** Go to the RDS Section to create your database.

The screenshot shows the AWS Management Console homepage. The URL in the address bar is <https://ap-southeast-1.console.aws.amazon.com/console/home?region=ap-soutl>. The page is titled "AWS Management Console". On the left sidebar, under the "Database" section, the "RDS" service is listed as "Managed Relational Database Service". Other services listed in the sidebar include Glacier, Import/Export, Storage Gateway, Database (DynamoDB, ElastiCache, Redshift, DMS), Networking (VPC), and Application Services (Service Catalog, Trusted Advisor, API Gateway, AppStream, CloudSearch, Elastic Transcoder, SES, SQS, SWF). A "Service Health" section indicates that all services are operating normally. A banner for "AWS re:Invent Announcements" is visible on the right.

**Step 3:** Click Instances in the next screen that pops up.

The screenshot shows the AWS RDS Dashboard. The left sidebar lists navigation options: Instances, Reserved Purchases, Snapshots, Security Groups, Parameter Groups, Option Groups, Subnet Groups, Events, Event Subscriptions, and Notifications. The main content area features a large blue circular icon at the top, followed by the text "Amazon Relational Database Service". Below this, a descriptive paragraph reads: "Amazon Relational Database Service (Amazon RDS) makes it easy to set up, operate, and scale relational databases in the cloud. It provides cost-efficient and resizable". At the bottom of the page, there are links for Feedback, English, Privacy Policy, Terms of Use, and Activation Windows.

**Step 4:** Click the **Launch DB** option in the next screen that comes up.

The screenshot shows the AWS RDS Dashboard. The left sidebar lists various RDS management options: Instances, Reserved Purchases, Snapshots, Security Groups, Parameter Groups, Option Groups, Subnet Groups, Events, Event Subscriptions, and Notifications. The main content area has a header with 'Launch DB Instance' (which is highlighted in blue), 'Show Monitoring', and 'Instance Actions'. Below this is a search bar with 'Filter: All Instances' and a search input field. A message states 'No DB Instances' and provides filtering options for Engine, DB Instance, Status, CPU, Current Activity, Maintenance, and Class. A note at the bottom says 'Relational Database Service (RDS) is a web service that makes it easy to set up, operate, and scale a' and 'Note: Your DB Instances will launch in the Asia Pacific (Singapore) region'. At the bottom, there are links for Feedback, English, Privacy Policy, Terms of Use, and footer information including a copyright notice for 2008-2016 and a link to the AWS User Agreement.

**Step 5:** Choose the SQL Server tab and then choose the Select option for SQL Server Express.

The screenshot shows the AWS RDS console interface. At the top, there's a navigation bar with tabs for 'AWS', 'Services', and 'Edit'. On the right, it shows 'Alan' as the user, 'Singapore' as the region, and a 'Support' link. The main content area has a title 'Step 1: Select Engine' and a sub-instruction 'To get started, choose a DB Engine below and click Select.' Below this, there's a list of database engines with their respective logos and 'Select' buttons:

DB Engine	Description	Select Button
MySQL	Microsoft SQL Server Express Edition	Select
MariaDB	Microsoft SQL Server Express Edition is an affordable database management system that supports database sizes up to 10 GB. Refer to Microsoft's web site for more details.	Select
PostgreSQL	Microsoft SQL Server Web Edition	Select
ORACLE	Microsoft SQL Server Web Edition is an efficient and affordable database management system. In accordance with Microsoft's licensing policies, it can only be used to support public and Internet-accessible webpages, websites, web applications, and web services. Refer to the AWS Service	Select

At the bottom of the page, there are links for 'Feedback', 'English', 'Activate Windows', 'Privacy Policy', and 'Terms of Use'.

**Step 6:** Ensure that the following details are entered to confirm that you are using the free tier of databases available from Amazon.

The screenshot shows the AWS RDS console interface for launching a new database instance. The top navigation bar includes 'AWS Services' and 'Edit'. The main configuration area is titled 'Launch New DB Instance'.

**DB Engine:** sqlserver-ex

**License Model:** license-included

**DB Engine Version:** 12.00.4422.0.v1

**DB Instance Class:** db.t2.micro — 1 vCPU, 1 GiB RAM

A note states: "Scaling storage after launching a DB Instance is currently not supported for SQL Server. You may want to provision storage based on anticipated future storage growth."

**Storage Type:** General Purpose (SSD)

**Allocated Storage\***: 20 GB

**Settings**

**DB Instance Identifier\***: (empty field)

**Master Username\***: (empty field)

At the bottom, there are links for 'Feedback', 'English', 'Activate Windows', 'Privacy Policy', and 'Terms of Use'.

**Step 7:** Click the Next Step button once all the fields are filled.

The screenshot shows the AWS RDS console interface for creating a new database instance. The page title is "RDS - AWS Console". The URL in the address bar is <https://ap-southeast-1.console.aws.amazon.com/rds/home?region=ap-southeast-1#launch-dbinstance:create>. The top navigation bar includes "AWS", "Services", "Edit", "Alan", "Singapore", and "Support".

The main form is titled "Settings". It contains the following fields:

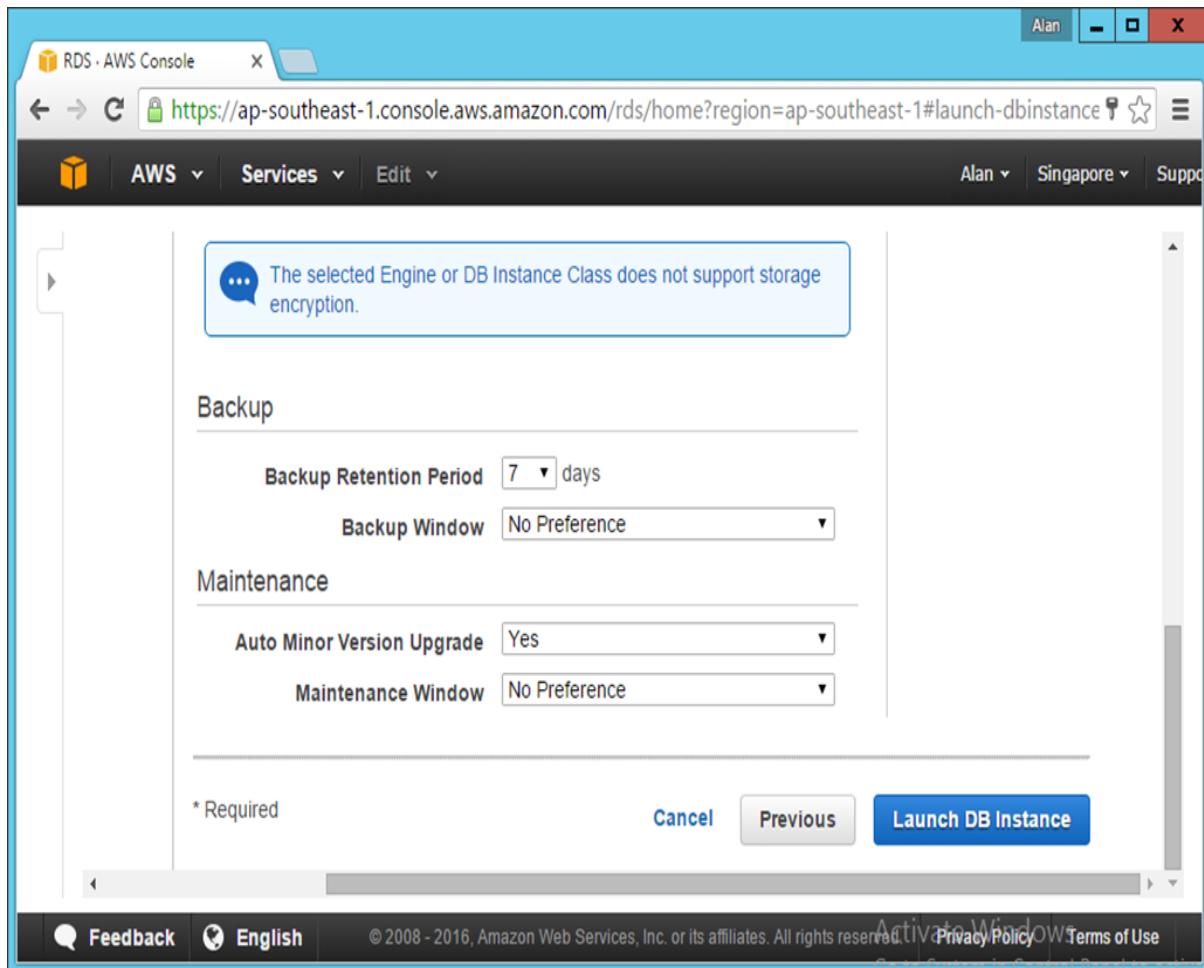
- Storage Type: General Purpose (SSD)
- Allocated Storage\*: 20 GB
- DB Instance Identifier\*: SampleDB
- Master Username\*: demouser
- Master Password\*: ..... (password masked)
- Confirm Password\*: ..... (password masked)

A tooltip on the right side of the master password field says: "Retype the value you specified for Master Password."

At the bottom left, there is a note: "\* Required". At the bottom right, there are three buttons: "Cancel", "Previous", and "Next Step".

The footer of the page includes links for "Feedback", "English", "Activate Windows", "Privacy Policy", and "Terms of Use". The copyright notice at the bottom is "© 2008 - 2016, Amazon Web Services, Inc. or its affiliates. All rights reserved."

**Step 8:** In the next screen that comes up, accept all the default settings and Click **Launch DB Instance**.



**Step 9:** You will then be presented with a screen that says that the DB is being successfully launched. On the same page, there will be a button to view the DB Instance. Click the link to see your **DB Instance** being set up.

The screenshot shows the AWS RDS console interface. At the top, the URL is https://ap-southeast-1.console.aws.amazon.com/rds/home?region=ap-southeast-1#launch-dbinstance. The main content area displays a green success message: "Your DB Instance is being created." Below it, a note says "Note: Your instance may take a few minutes to launch." To the left, a sidebar lists steps: "Select Engine", "Specify DB Details", and "Configure Advanced Settings". Below the message, a section titled "Connecting to your DB Instance" contains a note about security group access and a link to "Go to the Security Groups Page". A "Related AWS Services" section includes a link to "Amazon ElastiCache". The bottom of the page features standard AWS navigation links for Feedback, English, Privacy Policy, Terms of Use, and Activation Windows.

The screenshot shows the AWS RDS Dashboard. On the left, there's a sidebar with options like Instances, Reserved Purchases, Snapshots, Security Groups, Parameter Groups, Option Groups, Subnet Groups, Events, Event Subscriptions, and Notifications. The main area has tabs for Launch DB Instance, Show Monitoring, and Instance Actions. A search bar at the top right says "Search DB Instances...". Below it, a message says "Viewing 1 of 1 DB Instances". A table lists one instance: Engine is SQL Server Express, DB Instance is sampledb, Status is creating, and Maintenance is None.

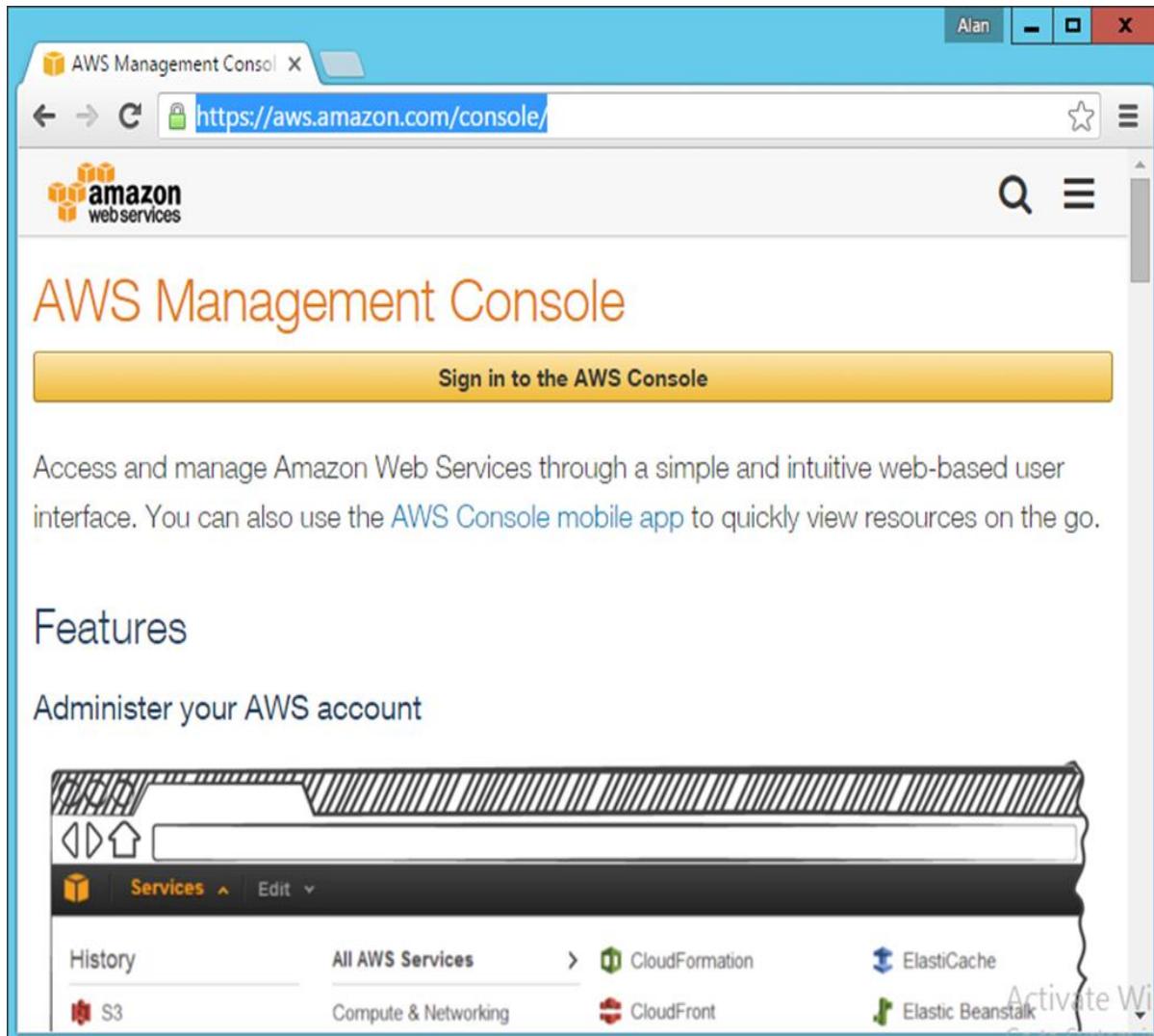
After some time, the status of the above screen will change to notify that the DB Instance has been successfully created.

## Web Server

The next step is to create your web server on Amazon, which will host the web application. This can be done by following the subsequent steps to have this in place.

**Step 1:** Go to Amazon Console - <https://aws.amazon.com/console/>

Login with your credentials. Note that you can apply for a **free id on the Amazon site**, which will allow you to have a free tier that allows you to use some of the resources on Amazon free of cost.



The screenshot shows a web browser window titled "AWS Management Console". The address bar displays the URL "https://aws.amazon.com/console/". The main content area is titled "AWS Management Console" and features a yellow button labeled "Sign in to the AWS Console". Below this, a descriptive text reads: "Access and manage Amazon Web Services through a simple and intuitive web-based user interface. You can also use the AWS Console mobile app to quickly view resources on the go." At the bottom of the page, there is a navigation bar with links for "Services", "Edit", "History", "All AWS Services", "CloudFormation", "CloudFront", "ElastiCache", "Compute & Networking", "Elastic Beanstalk", and "S3".

**Step 2:** Go to the **EC2 section** to create your web server.

The screenshot shows the AWS Management Console interface. At the top, there's a navigation bar with tabs for 'AWS' and 'Services'. Below the navigation bar, the main content area is titled 'Amazon Web Services' and lists various AWS services under different categories. On the right side, there's a sidebar titled 'Resource Groups' with a description and buttons for 'Create a Group' and 'Tag Editor'. Below the sidebar, there are sections for 'Additional Resources' and links to 'Getting Started', 'AWS Console Mobile App', and 'AWS Marketplace'.

Compute	Developer Tools	Internet of Things
EC2 Virtual Servers in the Cloud	CodeCommit Store Code in Private Git Repositories	AWS IoT Connect Devices to the Cloud
EC2 Container Service Run and Manage Docker Containers	CodeDeploy Automate Code Deployments	Game Development
Elastic Beanstalk Run and Manage Web Apps	CodePipeline Release Software using Continuous Delivery	GameLift Deploy and Scale Session-based Multiplayer Games
Lambda Run Code in Response to Events	Management Tools	Mobile Services
Storage & Content Delivery	CloudWatch Monitor Resources and Applications	Mobile Hub Build, Test, and Monitor Mobile Apps
S3 Scalable Storage in the Cloud	CloudFormation Create and Manage Resources with Templates	Cognito User Identity and App Data Synchronization
CloudFront Global Content Delivery Network	CloudTrail Track User Activity and API Usage	Device Farm Test Android, FireOS, and iOS Apps on Real Devices in the Cloud
Elastic File System PREVIEW Fully Managed File System for EC2	Config Track Resource Inventory and Changes	Mobile Analytics Collect, View and Export App Analytics
Glacier Archive Storage in the Cloud	OpsWorks	SNS
Import/Export Snowball		

**Step 3:** In the next screen, click Launch Instance.

The screenshot shows the AWS EC2 Management Console interface. The left sidebar has sections for EC2 Dashboard, Instances, Images, and Elastic Block Store. The main content area is titled 'Create Instance' with a sub-section 'Placement Groups'. It includes a note about deploying applications with Elastic Beanstalk, a 'Launch Instance' button, and service status information for Asia Pacific (Singapore). On the right, there's an 'Additional Information' sidebar with links to Getting Started Guide, Documentation, All EC2 Resources, Forums, Pricing, and Contact Us. Below that is an 'AWS Marketplace' section with a note about finding software trials.

Easily deploy Ruby, PHP, Java, .NET, Python, Node.js & Docker applications with [Elastic Beanstalk](#).

**Create Instance**

To start using Amazon EC2 you will want to launch a virtual server, known as an Amazon EC2 instance.

**Launch Instance**

Note: Your instances will launch in the Asia Pacific (Singapore) region

Service Status:	Asia Pacific (Singapore):
Asia Pacific (Singapore): This service is operating	No events

**Scheduled Events**

**AWS Marketplace**

Find [free software trial](#) products in the AWS Marketplace from the EC2 Launch Wizard. Or try these popular AMIs:

**Step 4:** Click Windows – **Microsoft Windows Server 2010 R2 Base.**

EC2 Management Console

https://ap-southeast-1.console.aws.amazon.com/ec2/v2/home?region=ap-southeast-1#LaunchInstanceW

AWS Services Edit Alan Singapore Support

1. Choose AMI    2. Choose Instance Type    3. Configure Instance    4. Add Storage    5. Tag Instance    6. Configure Security Group

Step 1: Choose an Amazon Machine Image (AMI)

**Free tier eligible** Ubuntu Server 14.04 LTS (HVM), EBS General Purpose (SSD)  
Volume Type. Support available from Canonical (<http://www.ubuntu.com/cloud/services>).  
Root device type: ebs    Virtualization type: hvm

**Windows** Microsoft Windows Server 2012 R2 Base - ami-9801cffb  
**Free tier eligible** Microsoft Windows 2012 R2 Standard edition with 64-bit architecture. [English]  
Root device type: ebs    Virtualization type: hvm

Select    64-bit

Are you launching a database instance? Try Amazon RDS. Hide

Feedback English © 2008 - 2016, Amazon Web Services, Inc. or its affiliates. All rights reserved. Activate Windows Privacy Policy Terms of Use

**Step 5:** Choose the **t2.micro** option, which is a part of the free tier. Click **Next: Configure Instance Details**.

The screenshot shows the AWS EC2 Management Console interface. The top navigation bar includes the AWS logo, Services dropdown, Edit dropdown, user name Alan, region Singapore, and Support. Below the navigation is a progress bar with six steps: 1. Choose AMI, 2. Choose Instance Type (which is currently selected), 3. Configure Instance, 4. Add Storage, 5. Tag Instance, and 6. Configure Security Group. The main content area is titled "Step 2: Choose an Instance Type". A table lists various instance types with their details: General purpose (t2.nano, t2.micro, t2.small, t2.medium, t2.large) and Compute optimized (m4.large). The t2.micro row is highlighted with a green background and has a "Free tier eligible" badge. The table columns include CPU, RAM, EBS only, and Free. At the bottom of the table are "Cancel", "Previous", "Review and Launch" (which is highlighted in blue), and "Next: Configure Instance Details". The footer contains links for Feedback, English, Privacy Policy, Terms of Use, and Activate Windows.

		(CPU)	(RAM)	EBS only	Free
<input type="checkbox"/>	General purpose t2.nano	1	0.5	EBS only	-
<input checked="" type="checkbox"/>	General purpose t2.micro <span style="background-color: #00AEEF; color: white; padding: 2px;">Free tier eligible</span>	1	1	EBS only	-
<input type="checkbox"/>	General purpose t2.small	1	2	EBS only	-
<input type="checkbox"/>	General purpose t2.medium	2	4	EBS only	-
<input type="checkbox"/>	General purpose t2.large	2	8	EBS only	-
<input type="checkbox"/>	General purpose m4.large	2	8	EBS only	Yes

**Step 6:** Accept the default settings on the next screen that comes up and then choose the option **Next: Add Storage**.

The screenshot shows the AWS EC2 Management Console interface. The top navigation bar includes the EC2 Management Console logo, a search bar with the URL <https://ap-southeast-1.console.aws.amazon.com/ec2/v2/home?region=ap-southeast-1#LaunchInstanceW>, and account information for Alan in Singapore. Below the navigation is a breadcrumb trail: 1. Choose AMI, 2. Choose Instance Type, 3. Configure Instance (which is highlighted in orange), 4. Add Storage, 5. Tag Instance, and 6. Configure Security Group.

**Step 3: Configure Instance Details**

Configure the instance to suit your requirements. You can launch multiple instances from the same AMI, request Spot instances to take advantage of the lower pricing, assign an access management role to the instance, and more.

**Number of instances** (i)  Launch into Auto Scaling Group (i)

**Purchasing option** (i)  
 Request Spot instances

**Network** (i)  
vpc-112b4974 (172.31.0.0/16) (default)

**Subnet** (i)  
No preference (default subnet in any Availability Zone)

Cancel Previous Review and Launch Next: Add Storage

Feedback English © 2008 - 2016, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use Activate Windows

**Step 7:** Accept the default settings on the next screen and choose the option **Next: Tag Instance**.

The screenshot shows the AWS EC2 Management Console interface for launching a new instance. The current step is '4. Add Storage'. The 'Volume Type' section shows a single root volume: 'Root' (Device: /dev/sda1, Snapshot: snap-2af8d9cb, Size: 30 GiB, Volume Type: General Purpose, IOPS: 90 / 3000, Delete on Termination checked, Encrypted not enabled). Below this is a button to 'Add New Volume'. At the bottom of the page are buttons for 'Cancel', 'Previous', 'Review and Launch' (which is highlighted in blue), and 'Next: Tag Instance'.

**Step 8:** Accept the default settings on the next screen and choose the option of **Next: Configure Security Group**.

The screenshot shows the AWS EC2 Management Console interface. The URL in the browser is <https://ap-southeast-1.console.aws.amazon.com/ec2/v2/home?region=ap-southeast-1#LaunchInstanceW>. The top navigation bar includes 'AWS Services' and 'Edit'. Below the navigation, a progress bar shows steps 1 through 6: '1. Choose AMI', '2. Choose Instance Type', '3. Configure Instance', '4. Add Storage', '5. Tag Instance' (which is highlighted in orange), and '6. Configure Security Group'. The main content area is titled 'Step 5: Tag Instance'. It asks for a tag key ('Name') and a tag value. A 'Create Tag' button is available for additional tags. At the bottom, there are buttons for 'Cancel', 'Previous', 'Review and Launch' (which is blue and bold), and 'Next: Configure Security Group'. The footer contains links for 'Feedback', 'English', 'Activate Windows', 'Privacy Policy', 'Terms of Use', and 'Customer Support'.

**Step 9:** Accept the default settings on the next screen and choose the option of **Review and Launch**.

The screenshot shows the AWS EC2 Management Console interface. The URL in the browser is <https://ap-southeast-1.console.aws.amazon.com/ec2/v2/home?region=ap-southeast-1#LaunchInstanceW>. The top navigation bar includes 'AWS Services' and 'Edit'. The main menu at the top has tabs: 1. Choose AMI, 2. Choose Instance Type, 3. Configure Instance, 4. Add Storage, 5. Tag Instance, and 6. Configure Security Group. The 'Configure Security Group' tab is currently selected. Below the tabs, the heading 'Step 6: Configure Security Group' is displayed. A descriptive text explains that a security group is a set of firewall rules that control the traffic for your instance. It mentions that you can add rules to allow specific traffic to reach your instance, such as setting up a web server and allowing Internet traffic. It also provides a link to learn more about Amazon EC2 security groups. The form fields for creating a new security group are shown: 'Security group name' (input field containing 'launch-wizard-4'), 'Description' (input field containing 'launch-wizard-4 created 2016-03-07T01:11:08.376-08:00'), and a table for defining security rules. The table has columns: Type, Protocol, Port Range, and Source. At the bottom right of the form are buttons for 'Cancel', 'Previous', and 'Review and Launch' (which is highlighted in blue).

**Step 10:** Click Launch in the next screen that comes up.

The screenshot shows the AWS EC2 Management Console with the URL <https://ap-southeast-1.console.aws.amazon.com/ec2/v2/home?region=ap-southeast-1#LaunchInstanceW>. The top navigation bar includes 'AWS', 'Services', 'Edit', 'Alan', 'Singapore', and 'Support'. Below the navigation, a progress bar shows steps 1 through 6: 'Choose AMI', 'Choose Instance Type', 'Configure Instance', 'Add Storage', 'Tag Instance', and 'Configure Security Group'. Step 3 is currently selected.

**Step 7: Review Instance Launch**

Please review your instance launch details. You can go back to edit changes for each section. Click **Launch** to assign a key pair to your instance and complete the launch process.

**Warning:** Improve your instances' security. Your security group, launch-wizard-4, is open to the world.

Your instances may be accessible from any IP address. We recommend that you update your security group rules to allow access from known IP addresses only.

You can also open additional ports in your security group to facilitate access to the application or service you're running, e.g., HTTP (80) for web servers. [Edit security groups](#)

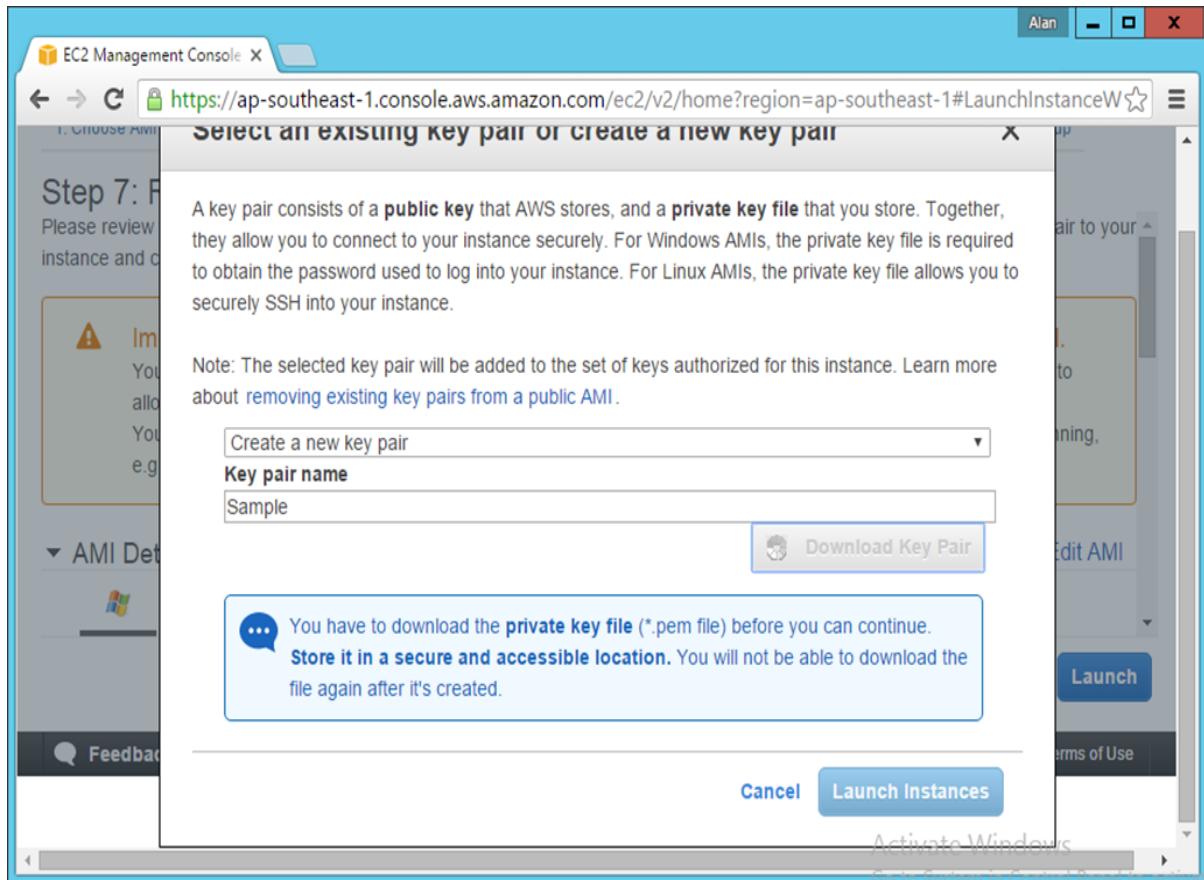
**AMI Details**

Microsoft Windows Server 2012 R2 Base - ami-9801cffb

Cancel Previous **Launch**

Feedback English © 2008 - 2016, Amazon Web Services, Inc. or its affiliates. All rights reserved. Active Windows Privacy Policy Terms of Use

**Step 11:** In the next screen that comes up, you will be prompted to create a key pair. This will be used to log into the server at a later point of time. Just create the key pair and click **Launch Instance**.



The instance will now be set up in Amazon.

### 3. CI – Reducing Risks

There are chances that things will go wrong on a project. By effectively practicing CI, you find out what happens at every step along the way, rather than later when the project is into the development cycle. CI helps you identify and mitigate risks when they occur, making it easier to evaluate and report on the health of the project based on concrete evidence.

This section is going to concentrate on the risks that can be avoided by using Continuous Integration.

On any project, there are many risks that need to be managed. By eliminating the risks earlier in the development lifecycle, there are lesser chances of these risks developing into issues later on, when the system actually goes live.

#### Risk 1 – Lack of Deployable Software

---

**“It works on my machine but does not work on another”** – This is probably one of the most common phrases encountered in any software organization. Because of the number of changes done to software builds on a daily basis, sometimes there is little confidence on whether the build of the software actually works or not. This concern has the following three side effects.

- Little or no confidence in whether we could even build the software.
- Lengthy integration phases before delivering the software internally (i.e., test team) or externally (i.e., customer), during which time nothing else gets done.
- Inability to produce and reproduce testable builds.

#### Solution

Eliminating tight coupling between the IDE and the build processes. Use a separate machine solely for integrating the software. Ensure that everything you need to build the software is contained in the version control repository. Finally, create a Continuous Integration system.

The Continuous Integration server can watch for changes in the version control repository and run the project build script when it detects a change to the repository. The capability of the Continuous Integration system can be increased to include having the build run through tests, perform inspections, and deploy the software in the development and test environments; this way you always have a working software.

**“Inability to synchronize with the database”** – Sometimes developers are unable to recreate the database quickly during development, and hence find it difficult to make changes. Often this is due to a separation between the database team and the development team. Each team will be focused on their own responsibilities and have little collaboration between each other. This concern has the following three side effects –

- Fear of making changes or refactoring the database or source code.
- Difficulty in populating the database with different sets of test data.

- Difficulty in maintaining development and testing environments (e.g., Development, Integration, QA, and Test).

## Solution

The solution to the above issue is to ensure that the placement of all database artifacts in the version control repository are carried out. This means everything that is required to recreate the database schema and data: database creation scripts, data manipulation scripts, stored procedures, triggers, and any other database assets are needed.

Rebuild the database and data from your build script, by dropping and recreating your database and tables. Next, apply the stored procedures and triggers, and finally, insert the test data.

Test (and inspect) your database. Typically, you will use the component tests to test the database and data. In some cases, you'll need to write database-specific tests.

## Risk 2 – Discovering Defects Late in the Lifecycle

---

Since there are so many changes which happen frequently by multiple developers to the source code, there are always chances that a defect can be introduced in the code that could only be detected at a later stage. In such cases, this can cause a big impact because the later the defect is detected in the software, the more expensive it becomes to remove the defect.

## Solution

**Regression Testing** – This is the most important aspect of any software development cycle, test and test again. If there is any major change to the software code, it is absolutely mandatory to ensure that all the tests are run. And this can be automated with the help of the Continuous Integration server.

**Test Coverage** – There is no point in testing if the test cases do not cover the entire functionality of the code. It is important to ensure that the test cases created to test the application are complete and that all code paths are tested.

For example, if you have a login screen which needs to be tested, you just can't have a test case that has the scenario of a successful login. You need to have a negative test case wherein a user enters a different combination of user names and passwords and then it is required to see what happens in such scenarios.

## Risk 3 – Lack of Project Visibility

---

Manual communication mechanisms require a lot of coordination to ensure the dissemination of project information to the right people in a timely manner. Leaning over to the developer next to you and letting them know that the latest build is on the shared drive is rather effective, yet it doesn't scale very well.

What if there are other developers who need this information and they are on a break or otherwise unavailable? If a server goes down, how are you notified? Some believe they can mitigate this risk by manually sending an e-mail. However, this cannot ensure the information is communicated to the right people at the right time because you may accidentally leave out interested parties, and some may not have access to their e-mail at the time.

## Solution

The Solution to this issue is again the Continuous Integration server. All CI servers have the facility to have automated emails to be triggered whenever the builds fail. By this automatic notification to all key stakeholders, it is also ensured that everyone is on board on what is the current state of the software.

## Risk 4 – Low Quality Software

---

There are defects and then there are potential defects. You can have potential defects when your software is not well designed or if it is not following the project standards, or is complex to maintain. Sometimes people refer to this as code or design smells — “a symptom that something may be wrong.”

Some believe that lower-quality software is solely a deferred project cost (after delivery). It can be a deferred project cost, but it also leads to many other problems before you deliver the software to the users. Overly complex code, code that does not follow the architecture, and duplicated code - all usually lead to defects in the software. Finding these code and design smells before they manifest into defects can save both time and money, and can lead to higher-quality software.

## Solution

There are software components to carry out a code quality check which can be integrated with the CI software. This can be run after the code is built to ensure that the code actually conforms to proper coding guidelines.

## 4. CI – Version Control

Version control systems, also known as source control, source code management systems, or revision control systems, are a mechanism for keeping multiple versions of your files, so that when you modify a file you can still access the previous revisions.

The first popular version control system was a proprietary UNIX tool called **SCCS** (Source Code Control System) which dates back to the 1970s. This was superseded by **RCS**, the Revision Control System, and later **CVS**, Concurrent Versions System.

Now the most popular version control system used are **Subversion** and **Git**. Let's first look at why we need to use a versioning control system and next let's look at putting our source code in **Git source code repository system**.

### Purpose of the Version Control System

---

One reason that we use the term version control in preference to source control is that version control isn't just for source code. Every single artifact related to the creation of your software should be under version control.

- **Developers should use it for source code** – By default all source code needs to be stored in the versioning control system.
- **Related artefacts** – Every system would be having related artefacts to the source code such as database scripts, build and deployment scripts, documentation, libraries and configuration files for your application, your compiler and collection of tools, and so on. All of these compliment the entire development and deployment process and also needs to be stored in the versioning control system.

By storing all the information for the application in source control, it becomes easier to re-create the testing and production environments that your application runs on. This should include configuration information for your application's software stack and the operating systems that comprise the environment, DNS Zone Files, Firewall Configuration, and so forth.

At the bare minimum, you need everything required to re-create your application's binaries and the environments in which they run. The objective is to have everything that can possibly change at any point in the life of the project stored in a controlled manner. This allows you to recover an exact snapshot of the state of the entire system, from development environment to production environment, at any point in the project's history.

It is even helpful to keep the configuration files for the development team's development environments in version control since it makes it easy for everyone on the team to use the same settings. Analysts should store requirements documents. Testers should keep their test scripts and procedures in version control. Project managers should save their release plans, progress charts, and risk logs here.

In short, every member of the team should store any document or file related to the project in version control.

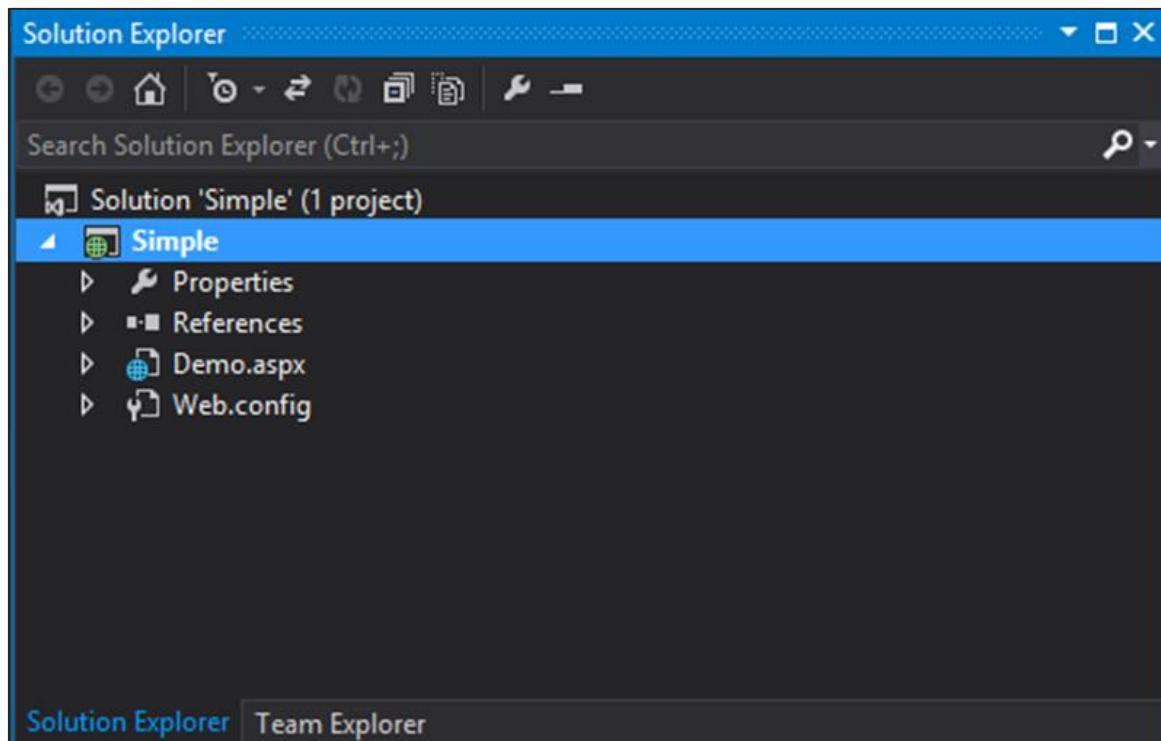
## Working with Git for Source Code Versioning Control System

This section will now focus on how Git can be used as a versioning control system. It will focus on how you can upload your code to the versioning control system and manage changes in it.

### Our Demo Application

For the purpose of this entire tutorial we are going to look at a simple **Web ASP.Net** application which will be used for the entire Continuous Integration Process. We don't need to focus on the entire code details for this exercise, just having an overview of what the project does is sufficient for understanding the entire continuous integration process. This .Net application was built using the **Visual Studio Integrated Development Environment**.

The following screenshot is the structure of the solution in the Visual Studio environment. It is a very simple Web application which has the main code in the **Demo.aspx** file.



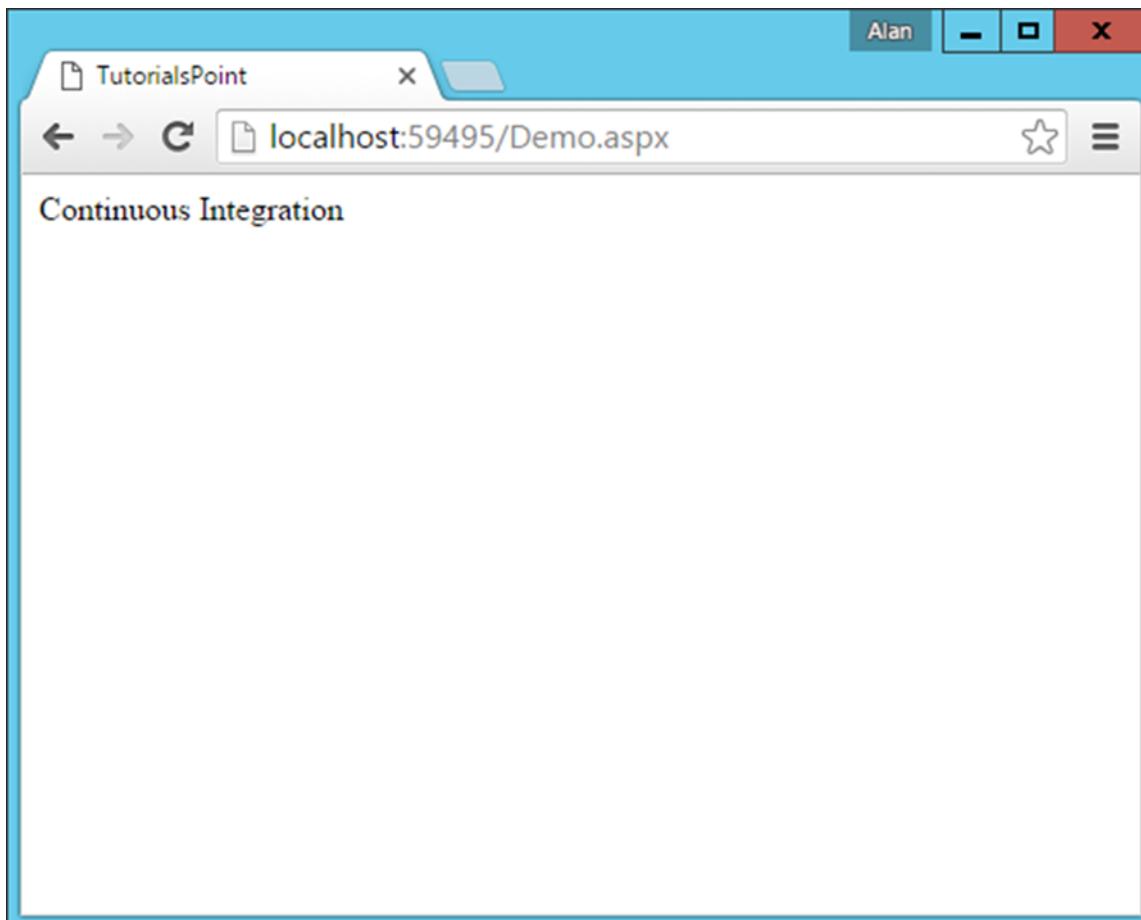
The code in the **Demo.aspx** file is shown in the following program:

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>TutorialsPoint</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
```

```
<%Response.Write("Continuous Integration"); %>
</div>
</form>
</body>
</html>
```

The code is very simple and just outputs the string "Continuous Integration" to the browser.

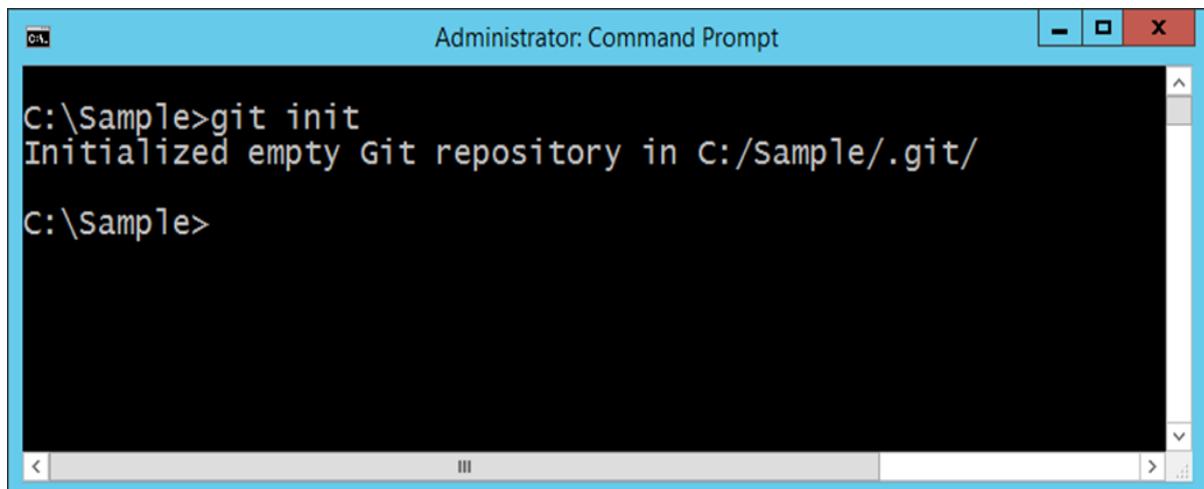
When you run the project in Google Chrome, the output will be as shown in the following screenshot.



## Moving Source Code to Git

We are going to show how to move the source code to Git from the command line interface, so that the knowledge of how Git can be used is clearer to the end user.

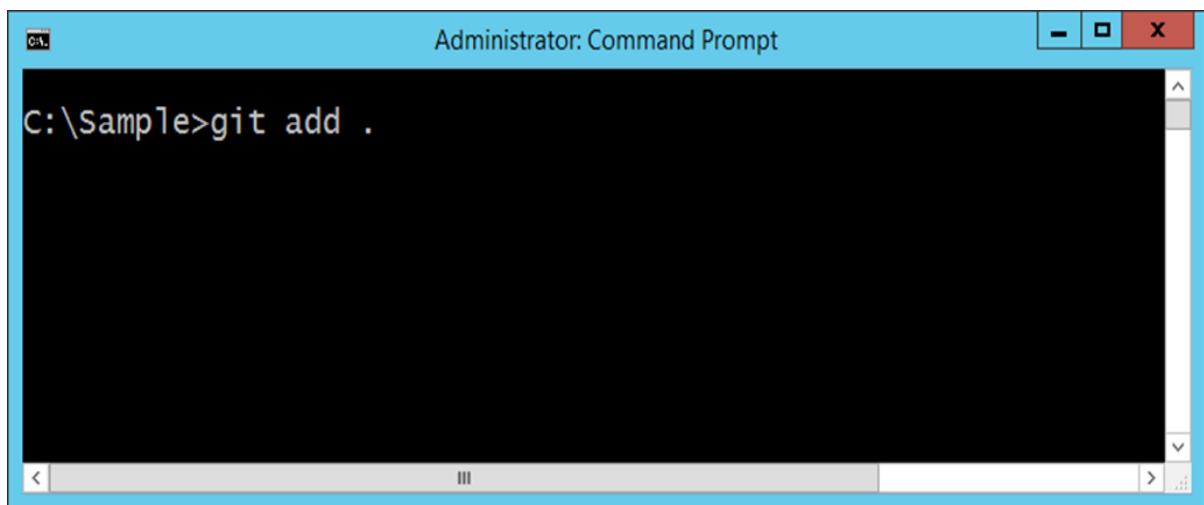
**Step 1:** Initialize the **Git Repository**. Go to the command prompt, go to your project folder and issue the command **git init**. This command will add the necessary Git files to the project folder, so that it can be recognized by Git when it needs to be uploaded to the repository.



Administrator: Command Prompt

```
C:\Sample>git init
Initialized empty Git repository in C:/Sample/.git/
C:\Sample>
```

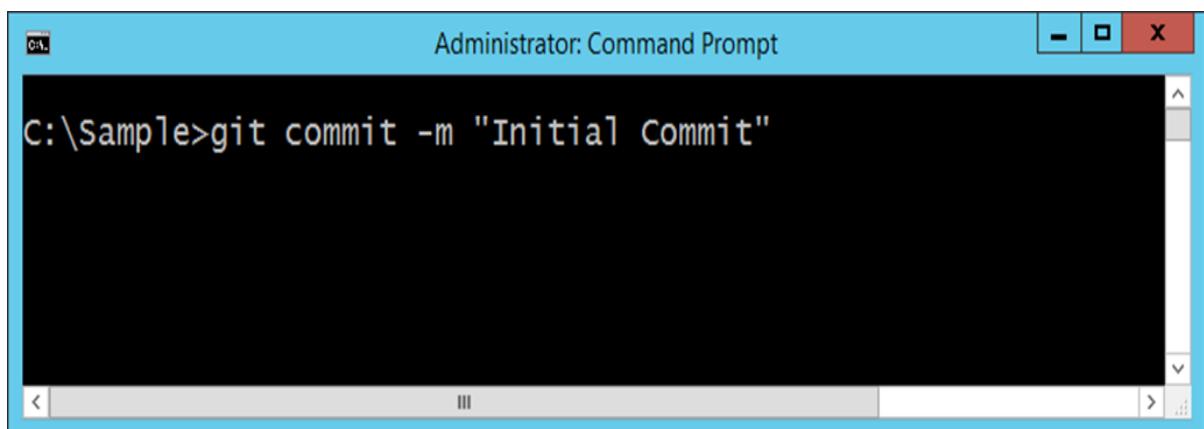
**Step 2:** Adding your files which need to be added to the Git repository. This can be done by issuing the **git add command**. The dot option tells Git that all files in the project folder need to be added to the Git repository.



Administrator: Command Prompt

```
C:\Sample>git add .
```

**Step 3:** The final step is to commit the project files to the Git repository. This step is required to ensure all files are now a part of Git. The command to be issued is given in the following screenshot. The **-m option** is to provide a comment to the upload of files.



Administrator: Command Prompt

```
C:\Sample>git commit -m "Initial Commit"
```

Your solution is now available in Git.

## 5. CI – Features

Following are some of the main features or practices for Continuous Integration.

- **Maintain a single source repository** – All source code is maintained in a single repository. This avoids having source code being scattered across multiple locations. Tools such as **Subversion and Git** are the most popular tools for maintaining source code.
- **Automate the build** – The build of the software should be carried out in such a way that it can be automated. If there are multiple steps that need to be carried out, then the build tool needs to be capable of doing this. For .Net, MSBuild is the default build tool and for Java based applications you have tools such as **Maven and Grunt**.
- **Make your build self-testing** – The build should be testable. Directly after the build occurs, test cases should be run to ensure that testing can be carried out for the various functionality of the software.
- **Every commit should build on an integration machine** – The integration machine is the build server and it should be ensured that the build runs on this machine. This means that all dependent components should exist on the Continuous Integration server.
- **Keep the build fast** – The build should happen in minutes. The build should not take hours to happen, because this would mean the build steps are not properly configured.
- **Test in a clone of the production environment** – The build environment should be close in nature to the production environment. If there are vast differences between these environments, then there can be a case that the build may fail in production even though it passes on the build server.
- **Everyone can see what is happening** – The entire process of build and testing and deployment should be visible to all.
- **Automate deployment** – Continuous Integration leads to Continuous deployment. It is absolutely necessary to ensure that the build should be easy to deploy onto either a staging or production environment.

# 6. CI – Requirements

Following is the list of the most significant requirements for Continuous Integration.

- **Check-In Regularly** – The most important practice for continuous integration to work properly is frequent check-ins to trunk or mainline of the source code repository. The check-in of code should happen at least a couple of times a day. Checking in regularly brings lots of other benefits. It makes changes smaller and thus less likely to break the build. It means the recent most version of the software to revert to is known when a mistake is made in any subsequent build.

It also helps to be more disciplined about refactoring code and to stick to small changes that preserve behavior. It helps to ensure that changes altering a lot of files are less likely to conflict with other people's work. It allows the developers to be more explorative, trying out ideas and discarding them by reverting back to the last committed version.

- **Create a Comprehensive Automated Test Suite** – If you don't have a comprehensive suite of automated tests, a passing build only means that the application could be compiled and assembled. While for some teams this is a big step, it's essential to have some level of automated testing to provide confidence that your application is actually working.

Normally, there are 3 types of tests conducted in Continuous Integration namely **unit tests**, **component tests**, and **acceptance tests**.

Unit tests are written to test the behavior of small pieces of your application in isolation. They can usually be run without starting the whole application. They do not hit the database (if your application has one), the filesystem, or the network. They don't require your application to be running in a production-like environment. Unit tests should run very fast — your whole suite, even for a large application, should be able to run in under ten minutes.

Component tests test the behavior of several components of your application. Like unit tests, they don't always require starting the whole application. However, they may hit the database, the filesystem, or other systems (which may be stubbed out). Component tests typically take longer to run.

- **Keep the Build and Test Process Short** - If it takes too long to build the code and run the unit tests, you will run into the following problems –
  - People will stop doing a full build and will run the tests before they check-in. You will start to get more failing builds.
  - The Continuous Integration process will take so long that multiple commits would have taken place by the time you can run the build again, so you won't know which check-in broke the build.
  - People will check-in less often because they have to sit around for ages waiting for the software to build and the tests to run.

- **Don't Check-In on a Broken Build** – The biggest blunder of continuous integration is checking in on a broken build. If the build breaks, the developers responsible are waiting to fix it. They identify the cause of the breakage as soon as possible and fix it. If we adopt this strategy, we will always be in the best position to work out what caused the breakage and fix it immediately.

If one of our colleagues has made a check-in and has as a result broken the build, then to have the best chance of fixing it, they will need a clear run at the problem. When this rule is broken, it inevitably takes much longer for the build to be fixed. People get used to seeing the build broken, and very quickly you get into a situation where the build stays broken all of the time.

- **Always Run All Commit Tests Locally Before Committing** – Always ensure that the tests designed for the application are run first on a local machine before running them on the CI server. This is to ensure the right test cases are written and if there is any failure in the CI process, it is because of the failed test results.
- **Take Responsibility for All Breakages that Result from Your Changes** – If you commit a change and all the tests you wrote pass, but others break, the build is still broken. Usually this means that you have introduced a regression bug into the application. It is your responsibility — because you made the change — to fix all tests that are not passing as a result of your changes. In the context of CI this seems obvious, but actually it is not a common practice in many projects.

## 7. CI – Building a Solution

There are a variety of build tools available for a variety of programming languages. Some of the most popular build tools include **Ant for Java** and **MSBuild for .NET**. Using a **scripting tool** designed specifically for building software, instead of a custom set of shell or batch scripts, is the most effective manner for developing a consistent, repeatable build solution.

So why do we need a build process to start with. Well for starters, for a Continuous Integration server, the build process should be easy to work with and should be seamless to implement.

Let's take a simple example of what a build file can look like for .Net –

```
<?xml version="1.0" encoding="utf-8"?>
<project xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
    <Target Name="Build">
        <Message Text ="Building Project" />
        <MSBuild Projects="project.csproj" Targets="Build"/>
    </Target>
</project>
```

The following aspects need to be noted about the above code –

- A target is specified with a name of the Build. Wherein, a target is a collection of logical steps which need to be performed in a build process. You can have multiple targets and have dependencies between targets.
- In our target, we keep an option message which will be shown when the build process starts.
- The **MSBuild task** is used to specify which .Net project needs to be built.

The above example is a case of a very simple build file. In Continuous Integration, it is ensured that this file is kept up-to-date to ensure that the entire build process is seamless.

### **Building a Solution in .Net**

The default build tool for .Net is MSBuild and is something that comes shipped with the .Net framework. Depending on the framework on your system, you will have the relevant MSbuild version available. As an example, if you have the .Net framework installed in the default location, you will find the **MSBuild.exe** file in the following location –

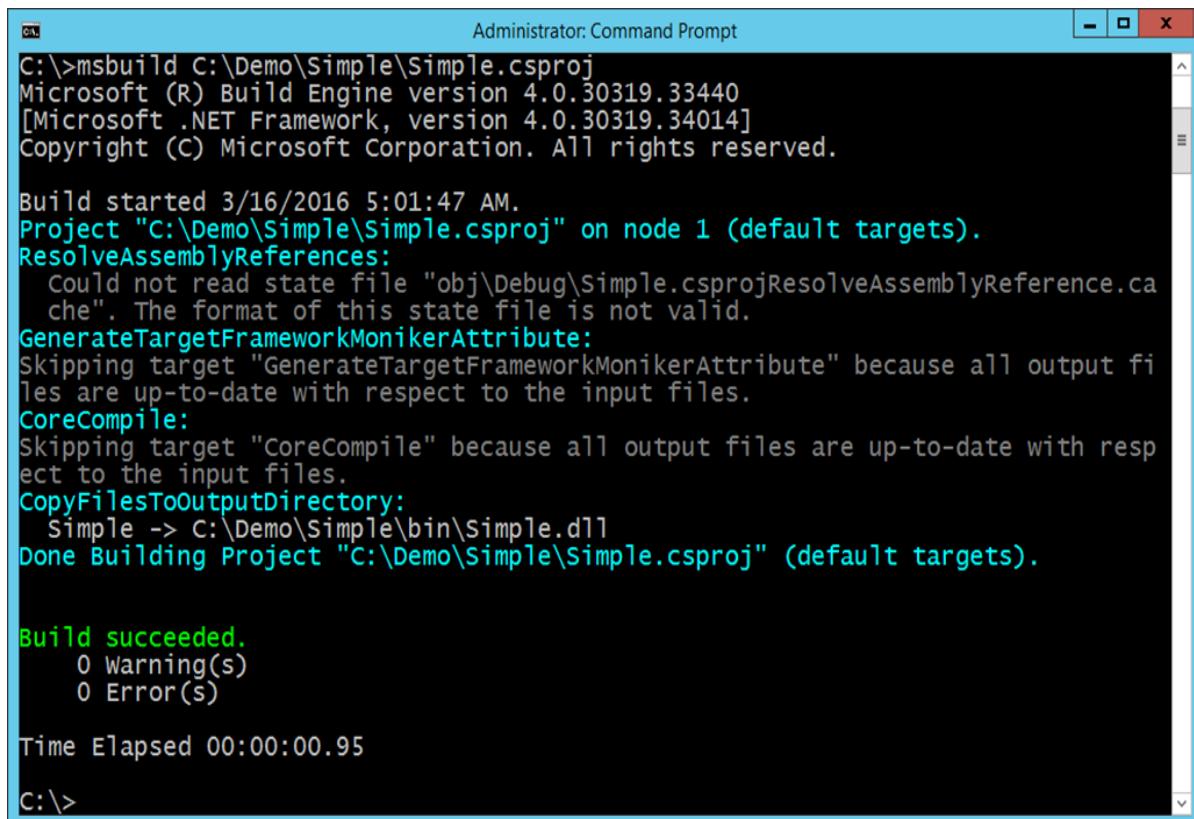
```
C:\Windows\Microsoft.NET\Framework\v4.0.30319
```

Let's see how we can go about building our sample project. Let's assume our Sample project is located in a folder called **C:\Demo\Simple**.

In order to use MSBuild to build the above solution, we need to open the command prompt and use the MSBuild option as shown in the following program.

```
msbuild C:\Demo\Simple\Simple.csproj
```

In the above example, **csproj** is the project file which is specific to .Net. The csproj file contains all the relevant information to ensure that the required information is present for the software to build properly. Following is the screenshot of the output of the MSBuild command.



```
Administrator: Command Prompt
C:\>msbuild C:\Demo\Simple\Simple.csproj
Microsoft (R) Build Engine version 4.0.30319.33440
[Microsoft .NET Framework, version 4.0.30319.34014]
Copyright (C) Microsoft Corporation. All rights reserved.

Build started 3/16/2016 5:01:47 AM.
Project "C:\Demo\Simple\Simple.csproj" on node 1 (default targets).
ResolveAssemblyReferences:
  Could not read state file "obj\Debug\Simple.csprojResolveAssemblyReference.ca
  che". The format of this state file is not valid.
GenerateTargetFrameworkMonikerAttribute:
  Skipping target "GenerateTargetFrameworkMonikerAttribute" because all output fi
  les are up-to-date with respect to the input files.
CoreCompile:
  Skipping target "CoreCompile" because all output files are up-to-date with resp
  ect to the input files.
CopyFilesToOutputDirectory:
  Simple -> C:\Demo\Simple\bin\simple.dll
Done Building Project "C:\Demo\Simple\Simple.csproj" (default targets).

Build succeeded.
  0 Warning(s)
  0 Error(s)

Time Elapsed 00:00:00.95
C:\>
```

You don't need to worry about the output warnings as long as the Build was successful and there were no errors.

## 8. CI – Build Scripts

Now let's look at certain aspects of the MSBuild file to see what they mean. These aspects are important to know from a Continuous Integration Cycle.

Build scripts are used to build the solution which will be a part of the entire continuous Integration cycle. Let's look at the general build script which is created as a part of Visual Studio in **.Net** for our sample solution. The build script is a pretty big one, even for a simple solution, so we will go through the most important parts of it. By default, the build script will be stored in a file with the same name as the main solution in Visual Studio. So in our case, if you open the file **Simple.csproj**, you will see all the settings which will be used to build the solution.

- Dependency on the MSBuild version used – The following settings will use the MSBuild files installed on the CI server.

```
<VisualStudioVersion Condition="$(VisualStudioVersion) ==  
'10.0"/>  
  
<VSToolsPath Condition="$(VSToolsPath) ==  
'$(MSBuildExtensionsPath32)\Microsoft\VisualStudio\v$(VisualStudioVersion)"/>  
  
<TargetFrameworkVersion>v4.5</TargetFrameworkVersion>  
  
<Import Project="$(MSBuildBinPath)\Microsoft.CSharp.targets" />  
  
  <Import  
    Project="$(VSToolsPath)\WebApplications\Microsoft.WebApplication.targets"  
    Condition="$(VSToolsPath) != ''" />  
  
  <Import  
    Project="$(MSBuildExtensionsPath32)\Microsoft\VisualStudio\v10.0\WebApplication  
s\Microsoft.WebApplication.targets" Condition="false" />
```

- What files are required to build the solution properly – The **ItemGroup** tag will contain all the necessary .Net files which are required for the project to build successfully. These files will need to reside on the build server accordingly.

```
<ItemGroup>  
  <Reference Include="Microsoft.CSharp" />  
  <Reference Include="System.Web.DynamicData" />  
  <Reference Include="System.Web.Entity" />  
  <Reference Include="System.Web.ApplicationServices" />  
  <Reference Include="System.ComponentModel.DataAnnotations" />  
  <Reference Include="System" />  
  <Reference Include="System.Data" />  
  <Reference Include="System.Core" />
```

```

<Reference Include="System.Data.DataSetExtensions" />
<Reference Include="System.Web.Extensions" />
<Reference Include="System.Xml.Linq" />
<Reference Include="System.Drawing" />
<Reference Include="System.Web" />
<Reference Include="System.Xml" />
<Reference Include="System.Configuration" />
<Reference Include="System.Web.Services" />
<Reference Include="System.EnterpriseServices" />
</ItemGroup>

```

- What are the Web server settings to be used – When we visit our topic of Continuous Deployment, you will see how MSBuild will be used to override these settings and deploy this to our server of choice.

```

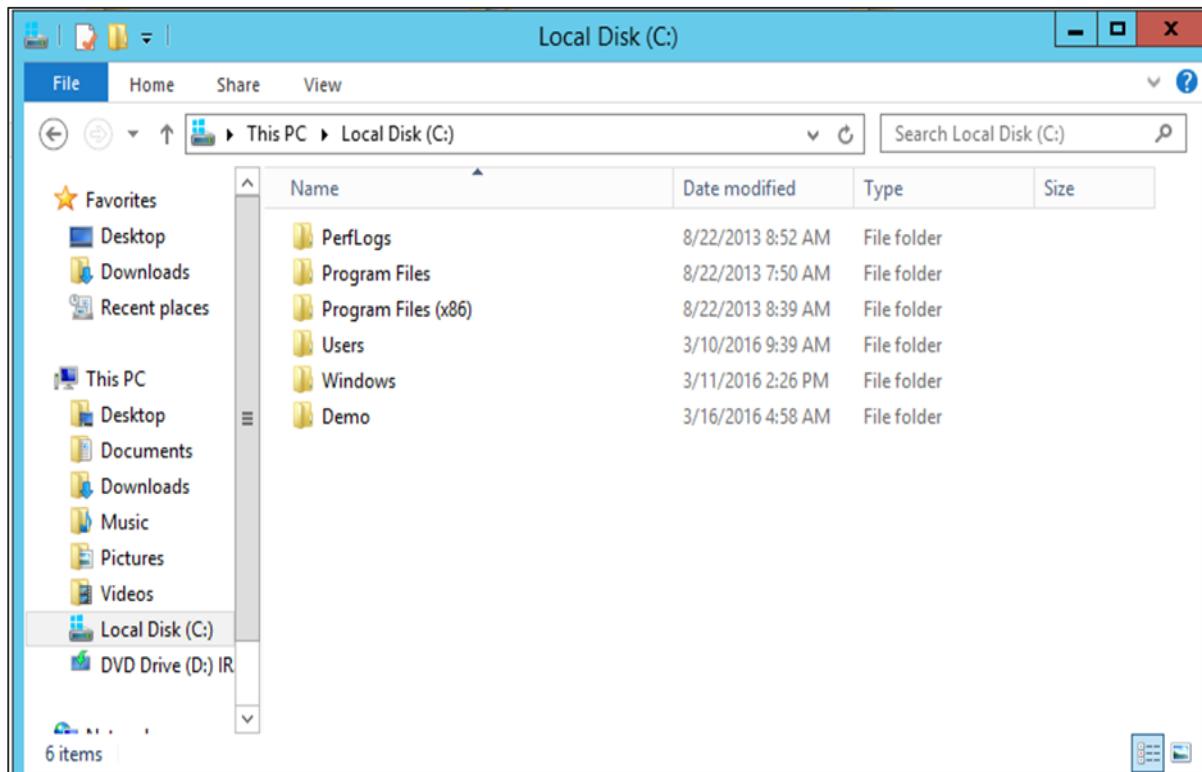
<UseIIS>True</UseIIS>
    <AutoAssignPort>True</AutoAssignPort>
    <DevelopmentServerPort>59495</DevelopmentServerPort>
    <DevelopmentServerVPath>/</DevelopmentServerVPath>
    <IISUrl>http://localhost:59495/</IISUrl>
    <NTLMAuthentication>False</NTLMAuthentication>
    <UseCustomServer>False</UseCustomServer>

```

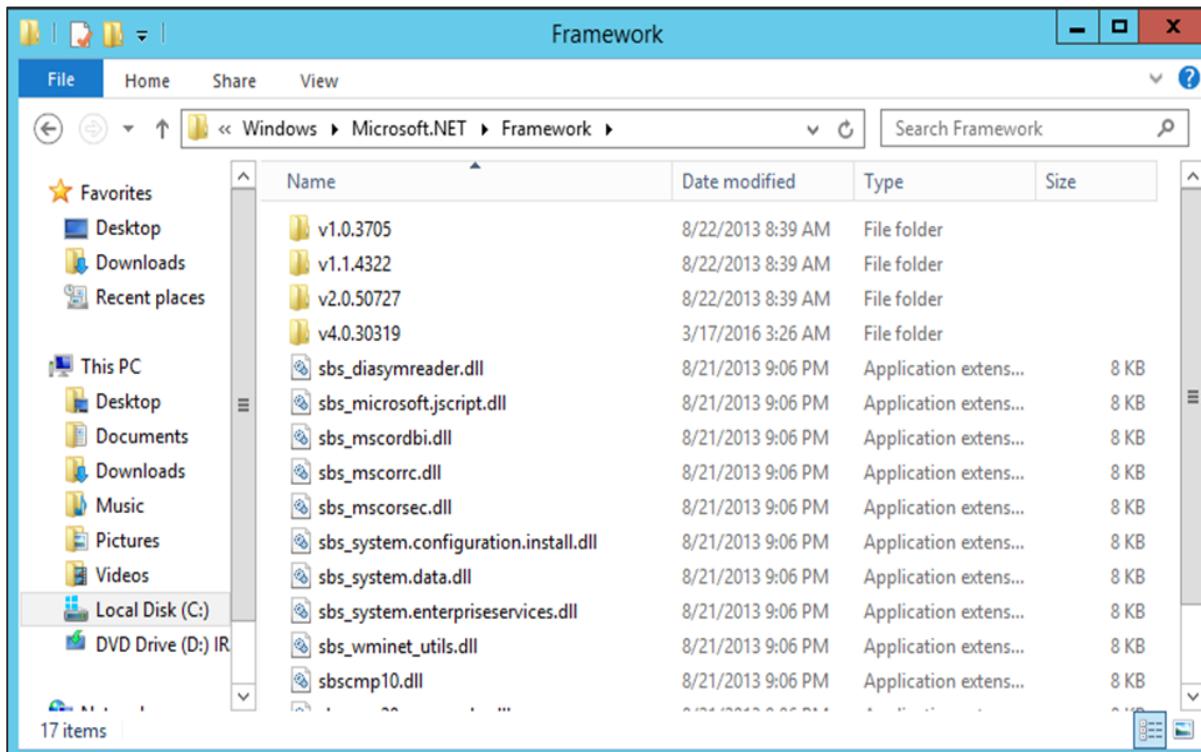
## 9. CI – Building on the Server

The next important step is to ensure that the solution builds on the build server. The first part is a manual step, because before the continuous integration tool is used, we first must ensure that the build gets run on the build server in the same manner as what was done on the client machine. To do this, we must implement the following steps:

**Step 1:** Copy the entire solution file to the server. We had created an Amazon instance server which would be used as our build server. So, do a manual copy to the server of the entire **.Net** solution onto the server.



**Step 2:** Ensure that the framework is present on the server. If you have compiled your application in .Net framework 4.0 on your client machine, you have to ensure that it is installed on the server machine as well. So go to the location **C:\Windows\Microsoft.NET\Framework** on your server and ensure the desired framework is present.



**Step 3:** Now let's just run MSBuild on the server and see what happens.

```

Administrator: Windows PowerShell
Time Elapsed 00:00:02.43
PS C:\Demo\Simple> MSBuild Simple.csproj
Microsoft (R) Build Engine version 4.0.30319.33440
[Microsoft .NET Framework, version 4.0.30319.34014]
Copyright (C) Microsoft Corporation. All rights reserved.

Build started 3/17/2016 6:22:20 AM.
Project "C:\Demo\Simple\Simple.csproj" on node 1 (default targets).
C:\Demo\Simple\Simple.csproj(86,3): error MSB4019: The imported project "C:\Program Files (x86)\MSBuild\Microsoft\VisualStudio\v10.0\WebApplications\Microsoft.WebApplication.targets" was not found. Confirm that the path in the <Import> declaration is correct, and that the file exists on disk.
Done Building Project "C:\Demo\Simple\Simple.csproj" (default targets) -- FAILED.

Build FAILED.

"C:\Demo\Simple\Simple.csproj" (default target) (1) ->
  C:\Demo\Simple\Simple.csproj(86,3): error MSB4019: The imported project "C:\Program Files (x86)\MSBuild\Microsoft\VisualStudio\v10.0\WebApplications\Microsoft.WebApplication.targets" was not found. Confirm that the path in the <Import> declaration is correct, and that the file exists on disk.

0 Warning(s)
1 Error(s)

Time Elapsed 00:00:00.23
PS C:\Demo\Simple>

```

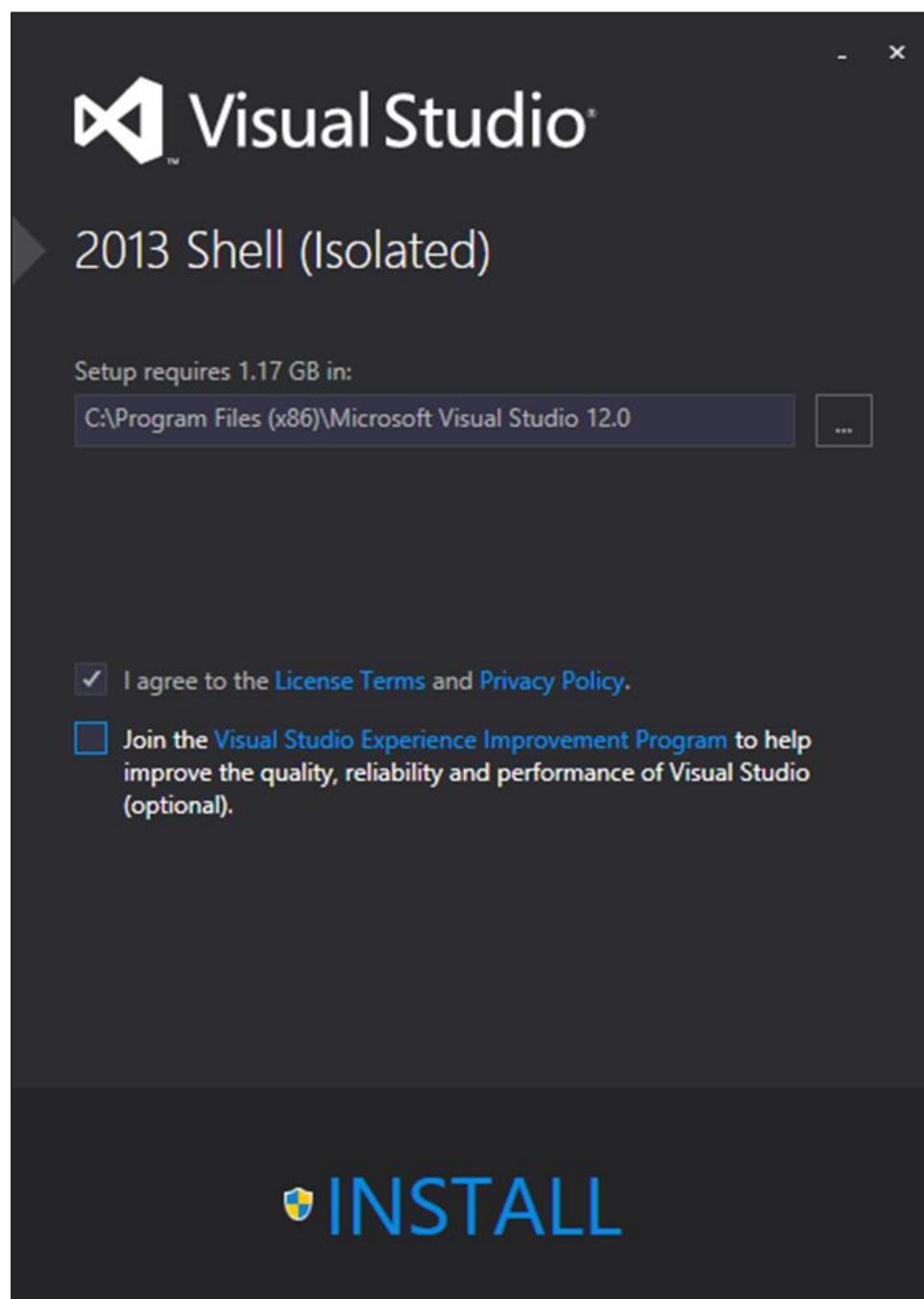
Ok, so it looks like we have hit an error. There is one important lesson in Continuous Integration and that is you need to ensure that the Build works on the build server. For this you need to ensure that all prerequisite software is installed on the build server.

For .Net, we need to install a component called **Visual Studio Redistributable package**. This package contains all the necessary files which are required for a **.Net** application to build on a server. So let's carry out the following installation steps on the build server.

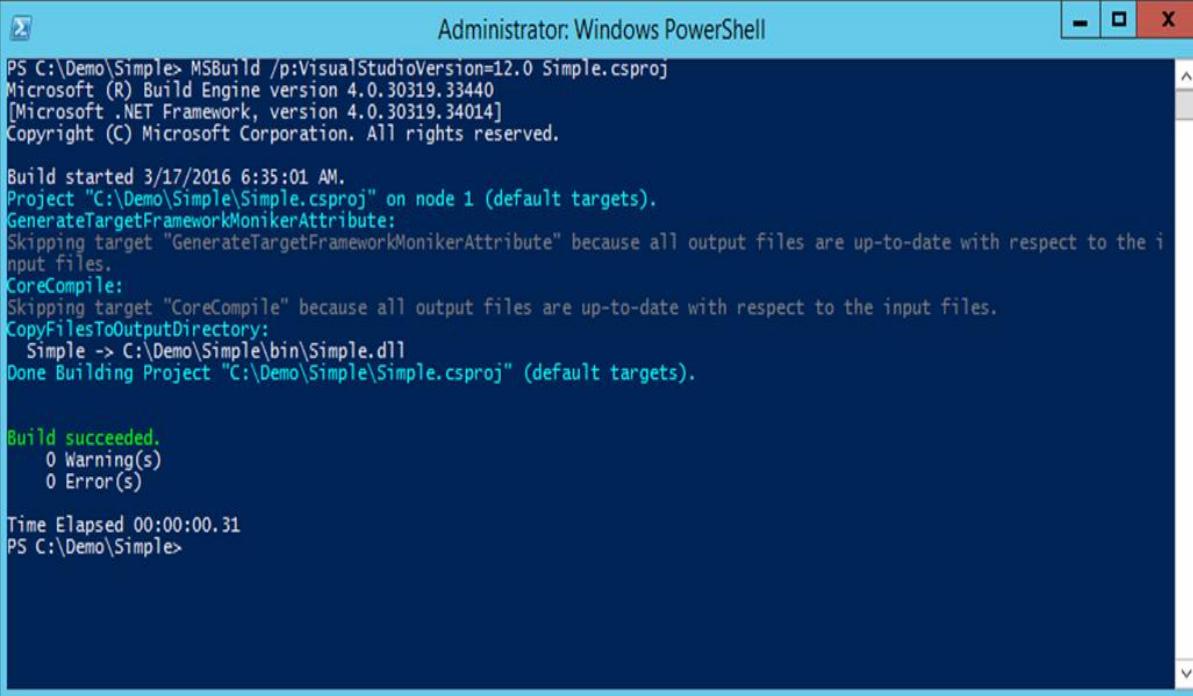
**Step 4:** Double-click the executable file to start the installation.



**Step 5:** In the next step, agree to the License Terms and click Install.



**Step 6:** Now when running MSBuild, we need to ensure that we include an additional parameter when calling MSBuild which is – **p:VisualStudioVersion=12.0**. This ensures that MSBuild references those files that were downloaded in the earlier step.



The screenshot shows an Administrator Windows PowerShell window. The title bar reads "Administrator: Windows PowerShell". The command entered is "PS C:\Demo\Simple> MSBuild /p:VisualStudioVersion=12.0 Simple.csproj". The output shows the build process starting at 3/17/2016 6:35:01 AM, skipping targets like "GenerateTargetFrameworkMonikerAttribute" because files are up-to-date, and successfully building the project "Simple" into "C:\Demo\Simple\bin\Simple.dll". It concludes with "Build succeeded.", 0 warnings, and 0 errors, and a total elapsed time of 00:00:00.31.

```
Administrator: Windows PowerShell
PS C:\Demo\Simple> MSBuild /p:VisualStudioVersion=12.0 Simple.csproj
Microsoft (R) Build Engine version 4.0.30319.33440
[Microsoft .NET Framework, version 4.0.30319.34014]
Copyright (C) Microsoft Corporation. All rights reserved.

Build started 3/17/2016 6:35:01 AM.
Project "C:\Demo\Simple\Simple.csproj" on node 1 (default targets).
GenerateTargetFrameworkMonikerAttribute:
Skipping target "GenerateTargetFrameworkMonikerAttribute" because all output files are up-to-date with respect to the input files.
CoreCompile:
Skipping target "CoreCompile" because all output files are up-to-date with respect to the input files.
CopyFilesToOutputDirectory:
  Simple > C:\Demo\Simple\bin\Simple.dll
Done Building Project "C:\Demo\Simple\Simple.csproj" (default targets).

Build succeeded.
  0 Warning(s)
  0 Error(s)

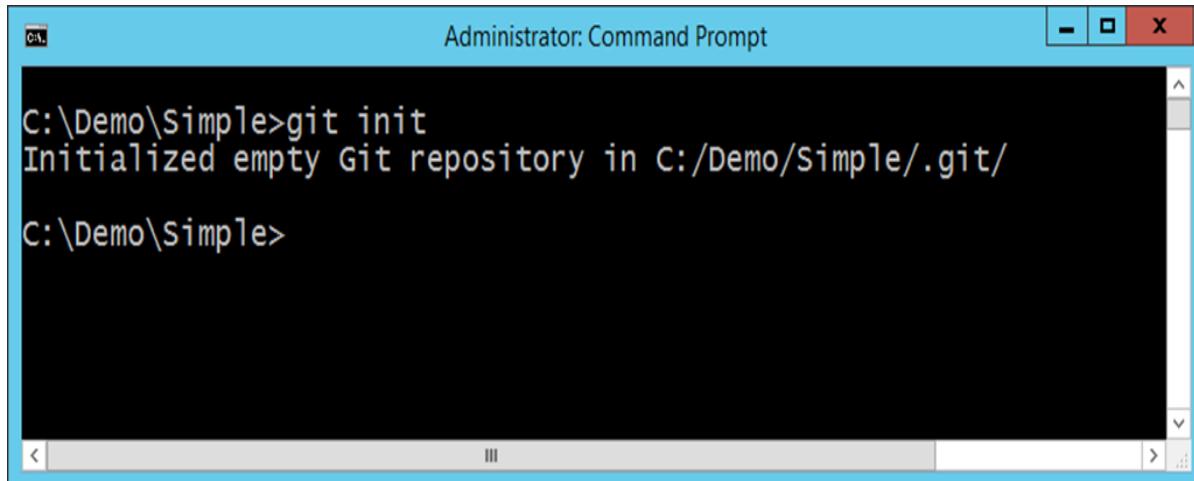
Time Elapsed 00:00:00.31
PS C:\Demo\Simple>
```

Now we can see that the solution has been built properly and we also know our baseline project builds correctly on the server.

# 10. CI – Checking in Source Code

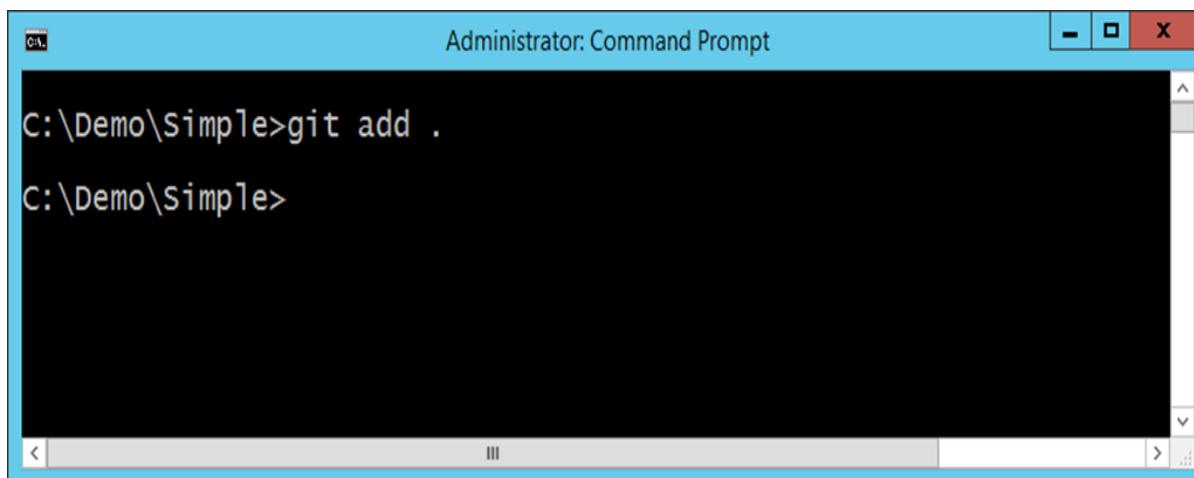
The next key aspect is to ensure our baseline code is checked into our source code repository management server which is Git. To do this, we need to follow these steps:

**Step 1:** Initialize the repository so that it can be uploaded to Git. This is done with the **git init** command. So you need to go to your project folder and issue the **git init** command.



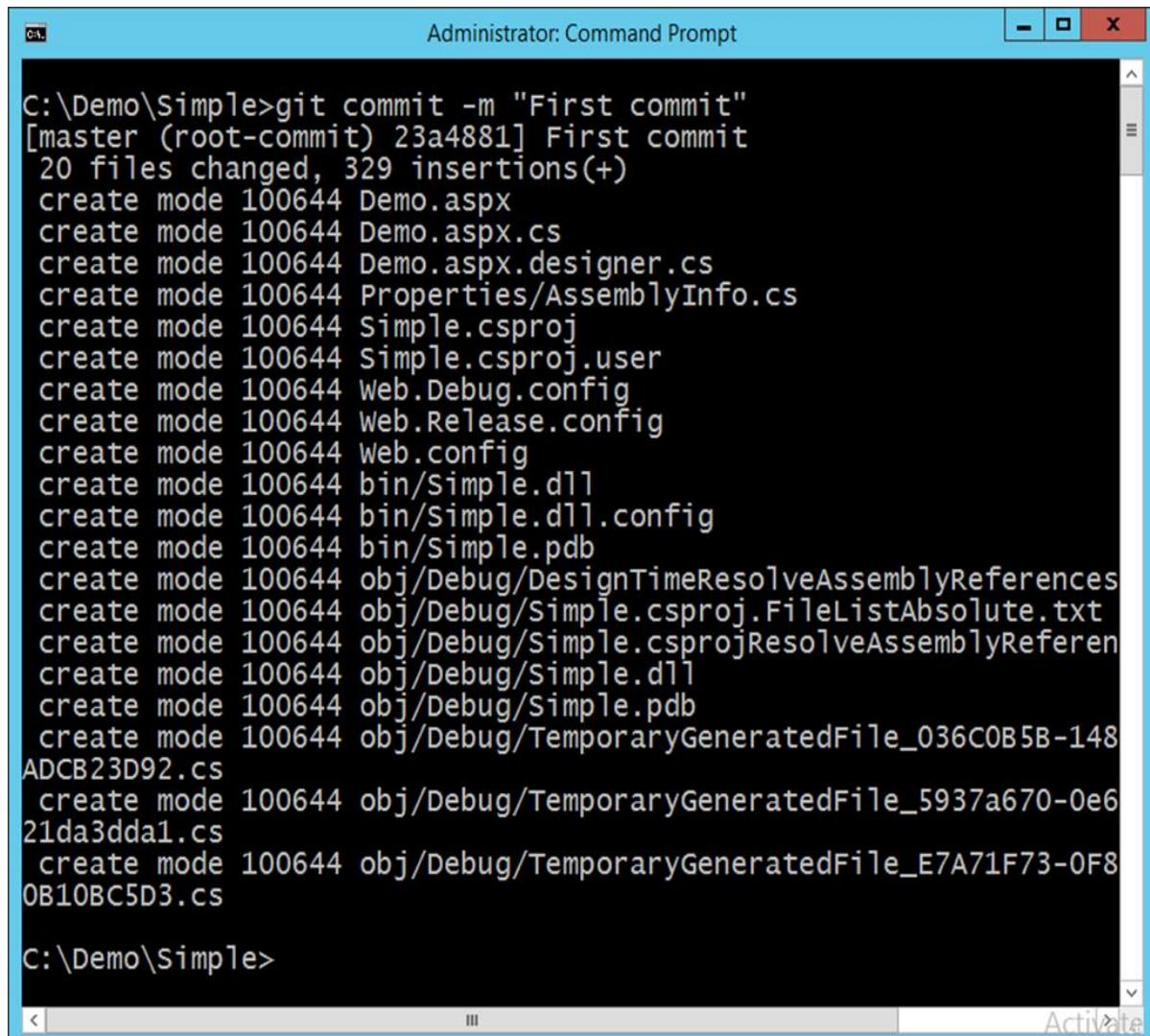
An Administrator Command Prompt window titled "Administrator: Command Prompt". The command entered is "C:\Demo\simple>git init". The output shows "Initialized empty Git repository in C:/Demo/Simple/.git/" followed by a new line and the prompt "C:\Demo\simple>".

**Step 2:** The next step is called staging files in Git. This prepares all the files in the project folder, which need to be added to Git. You do this with the **git add** command as shown in the following screenshot. The '.' notation is used to say that all files in the directory and subdirectory should be included in the commit.



An Administrator Command Prompt window titled "Administrator: Command Prompt". The command entered is "C:\Demo\simple>git add .". The output shows a blank line followed by the prompt "C:\Demo\simple>".

**Step 3:** The final step is to commit the files to the Git repository, so that it is now a full-fledged Git repository.



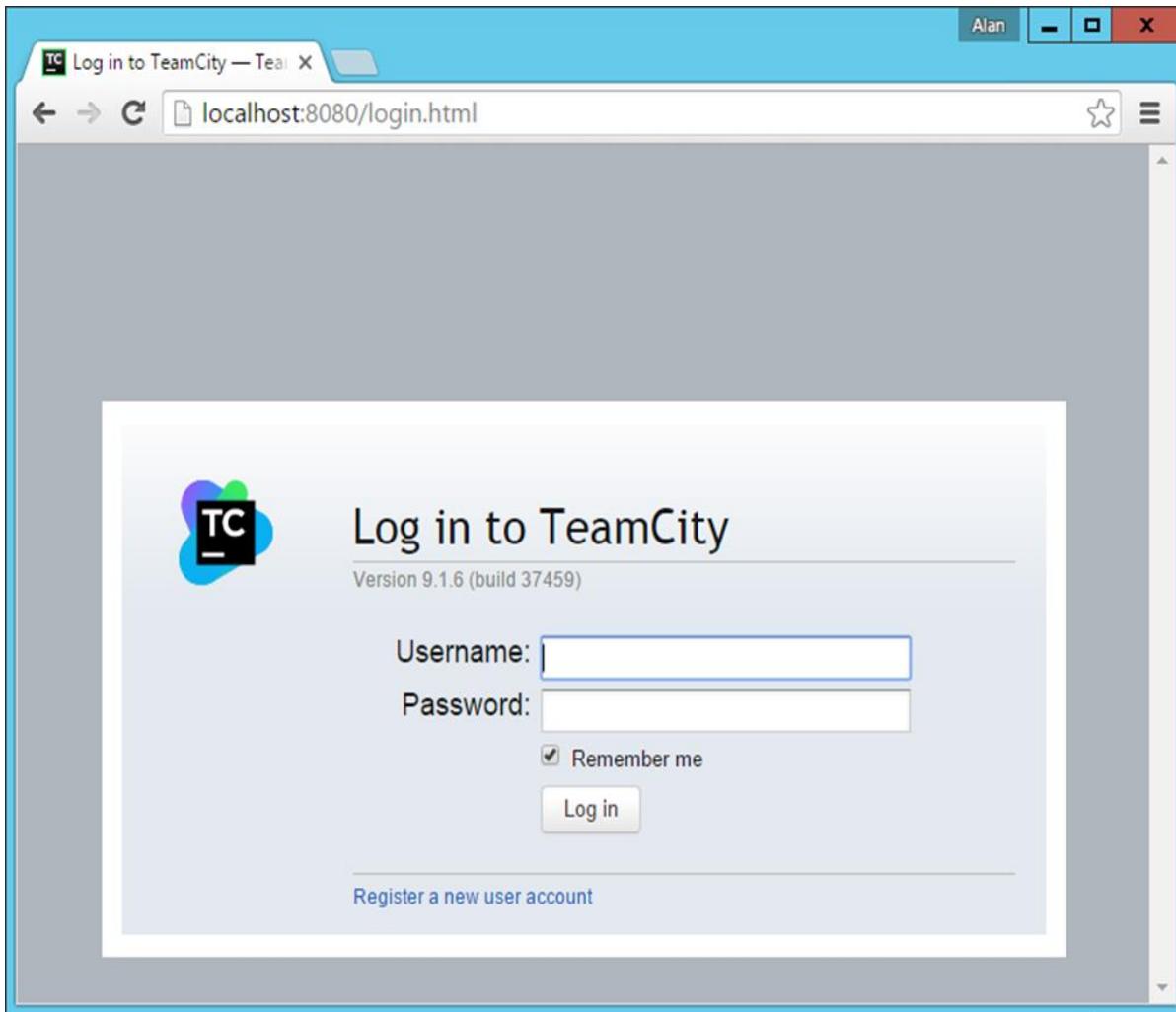
```
Administrator: Command Prompt
C:\Demo\Simple>git commit -m "First commit"
[master (root-commit) 23a4881] First commit
 20 files changed, 329 insertions(+)
 create mode 100644 Demo.aspx
 create mode 100644 Demo.aspx.cs
 create mode 100644 Demo.aspx.designer.cs
 create mode 100644 Properties/AssemblyInfo.cs
 create mode 100644 Simple.csproj
 create mode 100644 Simple.csproj.user
 create mode 100644 Web.Debug.config
 create mode 100644 Web.Release.config
 create mode 100644 Web.config
 create mode 100644 bin/Simple.dll
 create mode 100644 bin/Simple.dll.config
 create mode 100644 bin/Simple.pdb
 create mode 100644 obj/Debug/DesignTimeResolveAssemblyReferences
 create mode 100644 obj/Debug/Simple.csproj.FileListAbsolute.txt
 create mode 100644 obj/Debug/Simple.csprojResolveAssemblyReferen
 create mode 100644 obj/Debug/Simple.dll
 create mode 100644 obj/Debug/Simple.pdb
 create mode 100644 obj/Debug/TemporaryGeneratedFile_036C0B5B-148
ADCB23D92.cs
 create mode 100644 obj/Debug/TemporaryGeneratedFile_5937a670-0e6
21da3dd1.cs
 create mode 100644 obj/Debug/TemporaryGeneratedFile_E7A71F73-0F8
0B10BC5D3.cs

C:\Demo\simple>
```

# 11. CI – Creating a Project in TeamCity

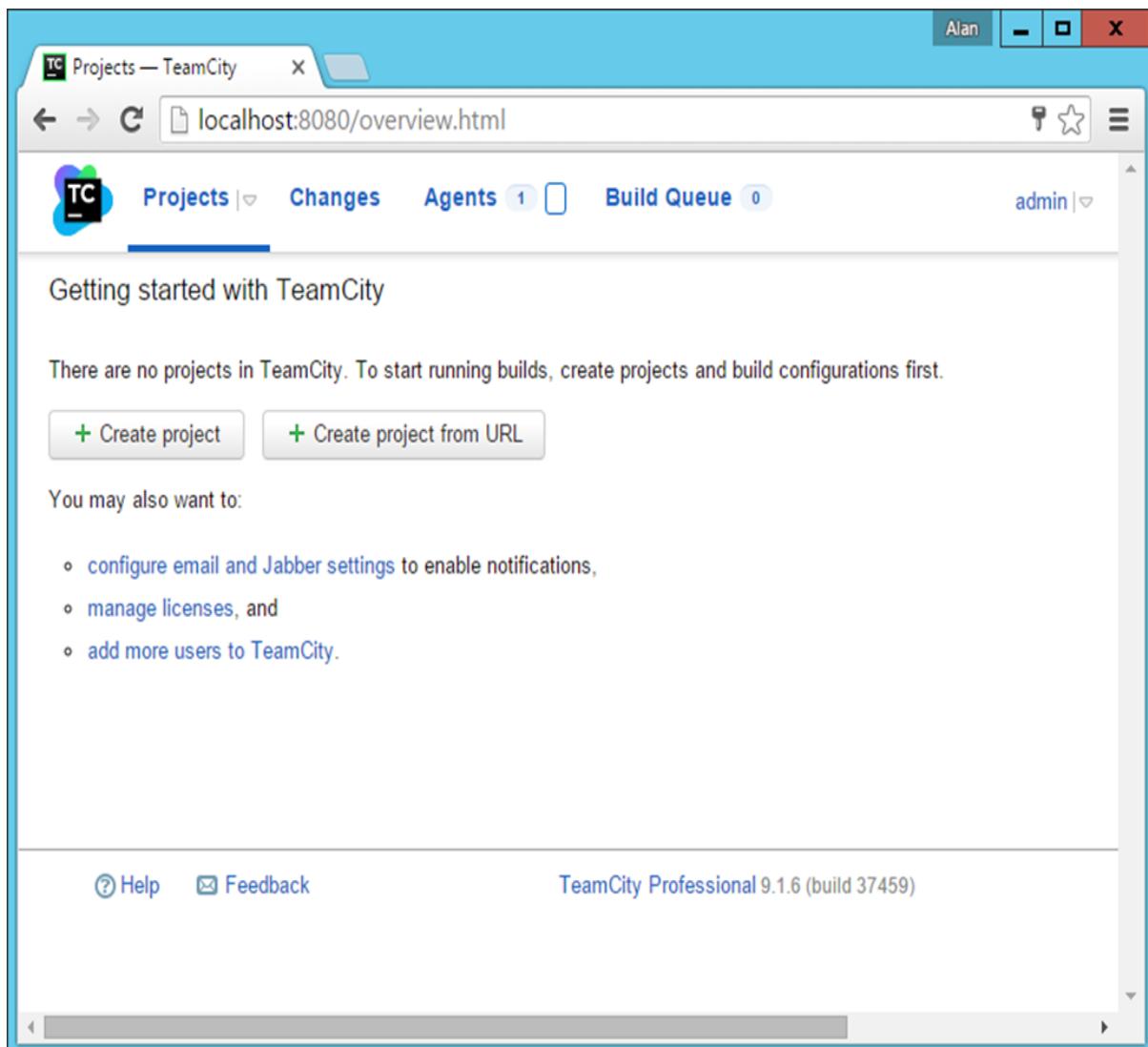
Now that we have our source code in the Git repository and all of our initial code works on the build server, it is time to create a project in our Continuous Integration server. This can be done via the following steps:

**Step 1:** Login to the TeamCity software. Go to the url on your Continuous Integration server – <http://localhost:8080/login.html>



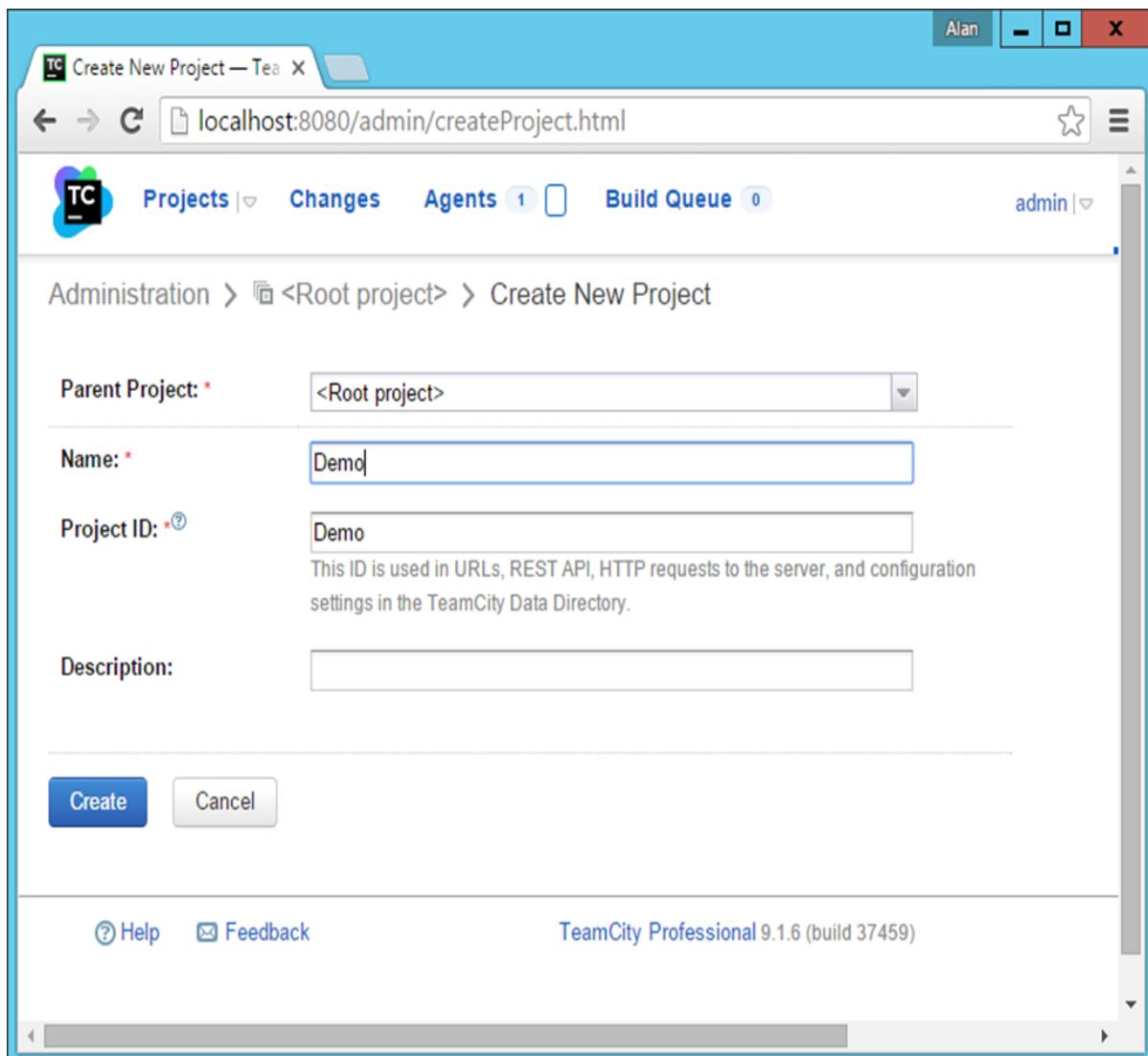
Enter the admin credentials and login to the server.

**Step 2:** Once logged in, you will be presented with the home screen. Click **Create Project** to start a new project.



The screenshot shows the TeamCity home page. At the top, there's a navigation bar with tabs for 'Projects', 'Changes', 'Agents' (with 1 build in progress), and 'Build Queue' (with 0 builds). The user is logged in as 'admin'. Below the navigation, a heading says 'Getting started with TeamCity'. A message states: 'There are no projects in TeamCity. To start running builds, create projects and build configurations first.' Two buttons are visible: '+ Create project' and '+ Create project from URL'. Below these buttons, a section titled 'You may also want to:' lists three items: 'configure email and Jabber settings to enable notifications,' 'manage licenses, and' and 'add more users to TeamCity.' At the bottom of the page, there are links for 'Help' and 'Feedback', and the text 'TeamCity Professional 9.1.6 (build 37459)'.

**Step 3:** Give a name for the project and click Create to start the project. In our case, we are giving the name as 'Demo' to our project as shown in the following screenshot.



**Step 4:** The next step is to mention the Git repository which will be used in our project. Remember that in a Continuous Integration environment, the CI server needs to pick up the code from the Git enabled repository. We have already enabled our project folder to be a Git enabled repository in the earlier step. In TeamCity, you need to create a **VCS root**. For this, click VCS Roots in the project's main screen.

The screenshot shows the TeamCity administration interface for a project named 'Demo'. The 'General Settings' tab is selected in the left sidebar. The 'Name:' field is set to 'Demo'. The 'Project ID:' field is also set to 'Demo', with a tooltip explaining it is used in URLs, REST API, and HTTP requests. The 'Description:' field is empty. At the bottom, there are 'Save' and 'Cancel' buttons. Below the settings, a section titled 'Build Configurations (20 left)' is shown, stating there are no build configurations in this project. It includes buttons for 'Create build configuration' and 'Create build configuration from URL'.

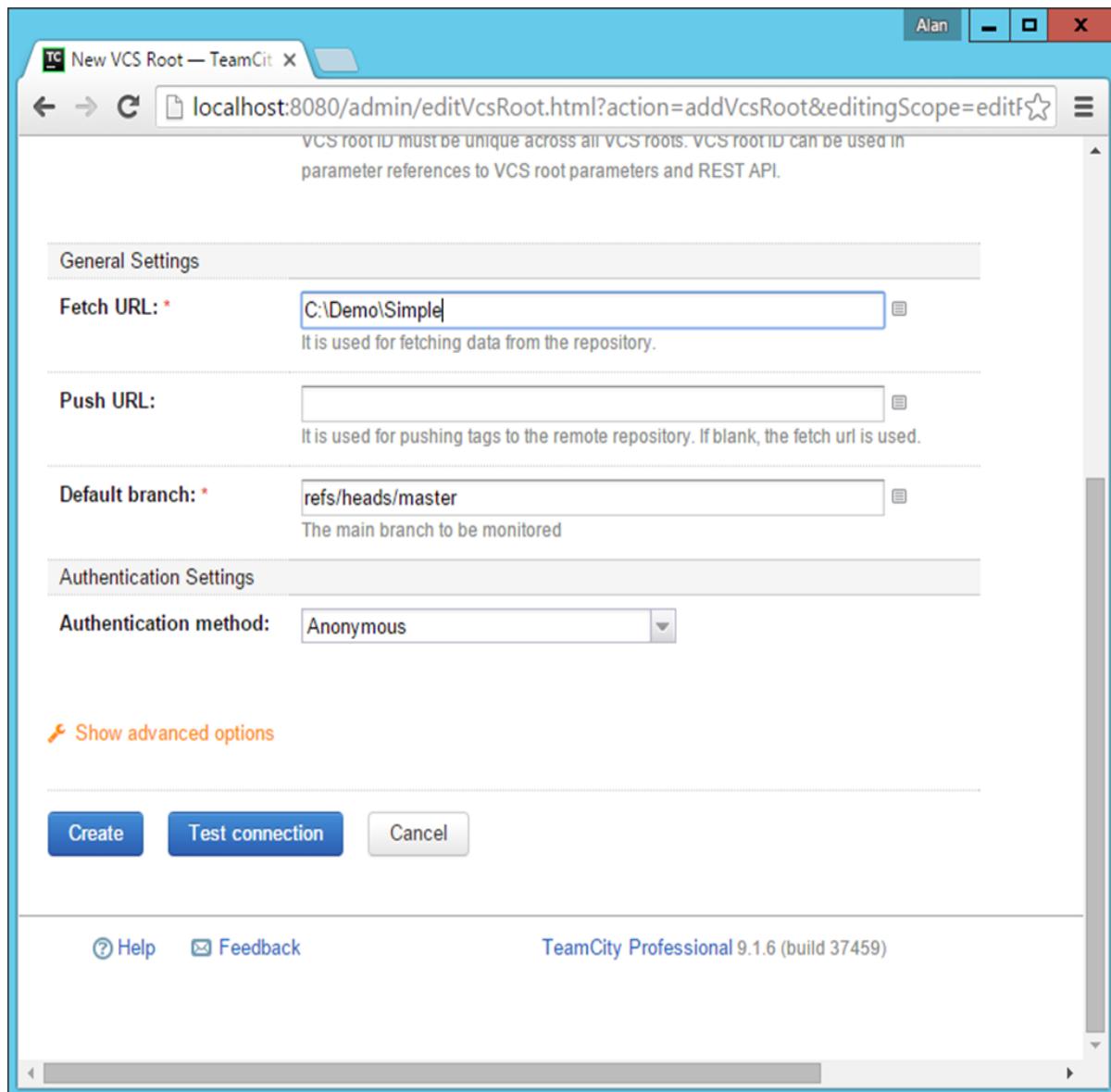
**Step 5:** In the screen that comes up next, click **Create VCS root** as shown in the following screenshot.

The screenshot shows the TeamCity administration interface for a project named 'Demo'. The left sidebar lists various project settings: Project Settings, General Settings, **VCS Roots**, Report Tabs, Parameters, Builds Schedule, Issue Trackers, Shared Resources, Meta-Runners, SSH Keys, Maven Settings, Clean-up Rules, and Versioned Settings. The 'VCS Roots' tab is currently selected. The main content area is titled 'VCS Roots' and contains a brief description: 'A VCS Root is a set of settings defining how TeamCity communicates with a version control system of a build'. Below this description is a prominent green button labeled '+ Create VCS root'. At the bottom of the page, there is a message box stating 'Created one minute ago by admin (view history)'. The top of the window shows the URL 'localhost:8080/admin/editProject.html?projectId=Demo&tab=projectVcsRoots' and the user 'admin'.

**Step 6:** In the next screen that comes up, perform the following steps:

- Mention the type of VCS as Git.
- Give a name for the VCS root, this can be any friendly name. We have given the name as **App**.
- Give the Fetch url as **C:\Demo\Simple** – This is **out git** enabled repository.
- If you scroll down the screen, you will get a Test connection button. Click it to ensure that you can successfully connect to the Git enabled repository.

The screenshot shows the 'New VCS Root' configuration page in TeamCity. The URL in the browser is `localhost:8080/admin/editVcsRoot.html?action=addVcsRoot&editingScope=editF`. The page has sections for 'Type of VCS' (set to Git), 'VCS Root' (with 'VCS root name' set to App and a note about uniqueness), 'VCS root ID' (set to Demo\_App with a note about uniqueness across all roots), 'General Settings' (Fetch URL set to C:\Demo\Simple with a note about fetching data, Push URL empty, and Default branch set to refs/heads/master with a note about monitoring).



**Step 7:** Click Create and you will now see your repository registered as shown in the following image.

The screenshot shows the TeamCity interface for managing VCS Roots. The URL is <localhost:8080/admin/editProject.html?projectId=Demo&tab=projectVcsRoots&v>. The page title is "Demo Project > VCS Root". The top navigation bar includes "Agents 1", "Build Queue 0", "admin | Administration", and a search bar. Below the navigation is a breadcrumb trail: "project > Demo". On the right, there are "Actions" and "Project Home" buttons.

**VCS Roots**

A VCS Root is a set of settings defining how TeamCity communicates with a version control system to monitor changes and get sources of a build<sup>②</sup>

**Create VCS root**

Git VCS root successfully created.

Filter:  Filter  Show unused VCS roots only

Name	Usages
(jetbrains.git) App	not monitored Edit Delete no usages

**Step 8:** The next step is to create a build configuration which will be used to build the project. Go to your project screen in **TeamCity -> General Settings**. Click Create Build Configuration.

The screenshot shows the TeamCity 'General Settings' page for a 'Demo' project. The 'General Settings' tab is selected. On the left, a sidebar lists various settings like VCS Roots, Report Tabs, Parameters, etc. A message at the bottom left indicates the project was created 9 minutes ago by admin. The main area shows fields for 'Name' (set to 'Demo') and 'Project ID' (set to 'Demo'). Below these are 'Description' and 'Save' / 'Cancel' buttons. To the right, a section titled 'Build Configurations (20 left)' displays a message stating there are no build configurations in this project, with 'Create build configuration' and 'Create build configuration from URL' buttons. At the bottom, a 'Build Configuration Templates' section is visible.

**Step 9:** In the following screen, give a name for the Build Configuration. In our case we have named it as **DemoBuild** and then click Create.

The screenshot shows the 'Create Build Configuration' dialog box in TeamCity. The URL in the browser is `localhost:8080/admin/createBuildType.html?projectId=Demo&init=1`. The navigation path is Administration > <Root project> > Demo > Create Build Configuration. The 'Name:' field contains 'DemoBuild'. The 'Build configuration ID:' field also contains 'DemoBuild' with a note below explaining it is used in URLs, REST API, HTTP requests, and configuration settings. There is a 'Description:' field which is empty. At the bottom are 'Create' and 'Cancel' buttons. The footer includes links for Help and Feedback, and information about TeamCity Professional 9.1.6 (build 37459).

**Step 10:** In the next screen that comes up, you will be asked to choose the **VCS repository** which was created in the earlier steps. So choose the name '**App**' and click Attach.

The screenshot shows the 'Build Configuration Settings' page for 'DemoBuild'. The 'Version Control Settings' section is active. Under 'Attach existing VCS root:', the dropdown is set to 'App' and the 'Attach' button is visible. A yellow status bar at the top right says 'Build configuration successfully created. You can now configure VCS roots.' On the left sidebar, 'Build Steps' is highlighted. At the bottom left, there's a note: 'Created moments ago by admin (view history)'.

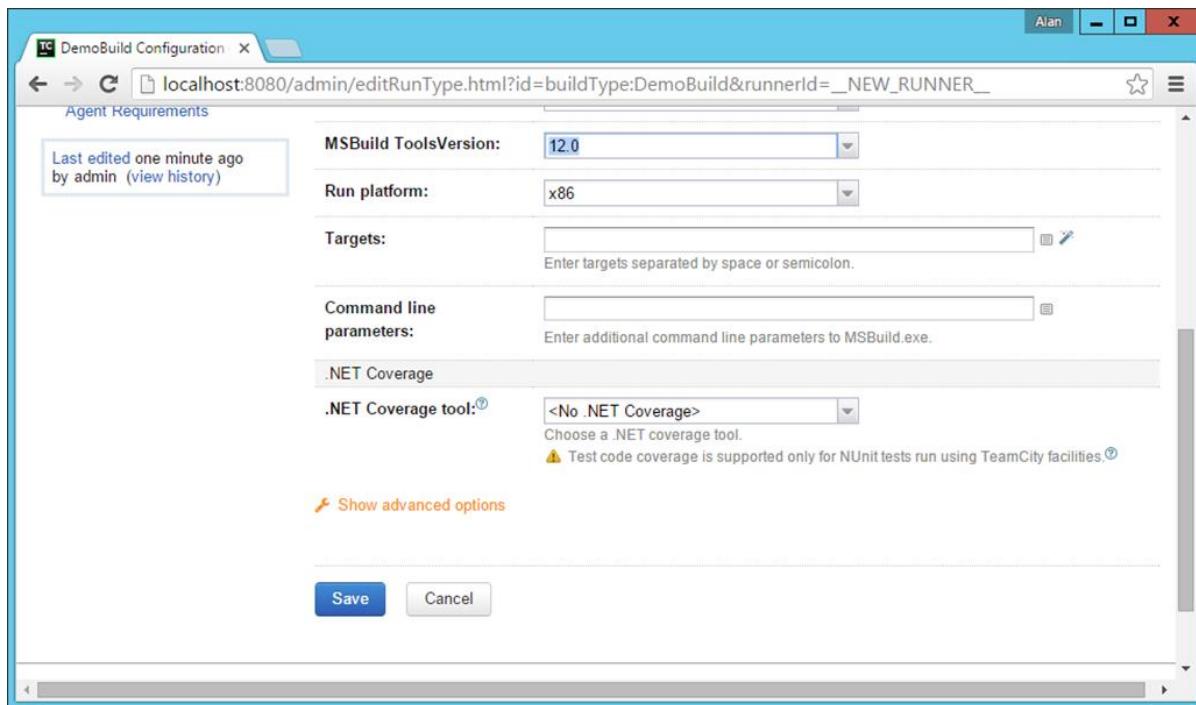
**Step 11:** Now in the next screen that pops up, we need to configure the build steps. So click the '**configure build steps manually**' hyperlink.

The screenshot shows the 'Build Configuration Settings' page for 'DemoBuild'. The 'Build Steps' section is active. A yellow status bar at the top right says 'VCS root "App" attached.' Another yellow status bar below it says 'You can now configure build steps.' The 'Auto-detected Build Steps' section contains the text: 'Build steps and their settings are detected automatically by scanning VCS repository. You can configure build steps manually if auto-detect did not find relevant build steps.' Below that, it says 'No build steps found, configure build steps manually.' On the left sidebar, 'Build Steps' is highlighted. At the bottom left, there's a note: 'Last edited moments ago by admin (view history)'.

**Step 12:** In the next build screen, we need to enter the following details:

- Choose the Runner type as MSBuild.
- Give an optional name for the step name.
- Give the name of the file which needs to be built. When we specify MSbuild in the earlier sections, we normally see that we give the option of **Simple.csproj**. The same thing is needed to be specified here.
- Choose the MSBuild version as 'Microsoft Build Tools 2013'.
- Choose the **MSBuild ToolsVersion** as 12.0.
- Scroll down the page to Save the settings.

The screenshot shows the 'New Build Step' configuration page in TeamCity. The left sidebar has tabs for 'General Settings', 'Version Control Settings', 'Build Steps' (which is selected), 'Triggers', 'Failure Conditions', 'Build Features', 'Dependencies', 'Parameters', and 'Agent Requirements'. A note at the bottom left says 'Last edited one minute ago by admin (view history)'. The main form has fields for 'Runner type:' (set to 'MSBuild'), 'Step name:' (set to 'Build'), 'Build file path:' (set to 'Simple.csproj'), 'MSBuild version:' (set to 'Microsoft Build Tools 2013'), 'MSBuild ToolsVersion:' (set to '12.0'), 'Run platform:' (set to 'x86'), and 'Targets:' (empty). Buttons at the top right include 'Run ...', 'Actions', and 'Build Configuration Home'.



**Step 13:** In the next screen, click Run.

Build Step	Parameters Description
Build	MSBuild Build file: Simple.csproj Targets: default Execute: If all previous steps finished successfully

You will see the build of your application now in progress.

Demo :: DemoBuild > #1

localhost:8080/viewLog.html?buildId=2&

Projects Changes Agents 1 Build Queue 0 admin | Administration

Demo > DemoBuild > #1 (17 Mar 16 07:23)

Run ... Actions Edit Configuration Settings

Overview Changes Build Log Parameters Artifacts

Started: 17 Mar 16 07:23 Stop Agent: WIN-50GP30FG075-1

Progress: 17 Mar 16 07:23 (7s) Triggered: by you on 17 Mar 16 07:23

Thread dump: View thread dump

Running step: Publishing internal artifacts: Publishing 1 file using [ArtifactsCachePublisher]

Help Feedback TeamCity Professional 9.1.6 (build 37459) License agreement

You should get a successful screen, which is a good sign that your solution is building properly.

Demo :: DemoBuild > #1

localhost:8080/viewLog.html?buildId=2&

Projects Changes Agents 1 Build Queue 0 admin | Administration

Demo > DemoBuild > #1 (17 Mar 16 07:23)

Run ... Actions Edit Configuration Settings

Overview Changes Build Log Parameters Artifacts First recorded build | All history | Last recorded build

Result: Success Agent: WIN-50GP30FG075-1

Time: 17 Mar 16 07:23:49 - 07:23:57 (7s) Triggered by: you on 17 Mar 16 07:23

Help Feedback TeamCity Professional 9.1.6 (build 37459) License agreement

You can also go to your build log to see all the steps that were covered by the Continuous Integration server as shown in the following screenshot.

The screenshot shows the TeamCity Professional 9.1.6 interface. The title bar reads "Demo :: DemoBuild > #1". The browser address bar shows "localhost:8080/viewLog.html?buildId=2&buildTypeId=DemoBuild&tab=buildLog#\_". The main navigation menu includes "Projects", "Changes", "Agents", "Build Queue", "admin", "Administration", and a search bar. Below the menu, the project structure is shown as "Demo > DemoBuild > #1 (17 Mar 16 07:23)". The "Build Log" tab is selected. The log content displays the following build steps:

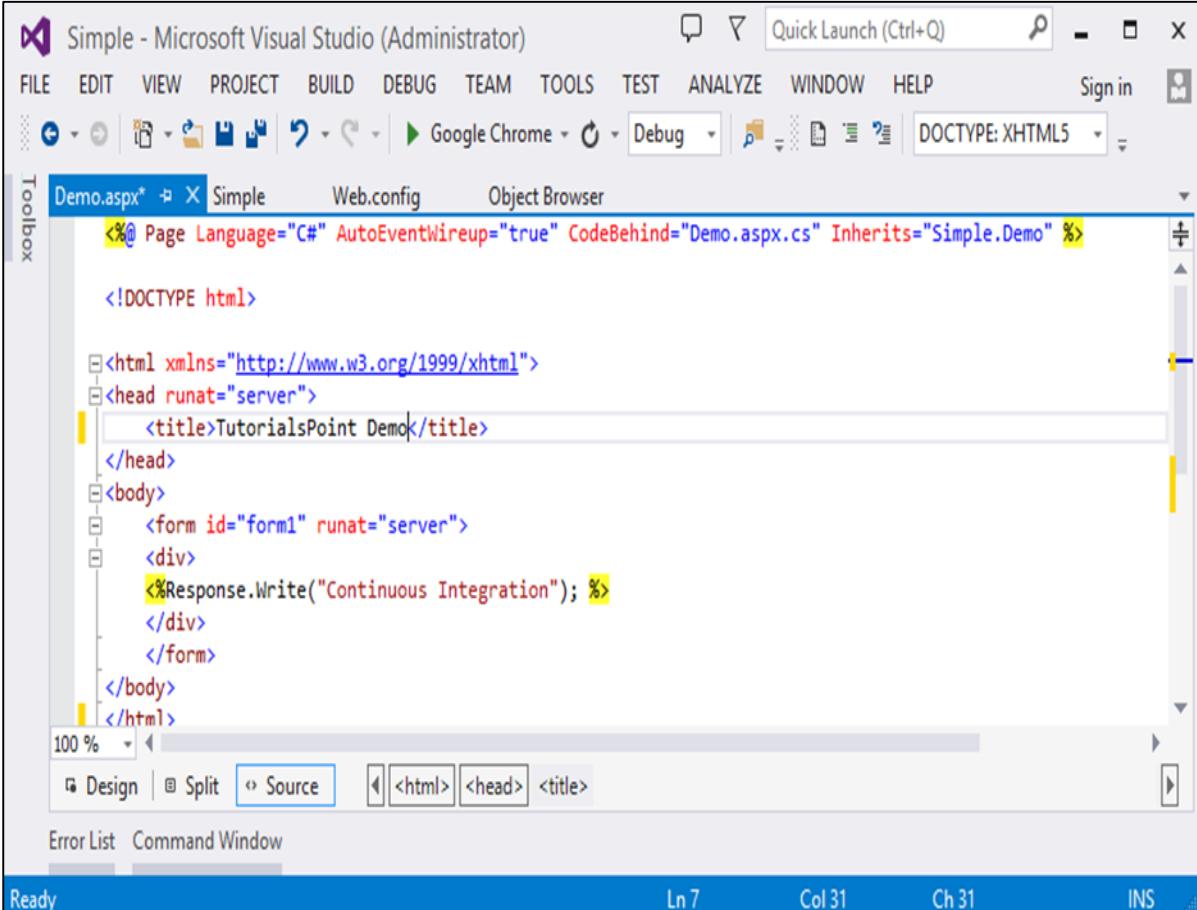
```
[07:23:49] + Collecting changes in 1 VCS root
[07:23:50]   Clearing temporary directory: C:\TeamCity\buildAgent\temp\buildTmp
[07:23:50] + Publishing internal artifacts
[07:23:50]   Checkout directory: C:\TeamCity\buildAgent\work\c87b4eddb2be6bc1
[07:23:50] + Updating sources: server side checkout (1s)
[07:23:52] + Step 1/1: Build (MSBuild) (4s)
[07:23:56] + Publishing internal artifacts
[07:23:57]   Build finished
```

At the bottom, there are links for "Help", "Feedback", "TeamCity Professional 9.1.6 (build 37459)", and "License agreement".

## 12. CI – Defining Tasks

Now that we have our base code in Git and a link to the Continuous Integration server, it's finally time to see the first step of Continuous Integration in action. This is done by defining tasks in the Continuous Integration server such as triggers, which makes the entire Continuous Integration Process as seamless as possible. Let's make a change to our code in Visual Studio.

**Step 1:** Go to the **Demo.aspx** page in Visual Studio and make a change to the title of the page.



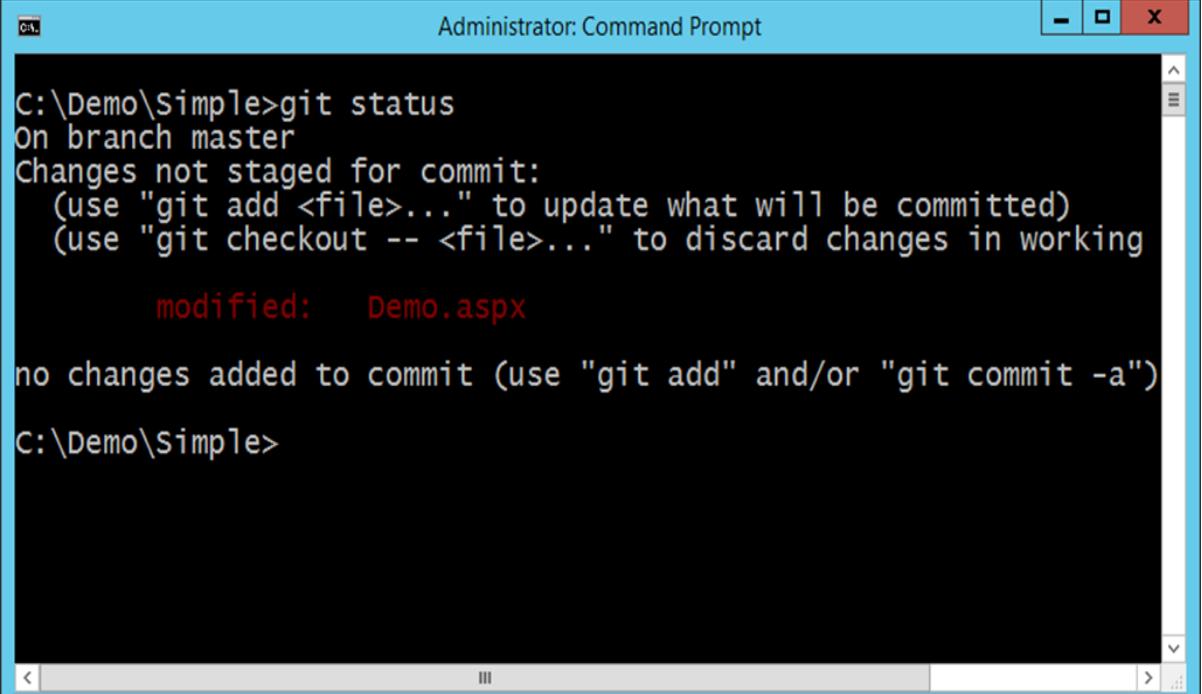
The screenshot shows the Microsoft Visual Studio interface with the title bar "Simple - Microsoft Visual Studio (Administrator)". The menu bar includes FILE, EDIT, VIEW, PROJECT, BUILD, DEBUG, TEAM, TOOLS, TEST, ANALYZE, WINDOW, and HELP. The toolbar has icons for file operations like Open, Save, and Print. A status bar at the bottom shows "Ready", "Ln 7", "Col 31", "Ch 31", and "INS". The main area displays the source code for "Demo.aspx". The code includes an ASPX header, an HTML document type declaration, an XML namespace declaration, a head section with a title containing "TutorialsPoint Demo", and a body section with a form and a response write statement. The code editor has syntax highlighting and a tool palette on the left.

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Demo.aspx.cs" Inherits="Simple.Demo" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>TutorialsPoint Demo</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <%Response.Write("Continuous Integration"); %>
        </div>
    </form>
</body>
</html>
```

**Step 2:** If we query our Git repository via the **git status** command, you will in fact see that the **Demo.aspx** file has been modified.



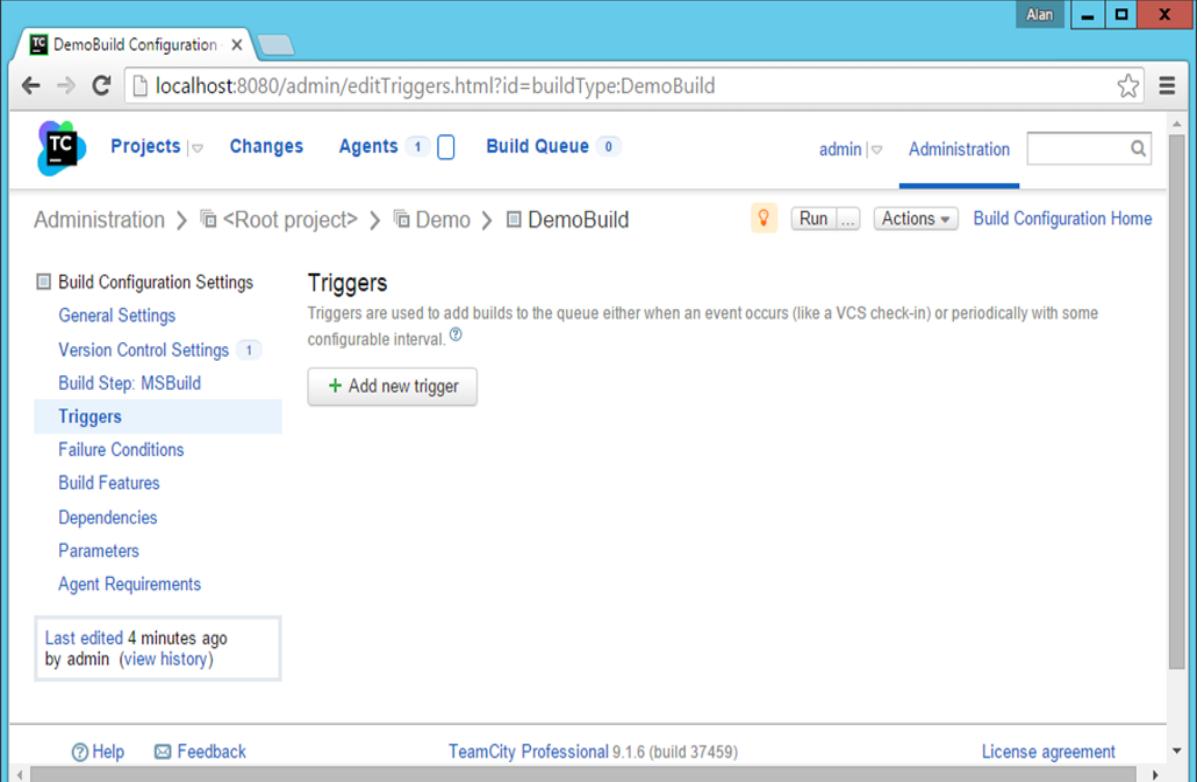
```
C:\Demo\simple>git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working

        modified:   Demo.aspx

no changes added to commit (use "git add" and/or "git commit -a")
C:\Demo\simple>
```

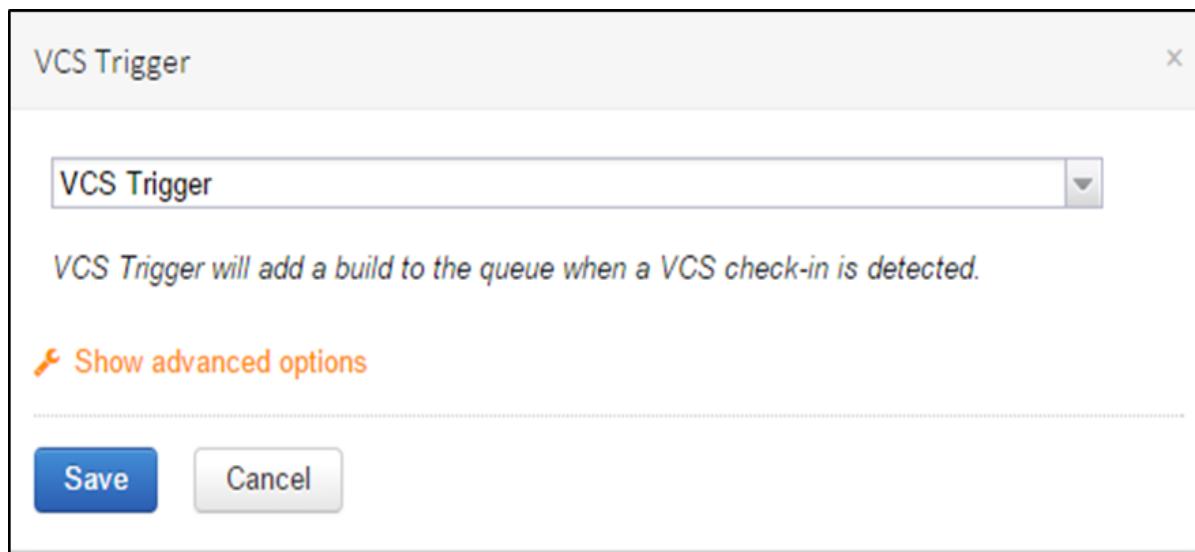
Now we need to ensure that every change in our code should trigger a build in our continuous integration server. For this we need to do the following changes.

**Step 3:** Go to your project dashboard and click the triggers section and click **Add new trigger**.



The screenshot shows the TeamCity interface for managing a build configuration named 'DemoBuild'. The 'Administration' tab is selected. In the left sidebar, the 'Triggers' section is currently active, indicated by a blue background. Below it, other settings like 'General Settings' and 'Version Control Settings' are listed. At the bottom of the sidebar, there is a note: 'Last edited 4 minutes ago by admin (view history)'. On the right, the main content area displays the 'Triggers' configuration page, which includes a brief description of what triggers are used for and a prominent '+ Add new trigger' button.

**Step 4:** In the next screen that comes up, choose **VCS trigger**, which will be used to create a trigger so that when a check-in is made to the repository, a build will be triggered.



**Step 5:** Click **Show Advanced Options** and ensure the options shown in the following screenshot are selected.

**Step 6:** Click Save. You will now see the trigger successfully registered as shown in the following screenshot.

The screenshot shows the Jenkins 'DemoBuild Configuration' page. In the left sidebar, 'Triggers' is selected. The main content area shows a table with one row for 'VCS Trigger'. The 'Trigger' column contains the rule '+root=Demo\_App;\*\*'. The 'Parameters Description' column notes that it triggers one build per each VCS check-in.

Trigger	Parameters Description
VCS Trigger	VCS trigger rules: +root=Demo_App;** Triggers one build per each VCS check-in (include several check-ins in build if they are from the same committer)

**Step 7:** Now it's time to check in our code into the Git repository and see what happens. So let's go to our command prompt and issue the **git add** command to stage our changed files.

```
C:\Demo\simple>git add .
```

**Step 8:** Now issue the **git commit** command, and it will push the changes into the Git repository.

```
C:\Demo\Simple>git commit -m "New commit"
[master e830db0] New commit
 1 file changed, 3 insertions(+), 1 deletion(-)

C:\Demo\Simple>
```

**Step 9:** If you now go to your Projects Overview screen, you will now see a new build would have been triggered and run.

A screenshot of a web browser displaying the TeamCity Professional 9.1.6 interface. The title bar shows "Project: Demo &gt; Overview" and the URL "localhost:8080/project.html?projectId=Demo&amp;tab=projectOverview". The main navigation menu includes "Projects" (selected), "Changes", "Agents 1", "Build Queue 0", and "admin". Below the menu, the project "Demo" is selected. A sub-menu for "Demo" shows "Overview" (selected), "Change Log", "Statistics", "Current Problems", "Investigations", and "Muted Problems". Under "Demo", a section for "DemoBuild" is expanded, showing build #3 which was successful ("Success") with no artifacts and one author ("alashro (1)"). At the bottom left are links for "Help" and "Feedback", and at the bottom right is the text "TeamCity Professional 9.1.6 (build 37459)".

If you see the **Change log Tab**, you will see the **git comment** which triggered the build.

The screenshot shows a web interface for a continuous integration system. At the top, there's a header with the project name 'Demo' and a user 'Alan'. Below the header, there are tabs for 'Projects', 'Changes', 'Agents', 'Build Queue', and 'admin'. The 'Changes' tab is active. Under the 'Changes' tab, there are sub-tabs: 'Overview', 'Change Log' (which is selected), 'Statistics', 'Current Problems', 'Investigations', and 'Muted Problems'. A search bar allows filtering changes by user ('<All users>'), with a 'Filter' button and a link to 'Advanced search'. There are also checkboxes for 'Show graph' (checked) and 'Show files'. Below the search area, it says 'Page 1 of 1 (Found 2 changes)'. Two commits are listed:

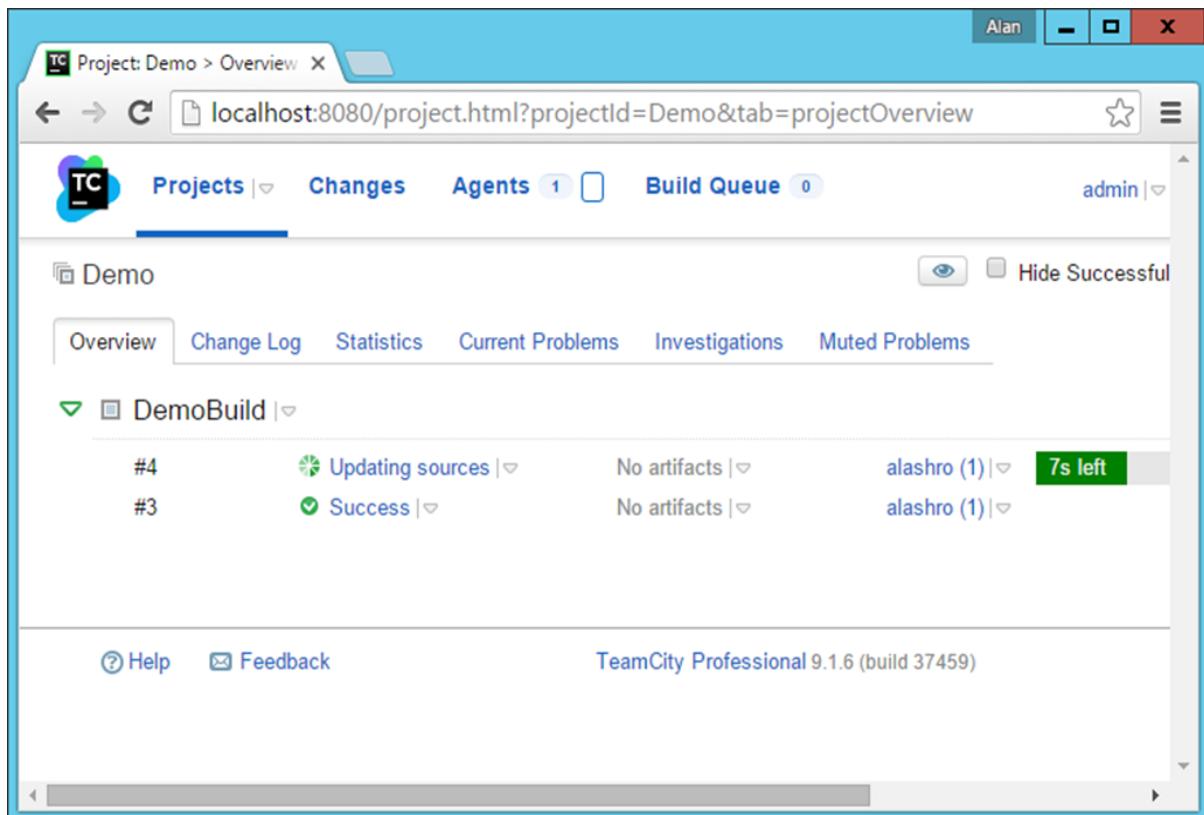
Commit	Author	Files	Hash
refs/heads/master New commit	alashro	1 file	b2e252
New commit	alashro	1 file	e830db

Let's try it one more time. Let's make another change to the **Demo.aspx** file. Let's carry out a **git add** command and a **git commit** command with the following commit message.

The screenshot shows a Windows Command Prompt window titled 'Administrator: Command Prompt'. The prompt is at 'C:\Demo\simple>'. The user runs the command 'git commit -m "Second commit"'. The output shows the commit was successful, adding one file with one insertion and one deletion. The prompt then changes to 'C:\Demo\simple>'.

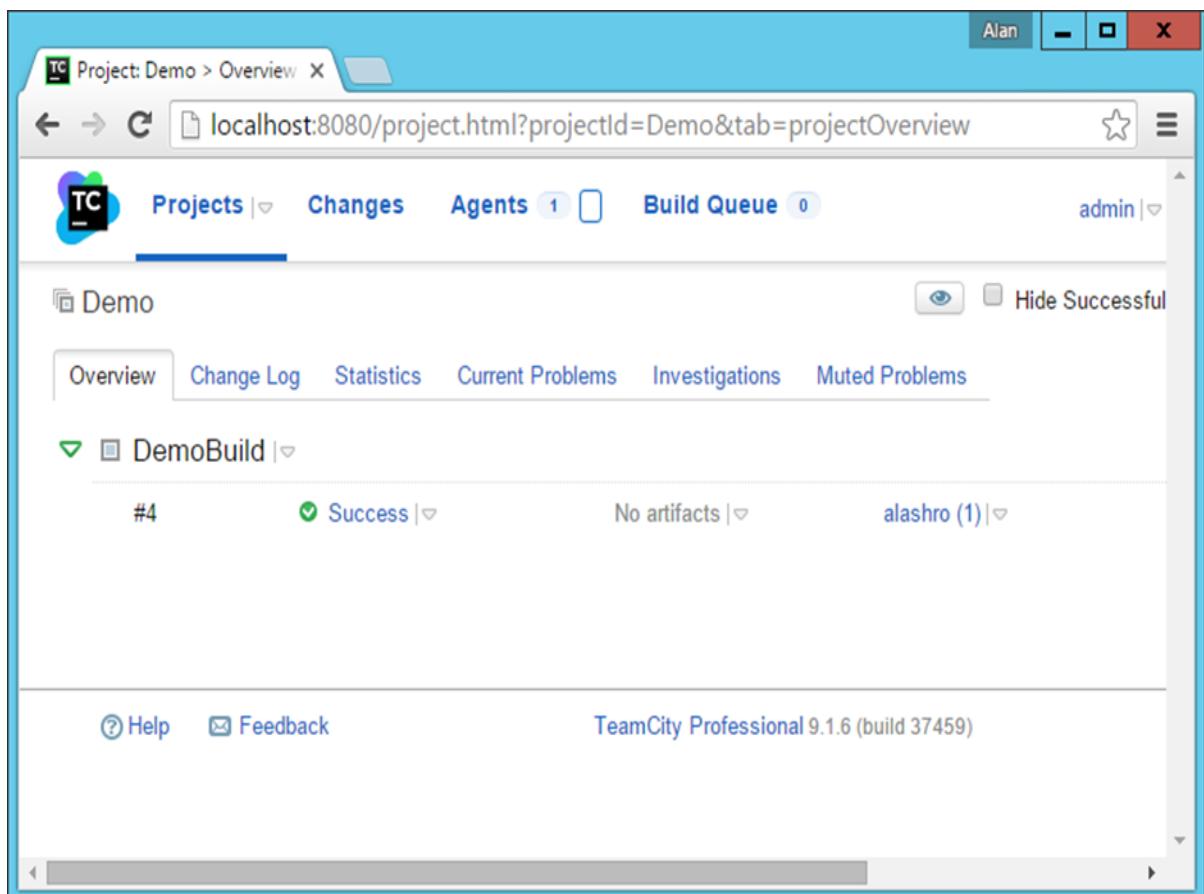
```
C:\Demo\simple>git commit -m "Second commit"
[master 038d0c0] Second commit
 1 file changed, 1 insertion(+), 1 deletion(-)

C:\Demo\simple>
```



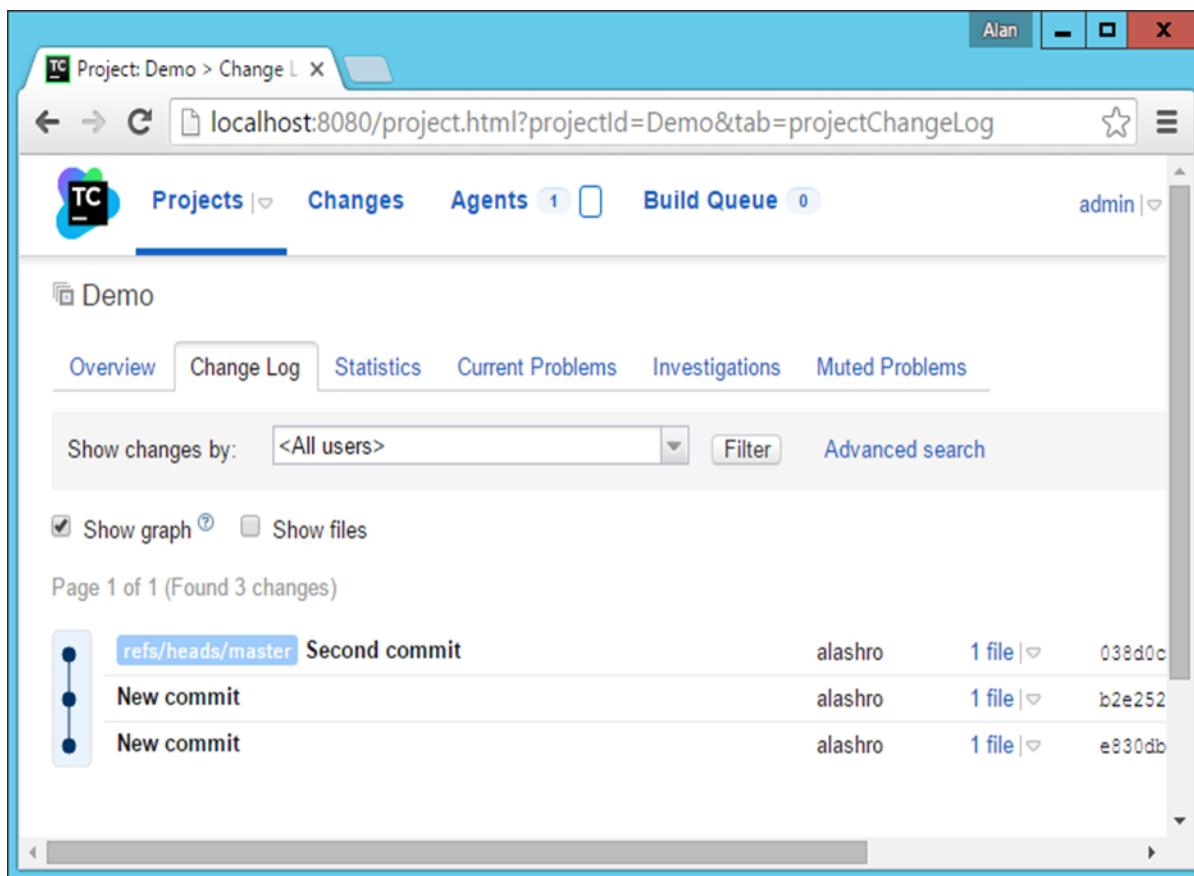
The screenshot shows the TeamCity interface for the 'Demo' project. At the top, there are tabs for 'Projects', 'Changes', 'Agents' (1), and 'Build Queue' (0). The 'Build Queue' tab is selected. On the right, the user is logged in as 'admin'. Below the tabs, the project name 'Demo' is displayed, with a 'Hide Successful' button. A navigation bar below the project name includes 'Overview', 'Change Log', 'Statistics', 'Current Problems', 'Investigations', and 'Muted Problems'. The 'Overview' tab is selected. Under the 'Demo' section, there is a 'DemoBuild' node expanded, showing two build items: #4 (Updating sources) and #3 (Success). Build #3 has a green status icon and is listed as 'Success'. To the right of each build item, there is a column for artifacts ('No artifacts') and agents ('alashro (1)'). A green progress bar indicates '7s left' for build #4. At the bottom of the interface, there are links for 'Help' and 'Feedback', and the text 'TeamCity Professional 9.1.6 (build 37459)'.

You will now see a build being automatically triggered in the Project dashboard in TeamCity.



This screenshot shows the same TeamCity interface as the previous one, but with a key difference: build #4 has completed successfully. The 'Build Queue' tab is still selected. The 'DemoBuild' node is expanded, and build #4 is now listed as 'Success' with a green checkmark icon. The other details (agents and artifacts) remain the same. The bottom of the screen shows the 'TeamCity Professional 9.1.6 (build 37459)' footer.

The build will show a success message.



The screenshot shows a web browser window with the URL `localhost:8080/project.html?projectId=Demo&tab=projectChangeLog`. The page title is "Project: Demo > Change Log". The top navigation bar includes links for "Projects", "Changes", "Agents 1", "Build Queue 0", and a user account for "admin". Below the navigation, the project name "Demo" is displayed. A tab menu at the top of the main content area includes "Overview", "Change Log" (which is selected), "Statistics", "Current Problems", "Investigations", and "Muted Problems". A search bar allows filtering by users ("Show changes by: <All users>") and provides "Filter" and "Advanced search" options. A checkbox "Show graph" is checked. The main content displays a commit history for the "refs/heads/master" branch:

Commit	Author	Files	Hash
Second commit	alashro	1 file	038d0c
New commit	alashro	1 file	b2e252
New commit	alashro	1 file	e830db

You will now see the message of 'Second commit' which was used when the change was committed to the **git repository**.

We have now successfully completed the first part of the Continuous Integration process.

# 13. CI – Build Failure Notifications

A Build Failure Notification is an event which is triggered whenever a build fails. The notification is sent to all key people whenever a build fails. The first important thing to do in such a case is to ensure time is spent on the failed build to ensure the build passed. The following steps are used to ensure that the build notifications are put in place in TeamCity.

Following are the steps to set up email notifications in TeamCity.

**Step 1:** In TeamCity, go to your Project dashboard, click on Administration in the top right hand corner. You will then see the **Email Notifier** link in the left hand side. Click on this link to bring up the general settings for Email.

The screenshot shows the TeamCity Administration interface. The top navigation bar includes 'Projects — TeamCity', a user icon 'Alan', and standard window controls. Below the bar, the URL 'localhost:8080/admin/admin.html?item=projects' is visible. The main content area is titled 'Administration > Projects'. On the left, a sidebar lists 'Project-related Settings' with 'Projects' selected, followed by 'Build Time', 'Disk Usage', 'Server Health', and 'Audit'. Under 'User Management', there are links for 'Users' and 'Groups'. Under 'Integrations', there are links for 'NuGet' and 'Tools'. Under 'Server Administration', there are links for 'Global Settings', 'Authentication', 'Email Notifier' (which is highlighted in blue), 'Jabber Notifier', and 'Agent Cloud'. The main panel displays a message: 'You have 2 active projects with 1 build configuration. You can have a maximum of 20 build configurations (archived.)'. It includes a 'Filter:' input field, a 'Filter' button, and a 'Show archived' checkbox. A tree view shows the project structure: '<Root project>' contains 'Demo'. There are also 'Create project' and 'Create project from URL' buttons at the top of the main panel.

**Step 2:** Next step is to enter the details of a valid **SMTP Server**. Gmail provides a free SMTP facility, which can be used by anyone. So we can enter those details in the next screen that comes up as shown in the following screenshot.

- SMTP Host – smtp.gmail.com
- SMTP port no – 465

- Send email messages from and SMTP login – This should be a valid Gmail id
- SMTP password – Valid password for that Gmail id
- Secure connection – Put this as SSL

The screenshot shows the TeamCity administration interface with the URL `localhost:8080/admin/admin.html?item=email`. The left sidebar has a tree view with nodes like Projects, Changes, Agents, Build Queue, Admin, and Email Notifier (which is selected). The main content area is titled "Email Notifier" and contains the following configuration:

Project-related Settings	The notifier is enabled <input type="button" value="Disable"/>
Projects	SMTP host: * <input type="text" value="smtp.gmail.com"/>
Build Time	SMTP port: * <input type="text" value="465"/>
Disk Usage	Send email messages from: * <input type="text" value="adusr2017@gmail.com"/>
Server Health	SMTP login: <input type="text" value="adusr2017@gmail.com"/>
Audit	SMTP password: <input type="password" value="....."/>
User Management	Secure connection: <input type="button" value="SSL"/>
Users	
Groups	
Integrations	
NuGet	
Tools	
The templates for Email notifications can be customized.	
Server Administration	
Global Settings	<input type="button" value="Save"/> <input type="button" value="Test connection"/>
Authentication	
Email Notifier	
Jabber Notifier	
Agent Cloud	
Dispositions	

**Step 3:** Click **Test Connection** just to ensure that the settings are working properly. Then click **Save** to save the settings.

**Step 4:** The next step is to enable build notifications for a user. The first task is to create a user which will receive these build notifications. Go to your project dashboard and choose the **Users Option**.

The screenshot shows the TeamCity administration interface with the URL `localhost:8080/admin/admin.html?item=projects`. The top navigation bar includes tabs for 'Projects' (selected), 'Changes', 'Agents 1', 'Build Queue 0', and 'Administration'. A user 'Alan' is logged in. The main content area is titled 'Administration > Projects'. On the left, a sidebar lists 'Project-related Settings' (Projects, Build Time, Disk Usage, Server Health, Audit), 'User Management' (Users, Groups), 'Integrations' (NuGet, Tools), and 'Server Administration' (Global Settings, Authentication, Email Notifier, Jabber Notifier, Agent Cloud). The 'Projects' section displays two active projects: '<Root project>' (Contains all other projects) and 'Demo'. Buttons for '+ Create project' and '+ Create project from URL' are visible.

**Step 5:** Create a new user. Enter the required username and password. Then Click the Create User button, which will be located at the bottom of the screen.

The screenshot shows a web application interface for creating a new user account. The title bar says "Create a New User Account" and the URL is "localhost:8080/admin/createUser.html". The top navigation bar includes "Projects", "Changes", "Agents", "Build Queue", and user information "admin | Admin". Below the navigation is a breadcrumb path: "Administration > Users > Create a New User Account".

**General**

- Username: \* demouser
- Name: demouser
- Email address: demousr2016@gmail.com
- New password: \*
- Confirm new password: \*

**Watched Builds and Notifications**

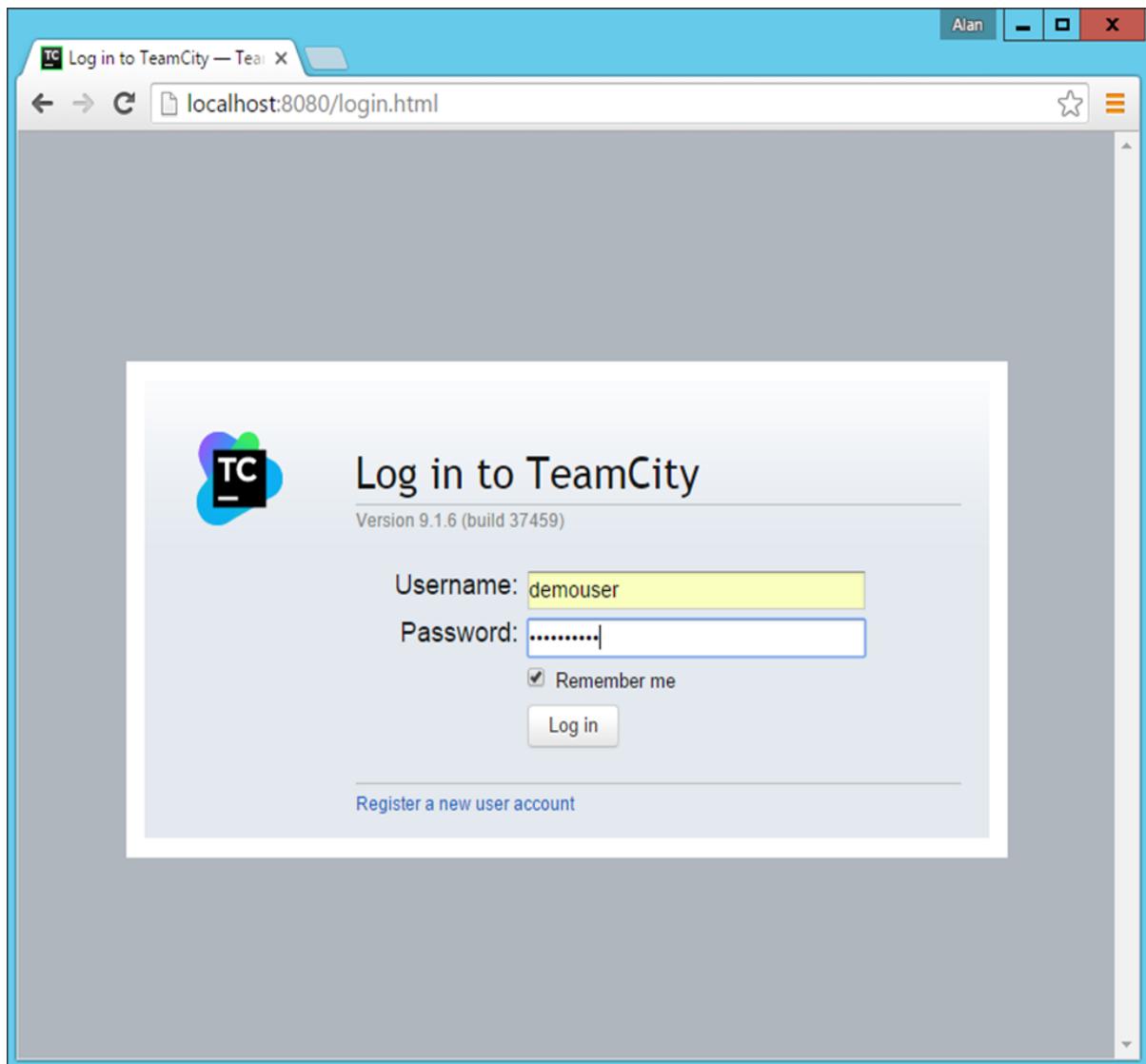
Jabber Notifier

Jabber account: [empty input field]

**Administrator status**

Give this user administrative privileges

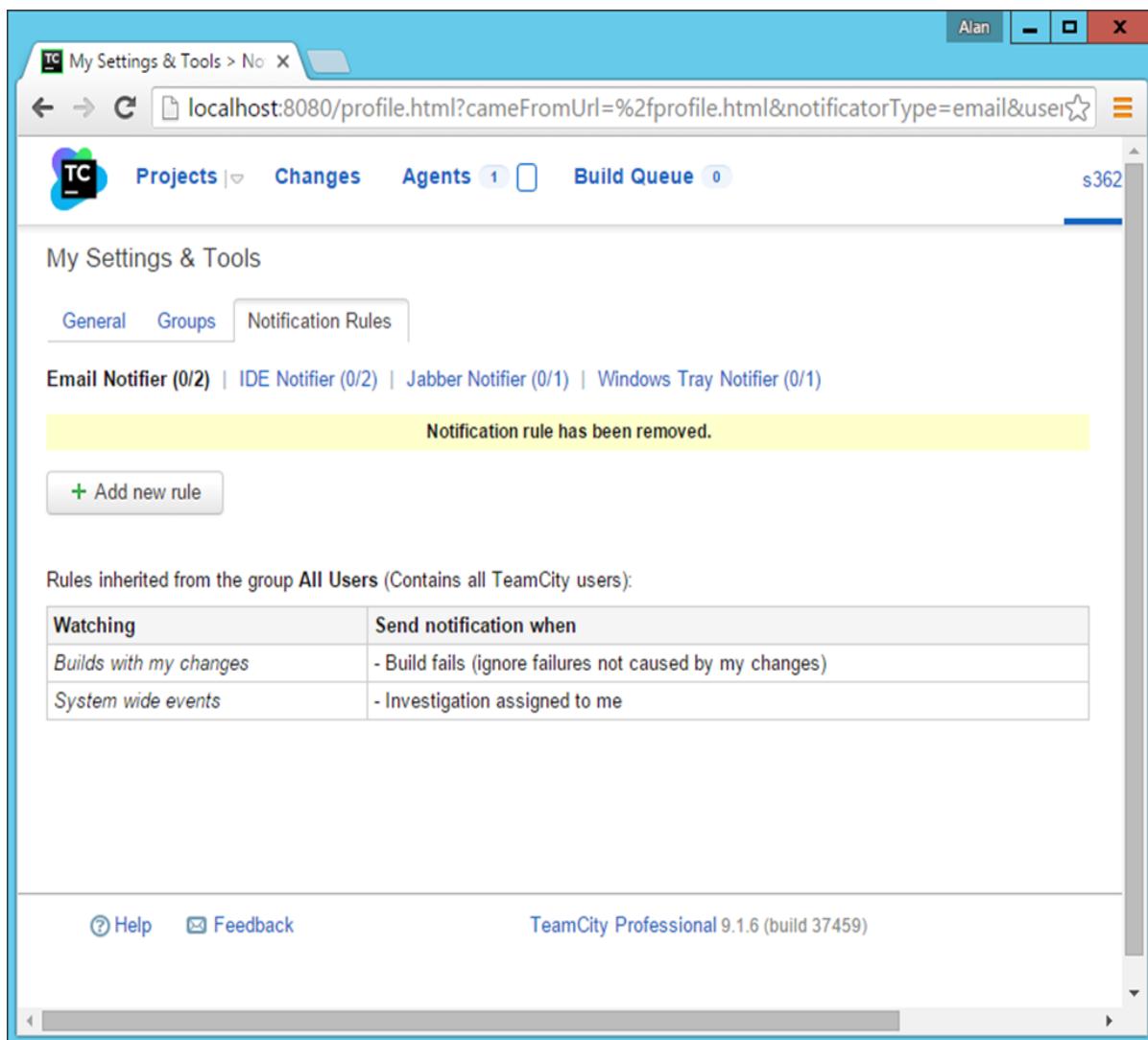
**Step 6:** Now login to the TeamCity system with this new user id and password.



**Step 7:** After you log in, you will be presented with the General settings of the user. In the Email Notifier section, click Edit.

The screenshot shows the 'My Settings & Tools' page in TeamCity. The top navigation bar includes 'Projects', 'Changes', 'Agents 1', 'Build Queue 0', and a user dropdown for 'Alan'. Below the navigation is a sub-menu with 'My Settings & Tools' selected. The main content area has tabs for 'General', 'Groups', and 'Notification Rules', with 'General' currently active. A yellow banner at the top says 'Welcome to TeamCity! Please fill in your profile. To start working with TeamCity go to [Overview page](#)'. The 'General' section contains fields for 'Username' (demouser), 'Name' (demouser), 'Email address' (demouser2016@gmail.com), 'Current password' (empty field), 'New password' (empty field), and 'Confirm new password' (empty field). To the right, under 'Watched Builds and Notifications', there are four sections: 'Email Notifier' (Edit link), 'IDE Notifier' (Edit link), 'Jabber Notifier' (Edit link), and 'Windows Tray Notifier' (Edit link). Each section has a message stating 'You are not watching any build configurations.'

**Step 8:** In the next screen that comes up, click **Add new rule**.



The screenshot shows the 'My Settings & Tools' section of the TeamCity interface. The URL in the browser is `localhost:8080/profile.html?cameFromUrl=%2fprofile.html&notificatorType=email&userId=1`. The top navigation bar includes 'Projects', 'Changes', 'Agents 1', 'Build Queue 0', and a user icon 'Alan'. A sidebar on the right shows a progress bar at 's362'. The main content area is titled 'My Settings & Tools' and has tabs for 'General', 'Groups', and 'Notification Rules', with 'Notification Rules' selected. Below the tabs, it says 'Email Notifier (0/2) | IDE Notifier (0/2) | Jabber Notifier (0/1) | Windows Tray Notifier (0/1)'. A yellow message bar at the top states 'Notification rule has been removed.' A button labeled '+ Add new rule' is visible. Below this, a table lists 'Watching' items and their corresponding 'Send notification when' conditions:

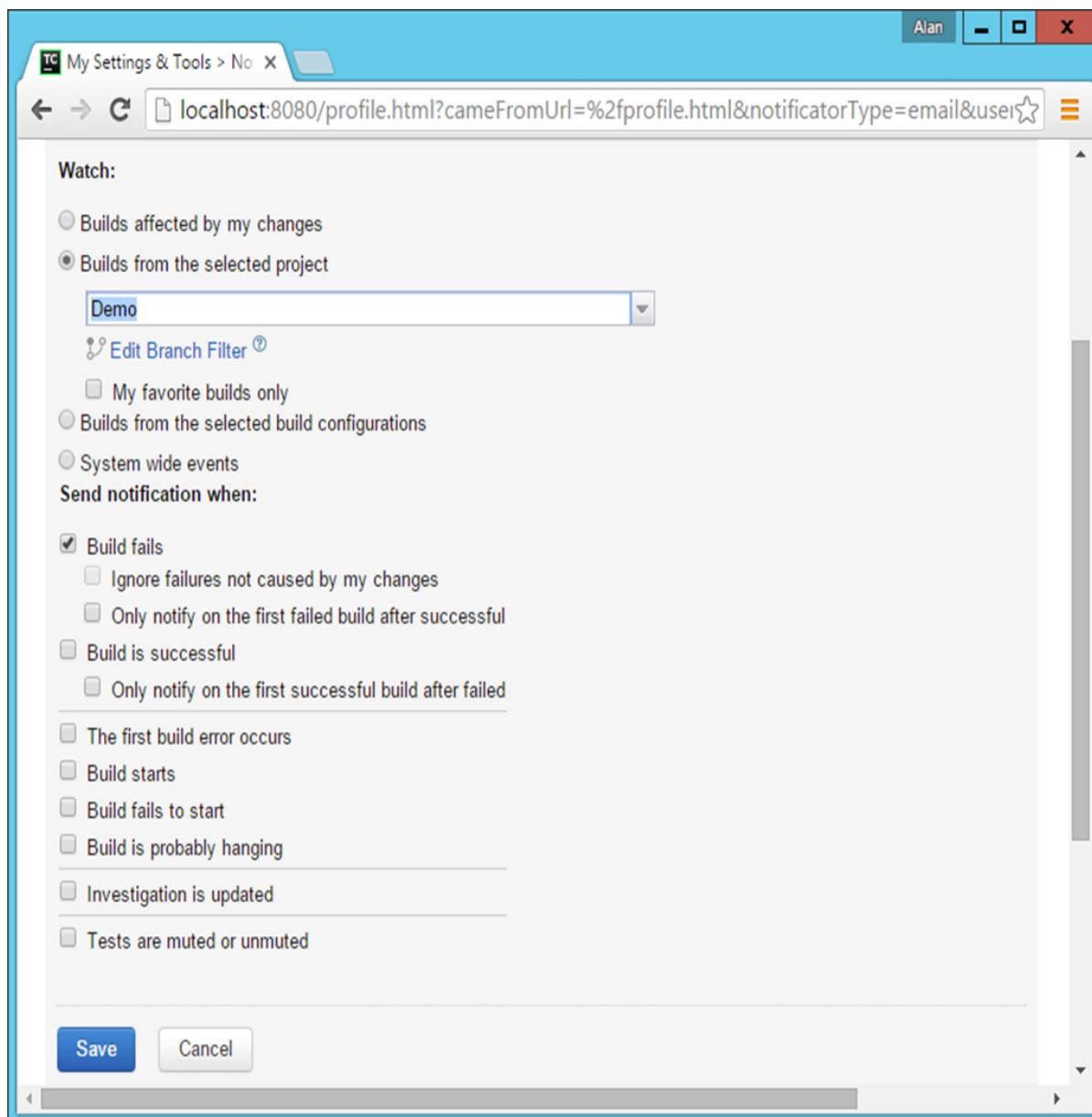
Watching	Send notification when
<i>Builds with my changes</i>	- Build fails (ignore failures not caused by my changes)
<i>System wide events</i>	- Investigation assigned to me

At the bottom, there are links for 'Help' and 'Feedback', and the text 'TeamCity Professional 9.1.6 (build 37459)'.

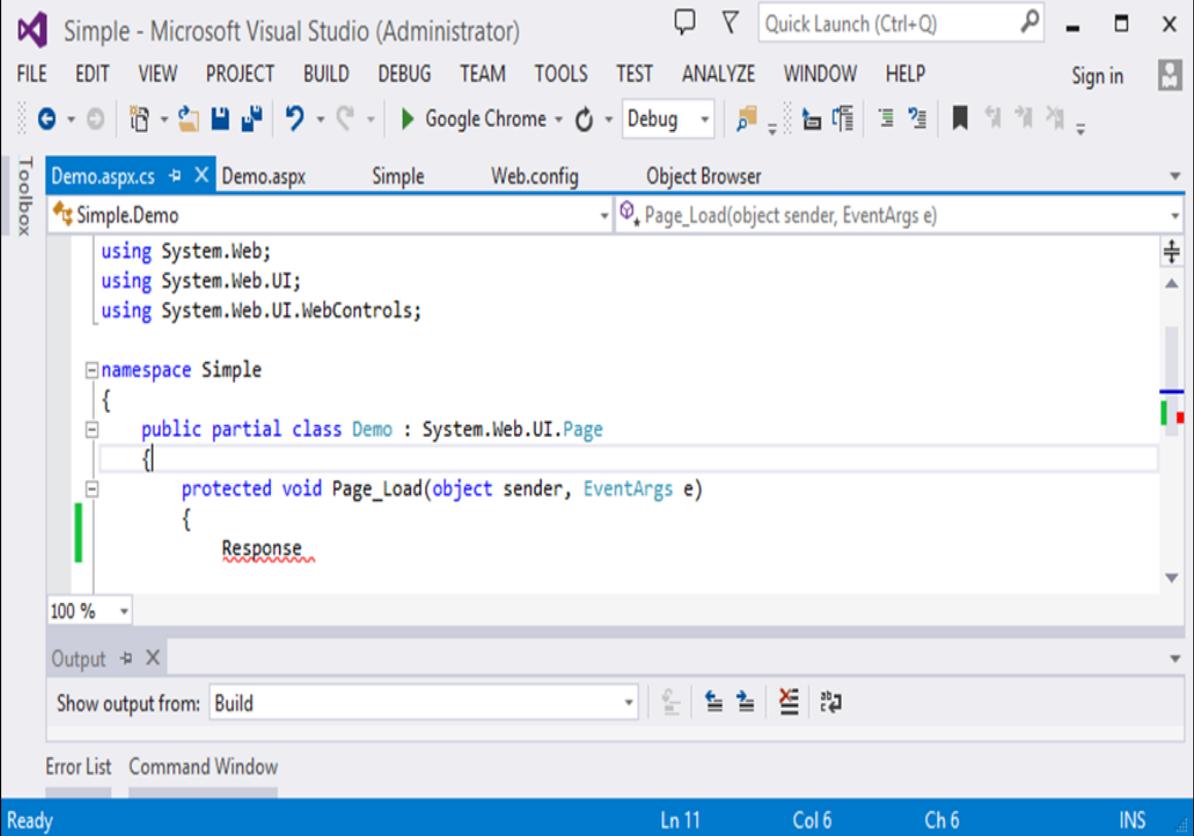
**Step 9:** In Add new rule, choose the following two options and then click Save.

- Builds from select projects – Choose the Demo project.
- Enable the checkbox for 'Build fails'.

By enabling these two options, now whenever a build fails for the Demo project, an email notification will be sent to the user – **demouser**.

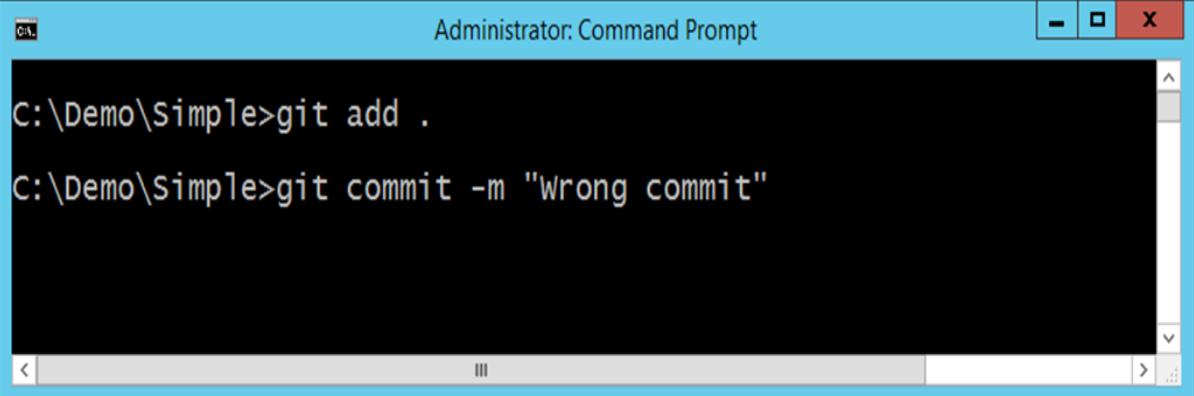


**Step 10:** Now let's trigger a wrong build to see this in action. In Visual Studio, go to the **demo.aspx.cs** file and add a wrong line of code.



The screenshot shows the Microsoft Visual Studio interface with the title bar "Simple - Microsoft Visual Studio (Administrator)". The menu bar includes FILE, EDIT, VIEW, PROJECT, BUILD, DEBUG, TEAM, TOOLS, TEST, ANALYZE, WINDOW, and HELP. The toolbar has various icons for file operations like Open, Save, and Build. The status bar at the bottom shows "Ready", "Ln 11", "Col 6", and "Ch 6". The main window displays the code editor for "Demo.aspx.cs". The code contains a partial class "Demo" with a "Page\_Load" event handler. A red squiggle underlines the word "Response", indicating a syntax error. The output window below the code editor shows "Show output from: Build".

**Step 11:** Now check-in the code from Git by doing a **git add** and **git commit**.



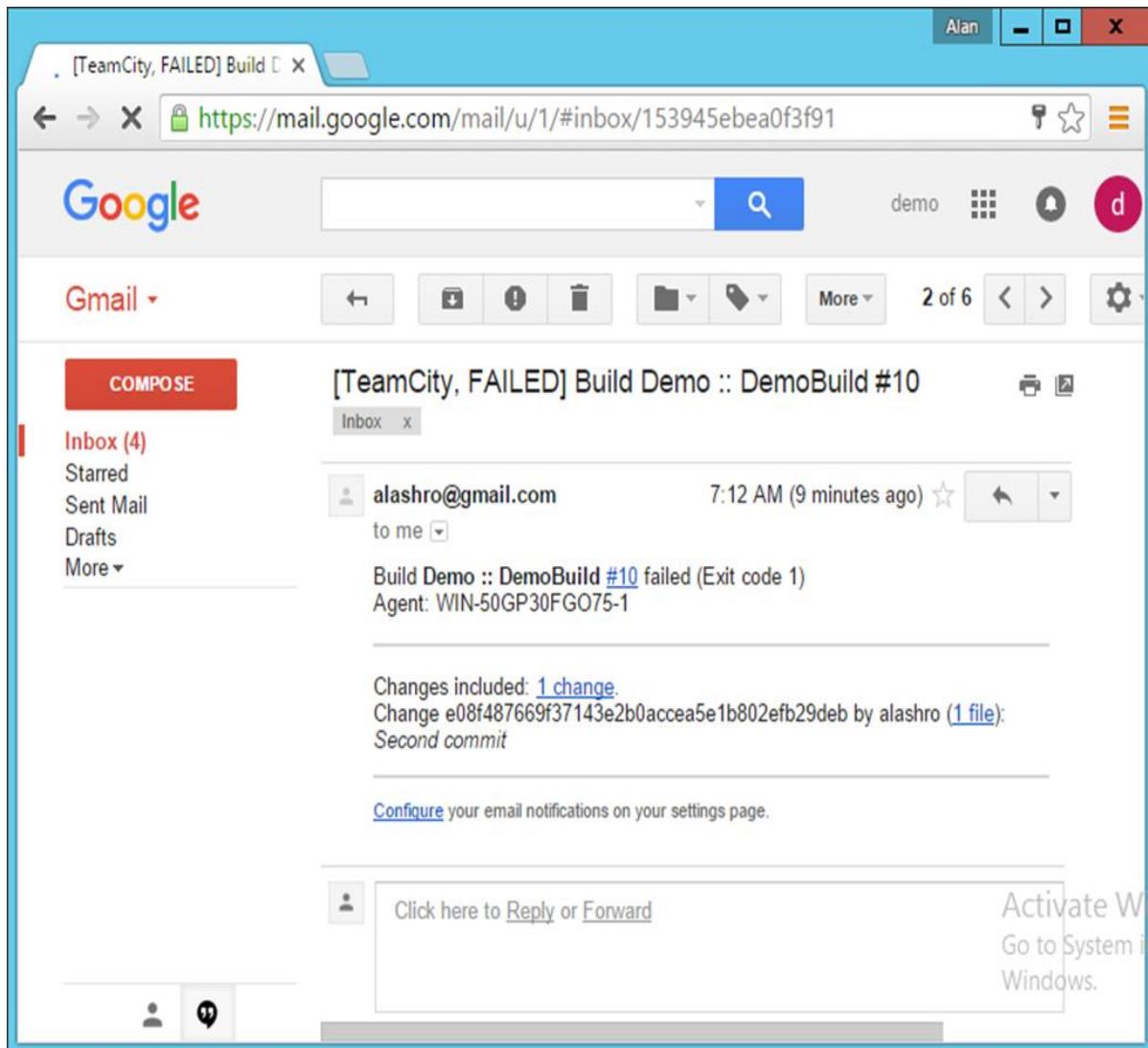
The screenshot shows an "Administrator: Command Prompt" window. The command line shows two commands being run: "C:\Demo\Simple>git add ." and "C:\Demo\Simple>git commit -m "Wrong commit"". The prompt then waits for a response, indicated by the three dots (...).

Now in the Project Dashboard, the build will automatically be triggered and you will see that the build would have failed as shown in the following screenshot.

The screenshot shows the TeamCity Project Dashboard for the 'Demo' project. The 'DemoBuild' configuration has failed with an 'Exit code 1 (new)' error. The build was run #7 moments ago. There are no artifacts. The build was performed by 'alashro (1)'. The URL is 'localhost:8080/overview.html'.

Build #	Status	Artifacts	Performed By	Time Ago
#7	Exit code 1 (new)	No artifacts	alashro (1)	moments ago (7s)

If you login into the Gmail id of the **demouser**, you will actually see a build failure notification in it as shown in the following screenshot.



# 14. CI – Documentation and Feedback

One of the key aspects of Continuous Integration is always to see how the builds are performing, gathering important metrics, documenting those outcomes and generating continuous feedback through continuous builds.

What are the benefits of having these metrics in place?

- **Not Committing Code Enough** – If developers are not committing code to a version control repository frequently, the reason may be a slow integration build. To begin to reduce build duration, perform a high-level analysis of the integration build environment to determine the bottlenecks.

Next, analyze the findings and determine the most appropriate improvement, then attempt to make changes in the build process to reduce the build's duration. Lastly, reevaluate the build duration to determine if further improvements are warranted.

- **Improve Test Performance** – Even in a well-functioning CI system, a bulk of the integration build time will be taken up by the execution of automated tests. Evaluating and improving the performance of these tests can dramatically reduce build duration.
- **Infrastructure Issues** – You may discover that integration builds are slow because of the system infrastructure. Perhaps network performance is slow or there is a slow-performing virtual private network connection.

Geographically dispersed systems and unreliable hardware or software can also induce performance issues. Investigate and improve any infrastructure resources to reduce the build duration.

## Metrics

---

Following are some of the metrics which are available in a Continuous Integration server.

Let's look at what TeamCity has to offer:

One of the simplest form of metrics is what is available in the project dashboard. The key element here is to note the duration of each build. If the duration of each build starts increasing disproportionately to the code being built, then this could be an issue. So, this is one feedback that can be taken and the causes of this could be that the CI server is low on resources and maybe the capacity of the server needs to be increased.

The screenshot shows a web-based Continuous Integration tool interface. At the top, there's a header bar with a logo, user name 'Alan', and standard window controls. Below the header, the URL is 'localhost:8080/viewType.html?buildTypeId=DemoBuild'. The main navigation menu includes 'Projects' (with a dropdown), 'Changes', 'Agents 1', 'Build Queue 0', and a search bar with ID 's362692'. A sub-navigation bar for the 'Demo' project shows 'DemoBuild' selected, with options like 'Run ...'.

The main content area displays the following sections:

- Pending changes:** No pending changes.
- Current status:** Idle.
- Investigation:** A link to 'Start investigation...' of current problems in this build configuration (DemoBuild).
- Recent history:** A table showing build results:

Results	Artifacts	Changes	Started	Duration	Agent	Tags
#11 <span style="color: red;">✖ Exit code 1</span>	None	alashro (1)	20 Mar 16 07:13	7s	WIN-50GP30FGO75-1	None
#10 <span style="color: red;">✖ Exit code 1</span>	None	alashro (1)	20 Mar 16 07:12	7s	WIN-50GP30FGO75-1	None
#9 <span style="color: red;">✖ Exit code 1</span>	None	alashro (1)	20 Mar 16 07:08	7s	WIN-50GP30FGO75-1	None
#8 <span style="color: red;">✖ Exit code 1</span>	None	alashro (1)	20 Mar 16 06:49	8s	WIN-50GP30FGO75-1	None
#7 <span style="color: red;">✖ Exit code 1 (new)</span>	None	alashro (1)	20 Mar 16 06:20	7s	WIN-50GP30FGO75-1	None
#6 <span style="color: green;">✔ Success</span>	None	alashro (1)	20 Mar 16 05:54	13s	WIN-50GP30FGO75-1	None
#5 <span style="color: green;">✔ Success</span>	None	alashro (1)	20 Mar 16 05:53	9s	WIN-50GP30FGO75-1	None

TeamCity has the facility to see if the CI server is in fact having any sort of issues with regards to infrastructure. In the **admin dashboard** in TeamCity, one can click on **Disk Usage** to see how much disk space is being consumed by each build.

The screenshot shows the TeamCity Admin Dashboard with the title "Disk Usage — TeamCity". The URL in the address bar is "localhost:8080/admin/admin.html?item=diskUsage". The navigation bar includes "Projects", "Changes", "Agents 1", "Build Queue 0", "admin", and "Administration". The "Administration" section is selected. The main content area is titled "Administration > Disk Usage". It displays project-related settings: "Total free disk space: 124.75 GB" and "Last full scan was done on 18 Mar 16 10:46". A sidebar lists "Project-related Settings" like "Projects", "Build Time", "Disk Usage" (which is selected), "Server Health", "Audit", "User Management", "Integrations", and "Server Administration". The "Disk Usage" section includes filters for "Group by project" (checked) and "Show archived projects" (unchecked). A table shows disk usage details:

Project/Configuration	Size	%	Artifacts
Total:	<1 MB		<1 MB
D Demo	<1 MB	100%	<1 MB

Buttons for "Rescan now" and a scroll bar are also visible.

If any more details are required, then TeamCity has the **diagnostics button**, which can give more information on the **CPU and Memory** being utilized by the CI Server.

The screenshot shows the TeamCity Administration interface with the 'Diagnostics' section selected. The left sidebar lists various administration categories, and the main content area displays memory usage statistics and a graph.

**Project-related Settings:**

- Projects
- Build Time
- Disk Usage
- Server Health
- Audit

**User Management:**

- Users
- Groups

**Integrations:**

- NuGet
- Tools

**Server Administration:**

- Global Settings
- Authentication
- Email Notifier
- Jabber Notifier
- Agent Cloud
- Diagnostics** (selected)
- Backup
- Projects Import

**Troubleshooting:**

- Browse Data Directory
- Search

**Debug Logging:**

Active logging preset: <Default>

**Hangs and Thread Dumps:**

If the TeamCity server appears slow or is not responding, please try taking several thread dumps with some interval. On this page you can either view a server thread dump in a new browser window or save a thread dump.

**Memory and CPU usage:**

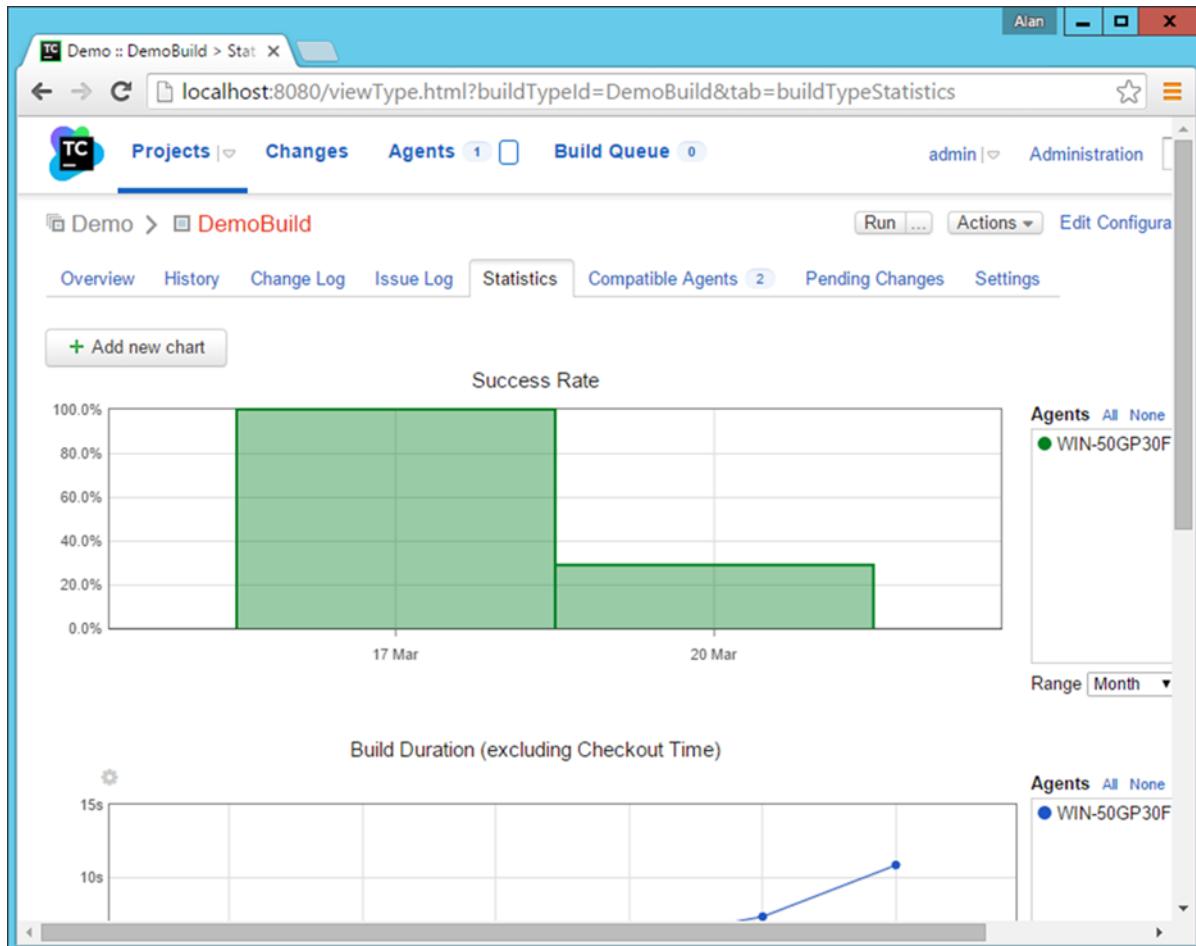
Total heap: 267.91 MB (59% of maximum available 455.25 MB)  
Data: 165.15 MB (48% of maximum available 341.5 MB)

**Memory usage graph:**

The graph shows memory usage over time, with two data series: Total heap (blue line) and Data (red line). The Y-axis represents percentage from 0.00% to 100.00%. The X-axis represents time. The Total heap usage fluctuates between 50% and 60%, while the Data usage stays relatively flat around 45%.

## Detailed View of Build Metrics

If one wants to see a detailed view of the builds of a particular project over time, then this is available as a part of the project builds. In the Project build screen, go to the Statistics screen, this will provide various statistics and charts on how the build is performing.



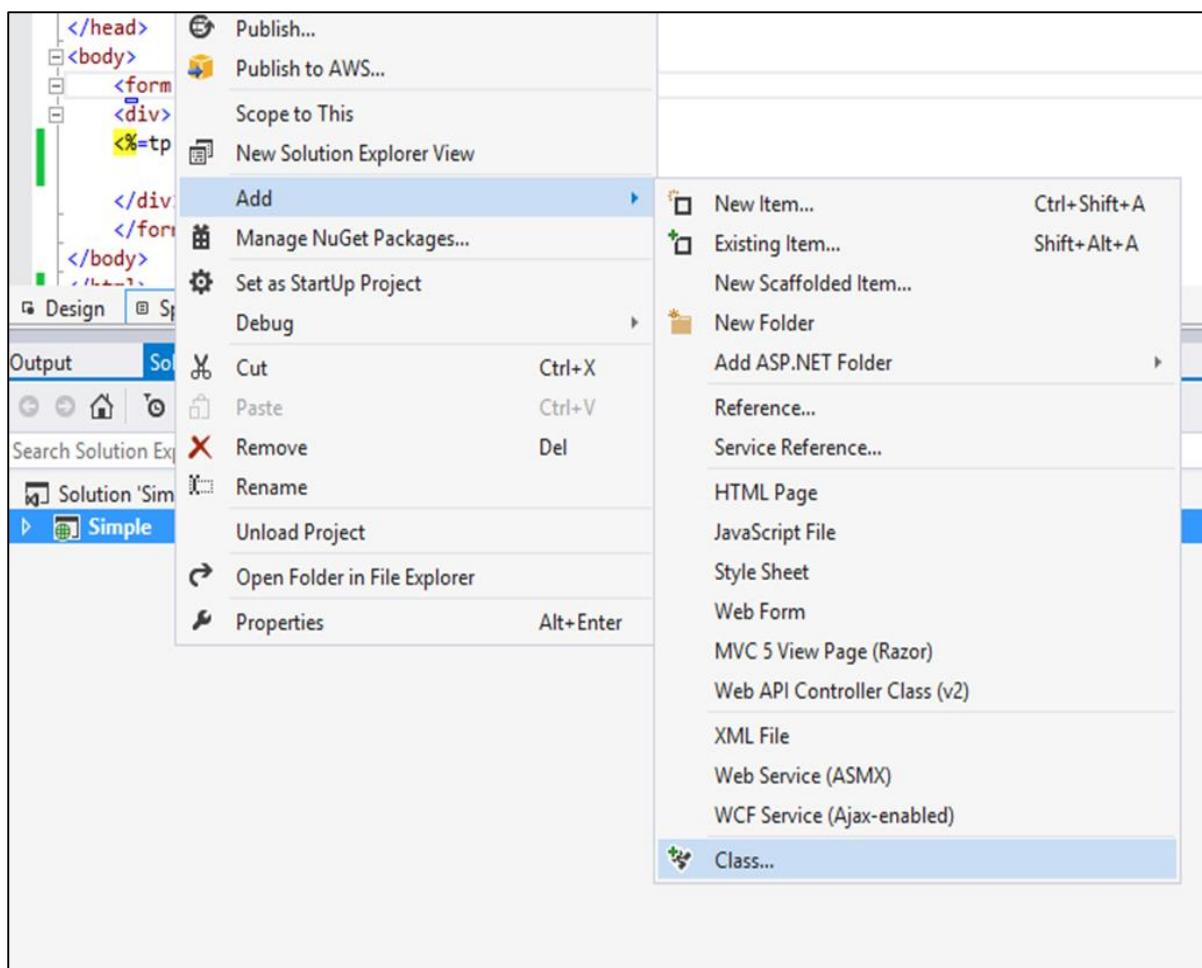
# 15. CI – Continuous Testing

One of the key features of Continuous Integration is to ensure that the **on-going testing** holds all the code which gets built by the CI server. After a build is carried out by the CI Server, it has to be ensured that the test cases are in place to get the required code tested. Every CI server has the ability to run unit test cases as part of the **CI suite**. In **.Net**, the unit testing is a feature which is inbuilt into the **.Net framework** and the same thing can be incorporated into the CI Server as well.

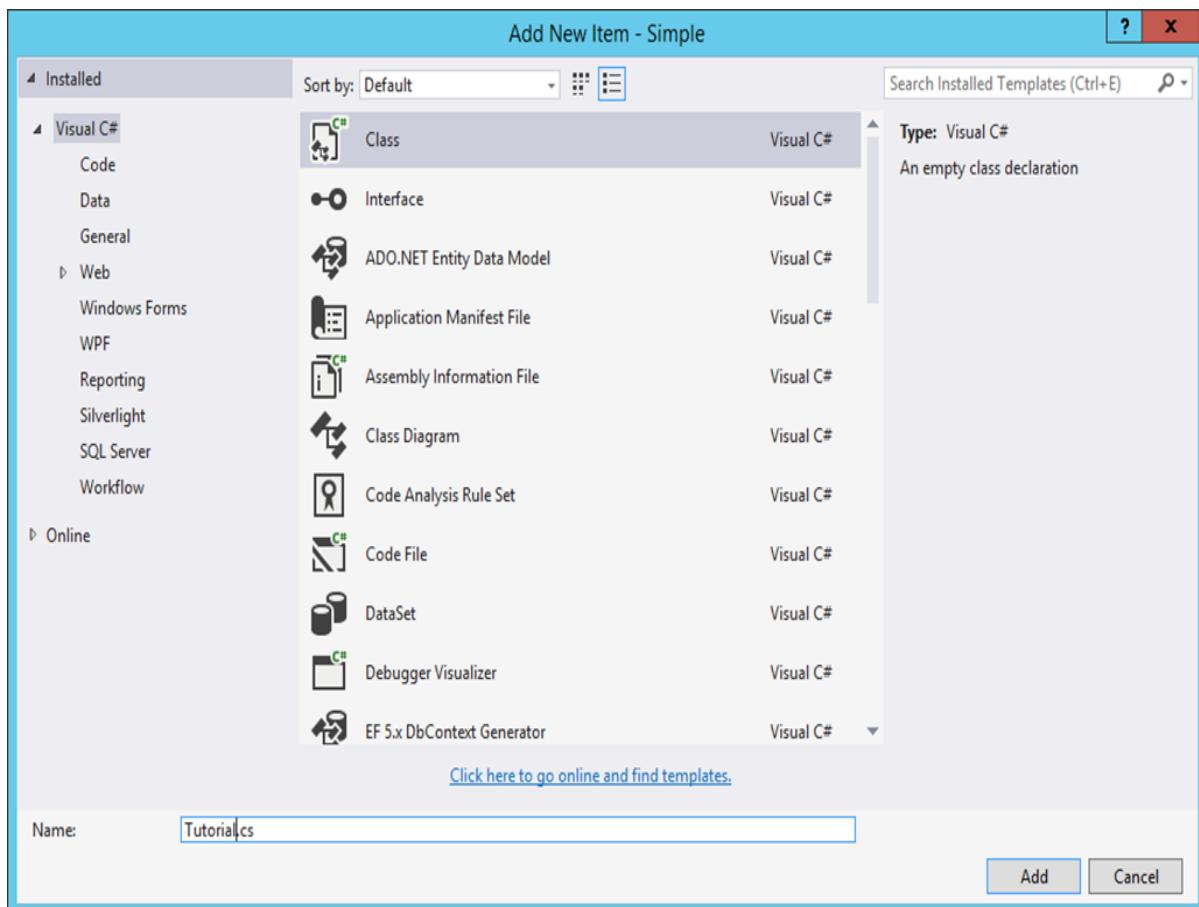
This chapter will see how we can define a test case in **.Net** and then let our TeamCity server run this test case after the build is completed. For this, we first need to ensure that we have a unit test defined for our sample project.

To do this, we must follow the ensuing steps with utmost carefulness.

**Step 1:** Let's add a new class to our solution, which will be used in our Unit Test. This class will have a name variable, which will hold the string "Continuous Integration". This string will be displayed on the web page. Right-click on the Simple Project and choose the menu option **Add -> Class**.



**Step 2:** Give a name for the class as **Tutorial.cs** and click the Add button at the bottom of the screen.



**Step 3:** Open the Tutorial.cs file and add the following code in it. This code just creates a string called **Name**, and in the Constructor assign the name to a string value as **Continuous Integration**.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace Simple
{
    public class Tutorial
    {
        public String Name;
        public Tutorial()
        {
            Name = "Continuous Integration";
        }
    }
}

```

```

    }
}

}

```

**Step 4:** Let us make the change to our **Demo.aspx.cs** file to use this new class. Update the code in this file with the following code. So this code will now create a new instance of the class created above.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace Simple
{

    public partial class Demo : System.Web.UI.Page
    {

        Tutorial tp = new Tutorial();

        protected void Page_Load(object sender, EventArgs e)
        {

            tp.Name = "Continuous Integration";
        }
    }
}

```

**Step 5:** In our **demo.aspx** file, let us now reference the **tp.Name** variable, which was created in the **aspx.cs** file.

```

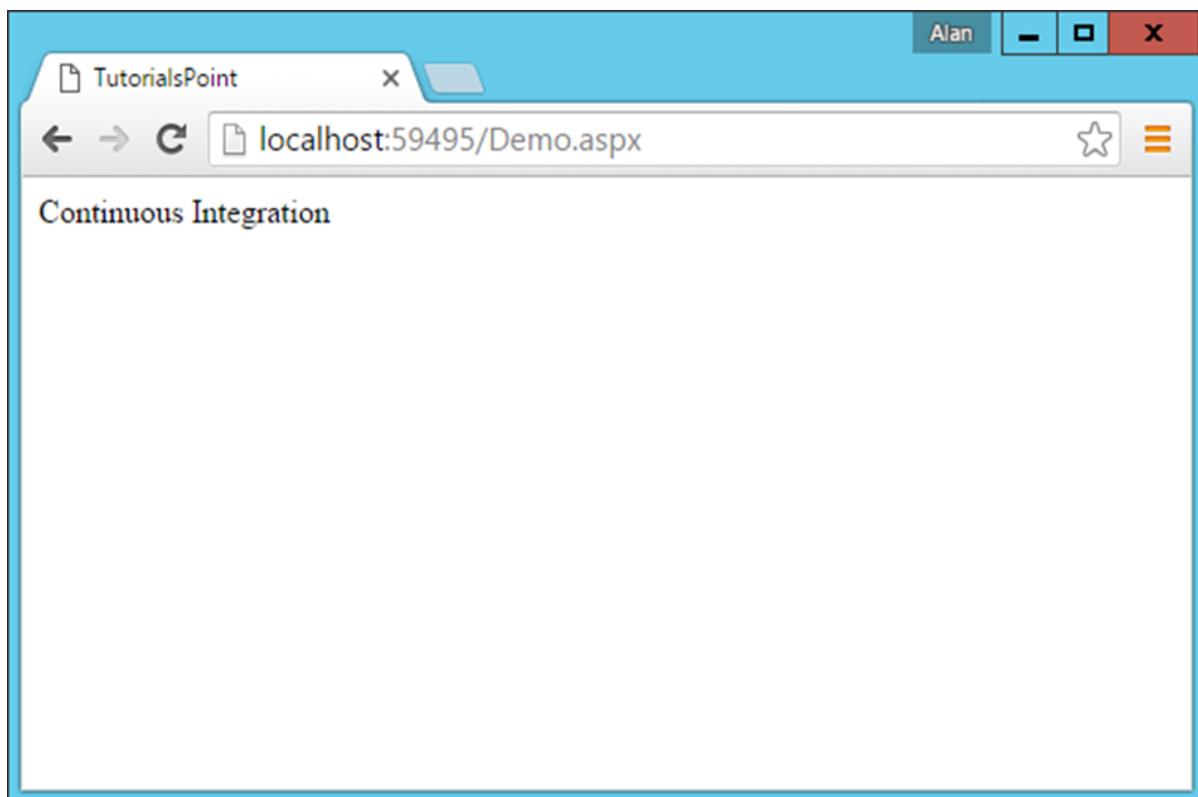
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Demo.aspx.cs"
Inherits="Simple.Demo" %>

<!DOCTYPE html>

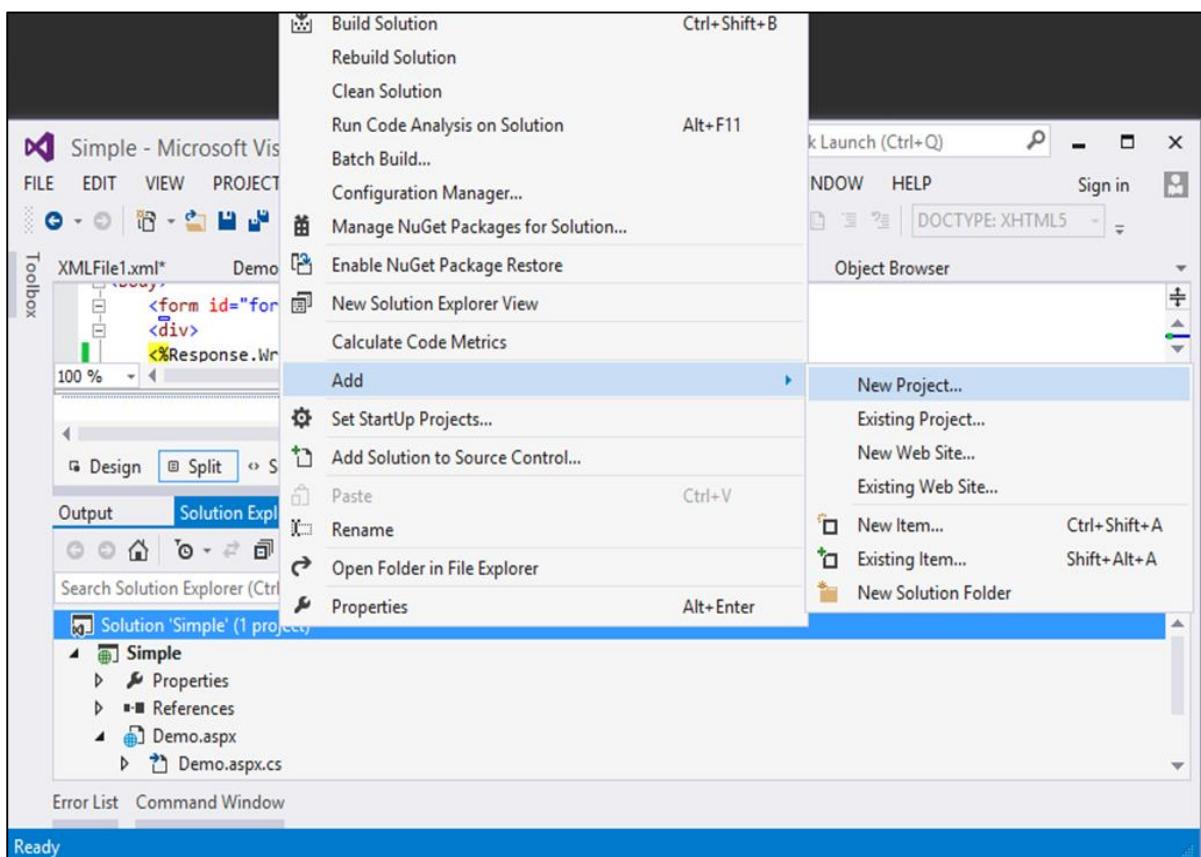
```

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>TutorialsPoint1</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <%=tp.Name%>
        </div>
    </form>
</body>
</html>
```

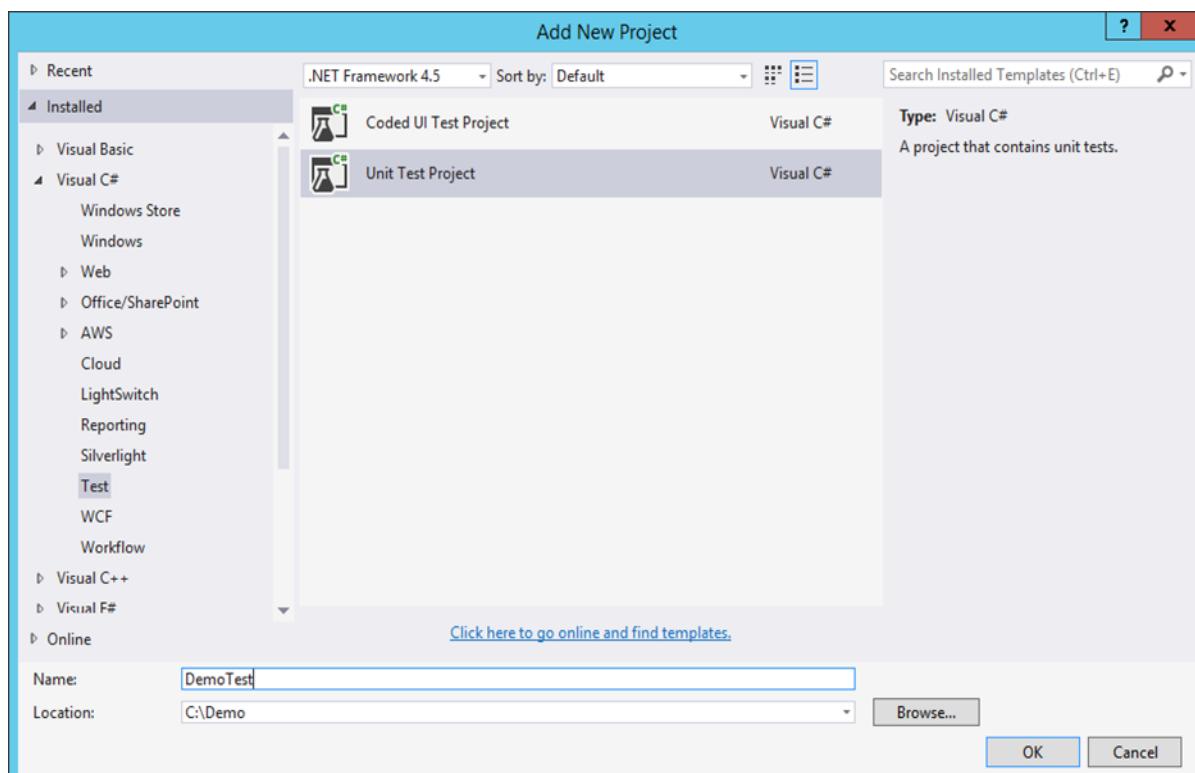
Just to ensure our code works fine with these changes, you can run the code in Visual Studio. You should get the following output once the compilation is complete.



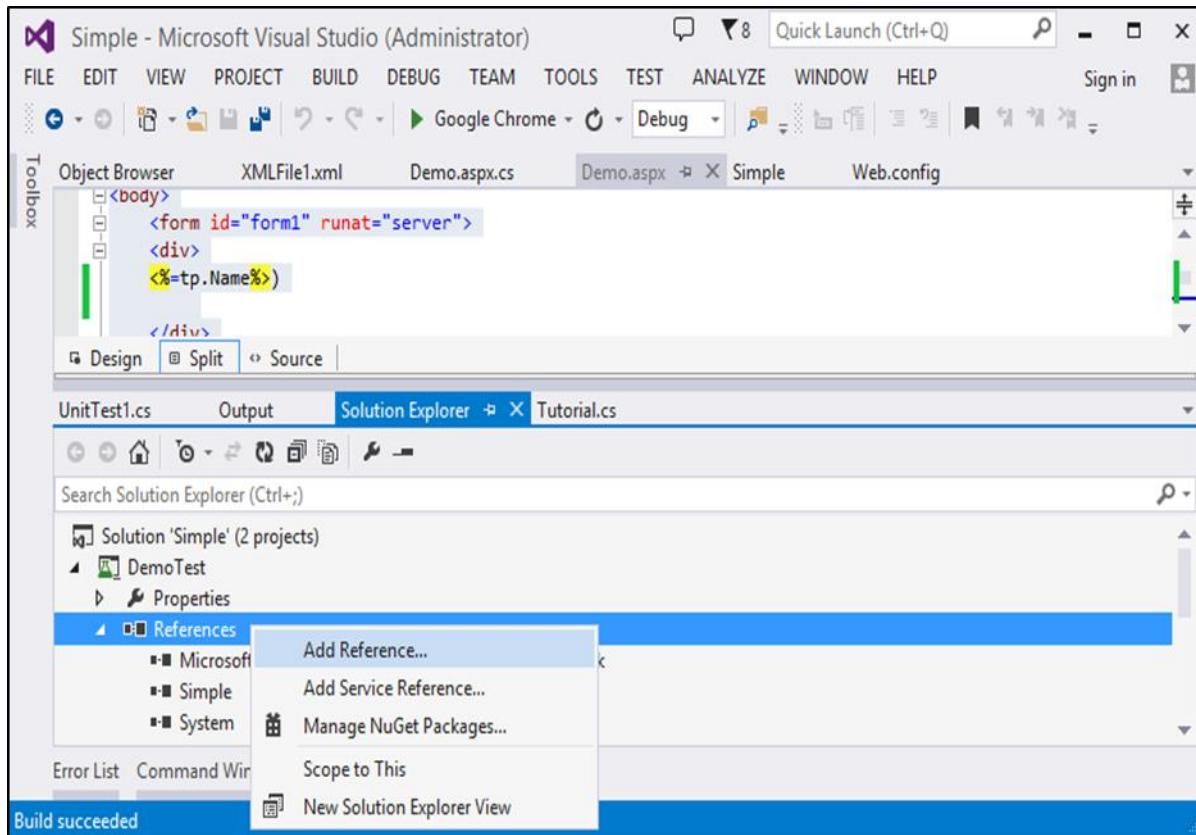
**Step 6:** Now it is time to add our Unit tests to the project. Right-click on **Solution** and choose the menu option **Add -> New Project**.



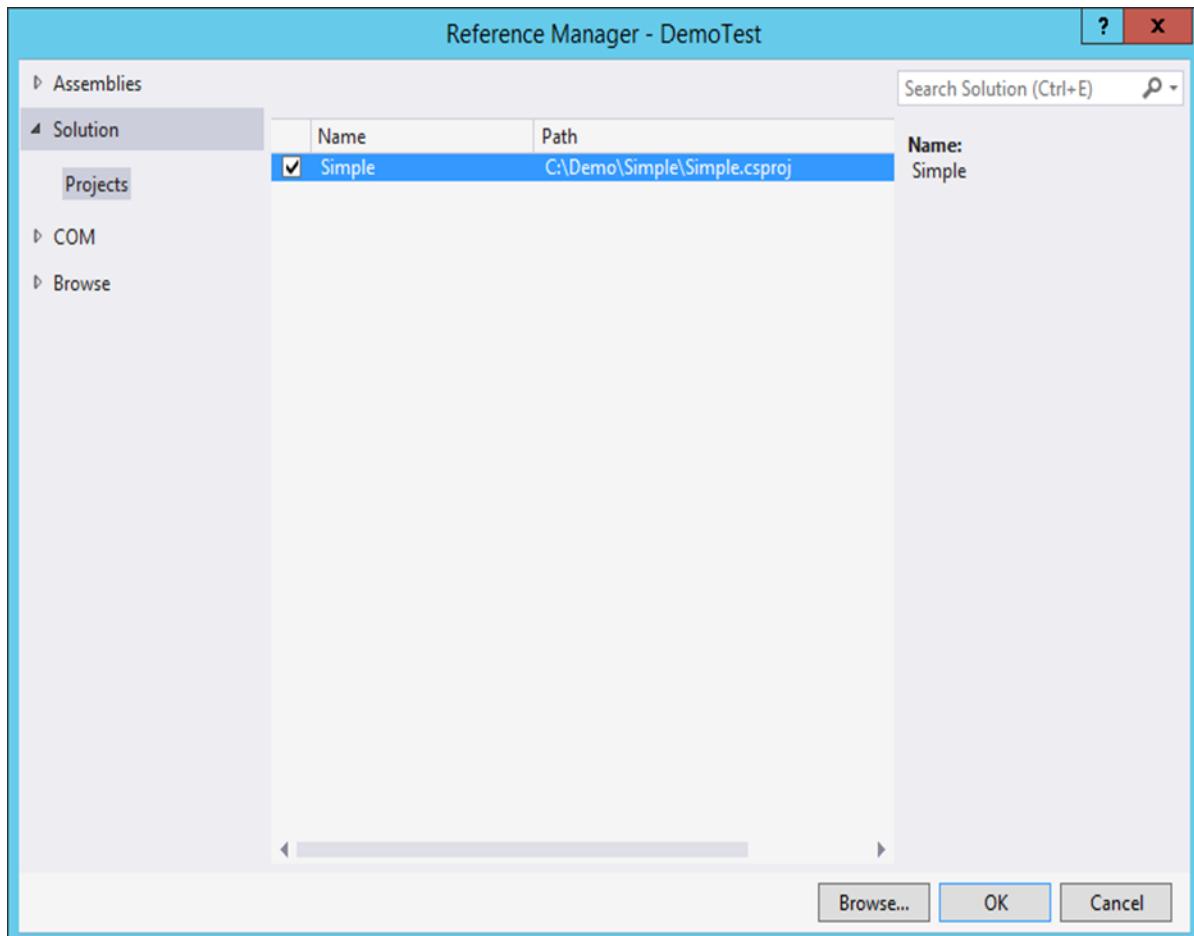
**Step 7:** Navigate to **Test** and on the right hand side, choose **Unit Test Project**. Give a name as **DemoTest** and then click **OK**.



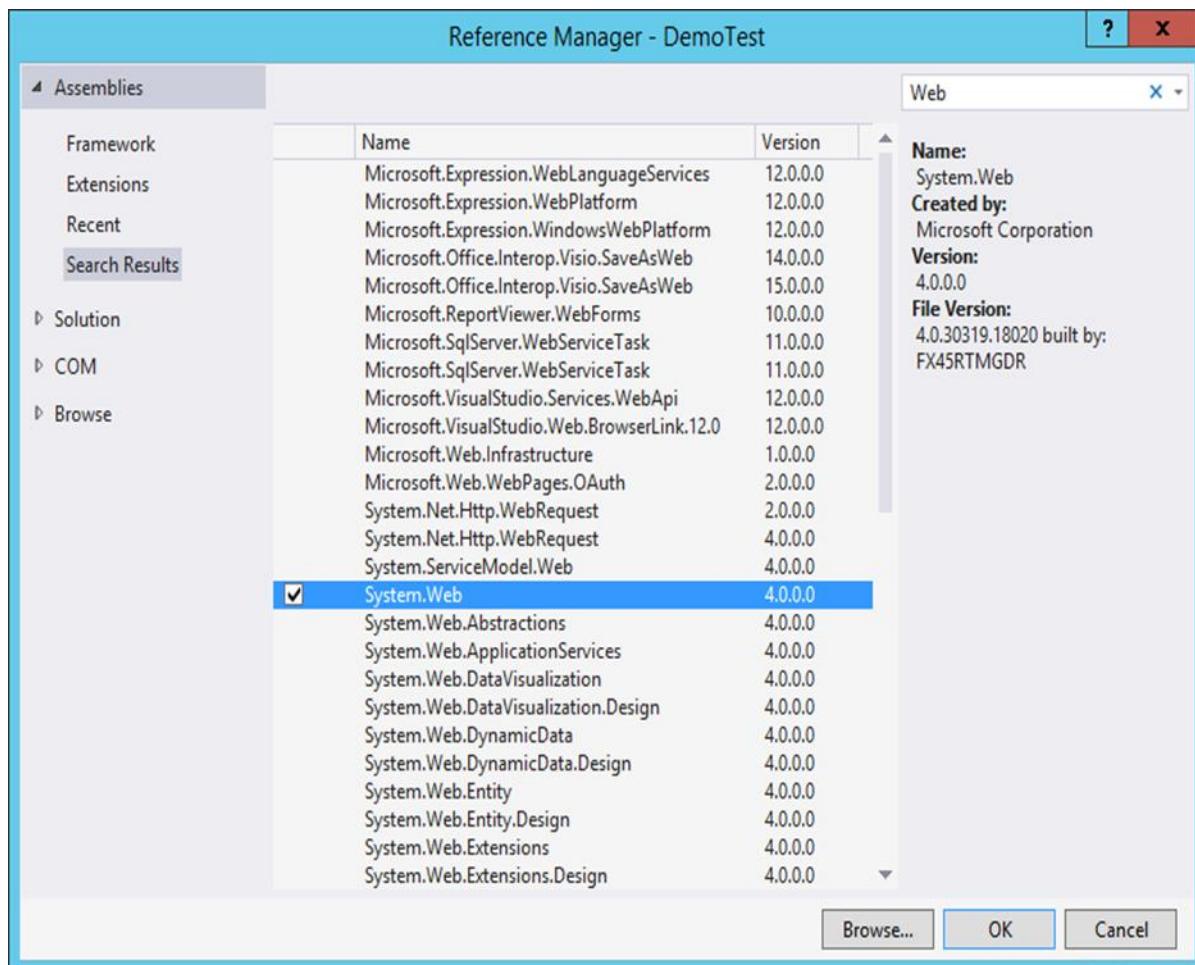
**Step 8:** In your **Demo Test project**, you need to add a reference to the Simple project and to the necessary **testing assemblies**. Right-click on the project and choose the menu option **Add Reference**.



**Step 9:** In the next screen that comes up, go to Projects, choose **Simple Reference** and click OK.



**Step 10:** Click **Add Reference** again, go to Assemblies and type **Web** in the Search box. Then add a reference of **System.Web**.



**Step 11:** In the **Unit Test file**, add the following code. This code will ensure that the Tutorial class has a string name variable. It will also assert the fact that the Name should equal a value of "Continuous Integration". This will be our simple Test case.

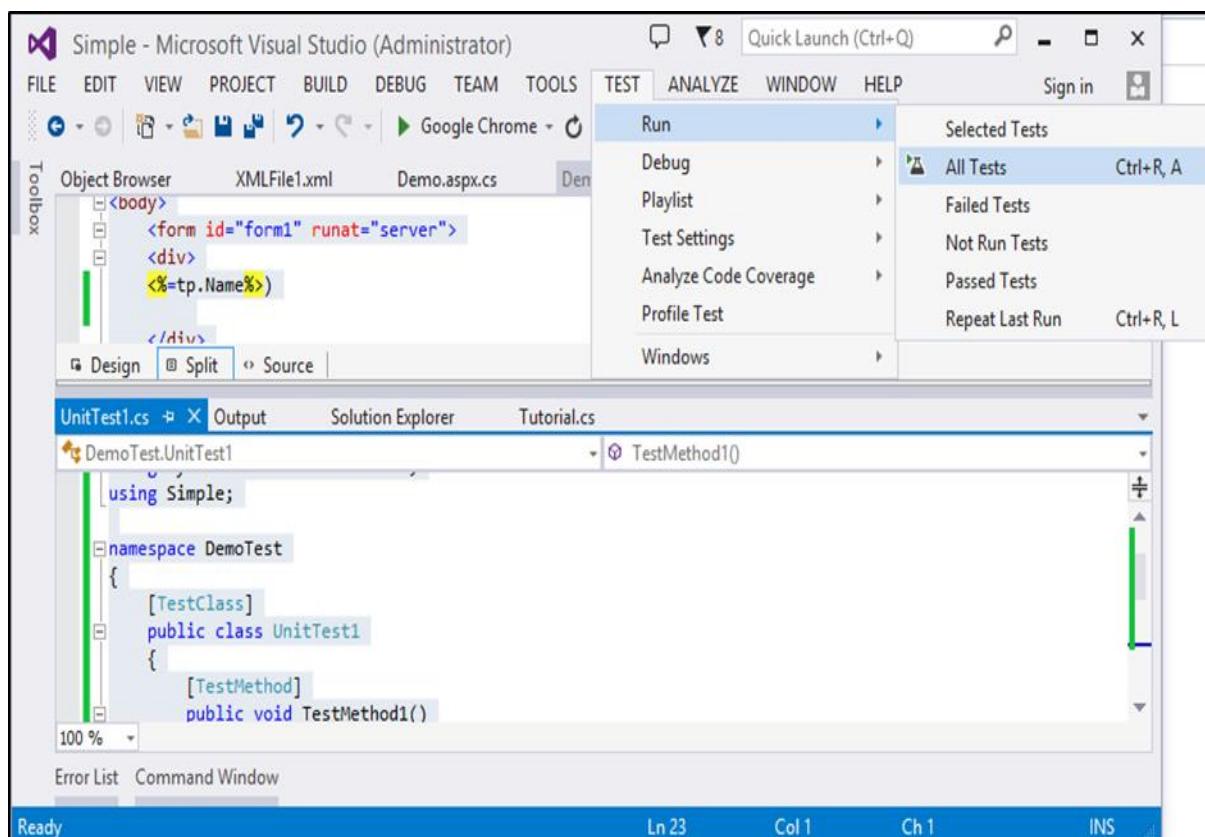
```
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using Microsoft.VisualStudio.TestTools.UnitTesting.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using Simple;

namespace DemoTest
{
    [TestClass]
    public class UnitTest1
    {
```

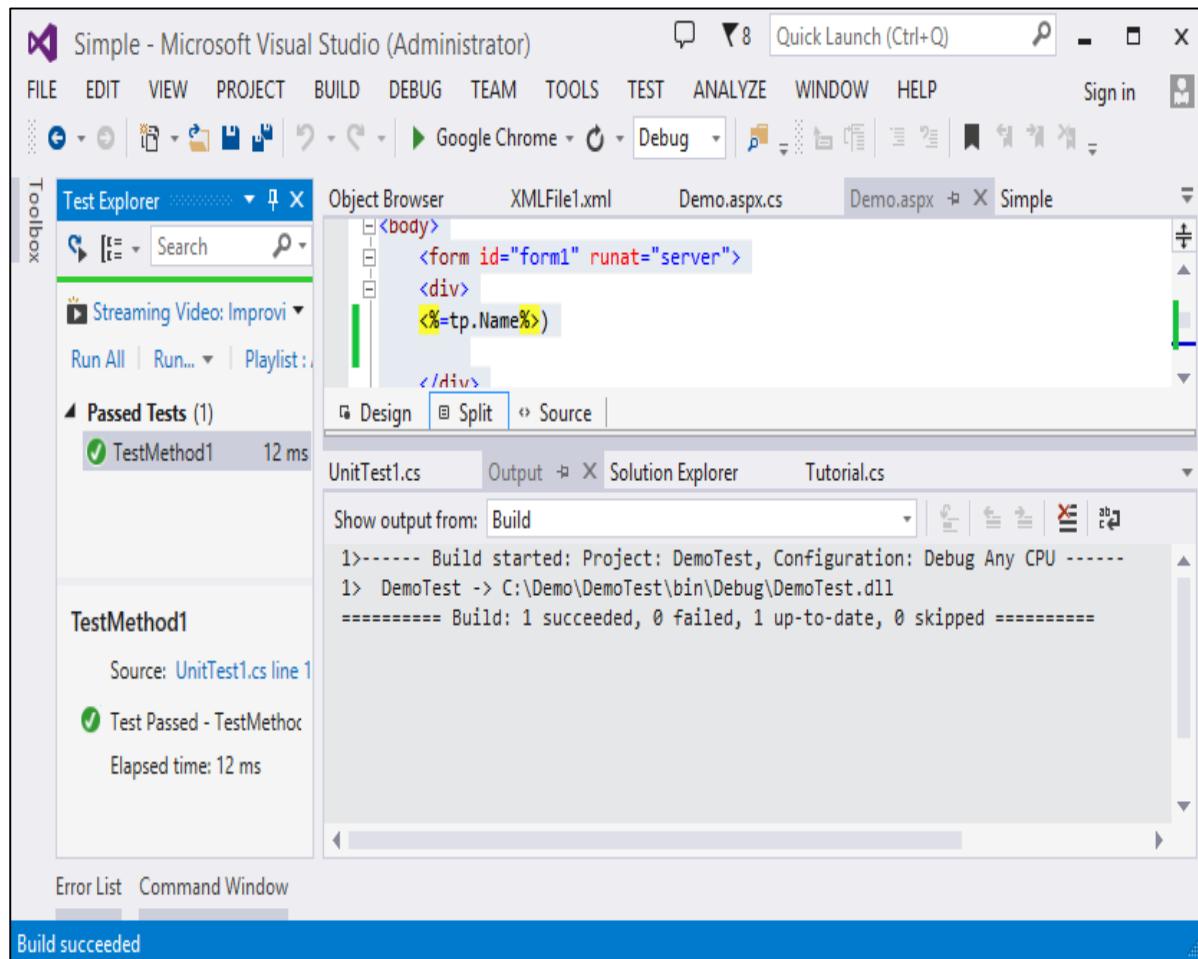
```
[TestMethod]
public void TestMethod1()
{
    Tutorial tp = new Tutorial();
    Assert.AreEqual(tp.Name, "Continuous Integration");

}
}
```

**Step 12:** Now let's run our test in Visual Studio to make sure it works. In Visual Studio, choose the menu option **Test -> Run -> All Tests**.

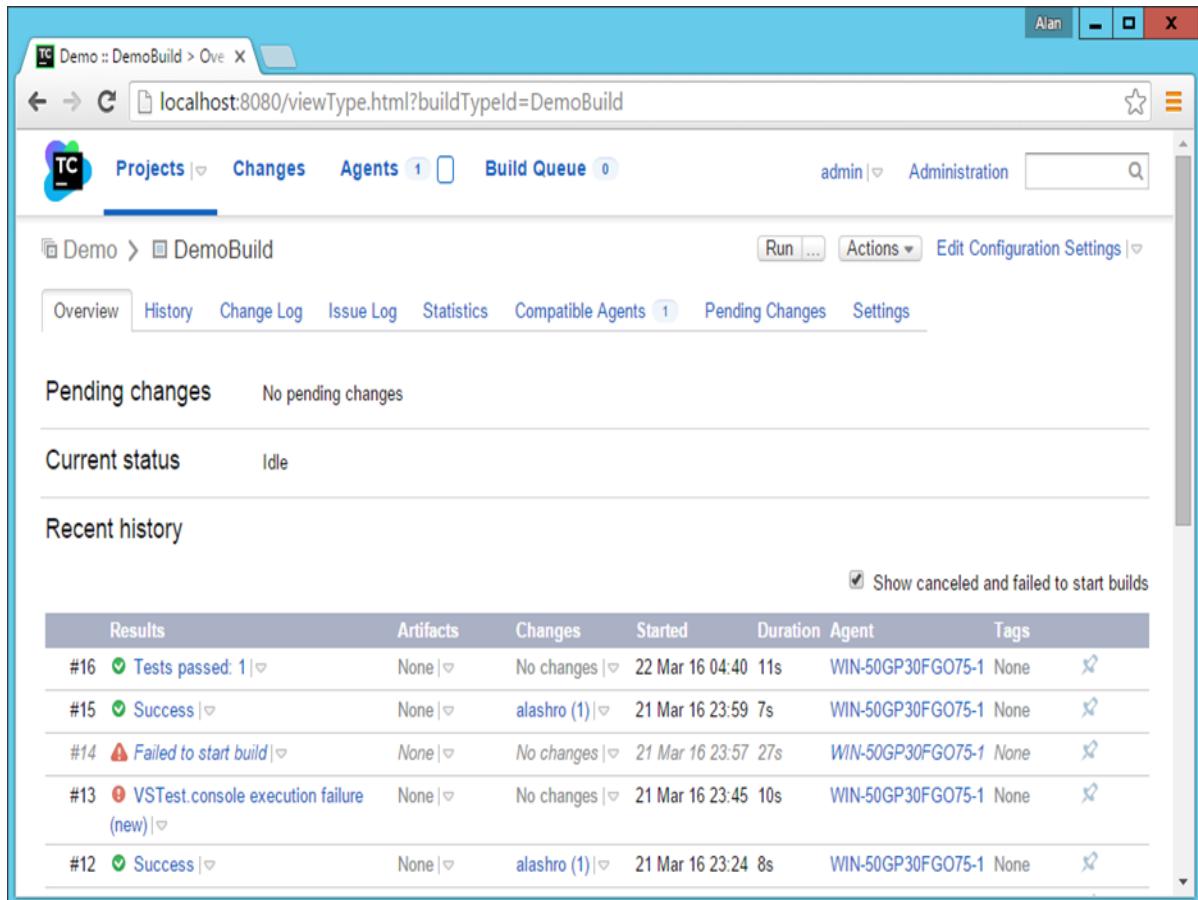


After running the test, you will see the Test successfully run on the left hand side of Visual Studio.



Enabling Continuous Testing within TeamCity – Now that all the test cases are in place, it is time to integrate these into our Team City server.

**Step 13:** For this, we need to create a build step in our Project configuration. Go to your project home and click Edit Configuration Settings.



The screenshot shows the TeamCity web interface for the 'DemoBuild' project. The top navigation bar includes 'Projects', 'Changes', 'Agents', 'Build Queue', 'admin', 'Administration', and a search bar. Below the navigation is a breadcrumb trail: Demo > DemoBuild. On the right side of the header are 'Run ...', 'Actions', and 'Edit Configuration Settings' buttons. The main content area has tabs for 'Overview', 'History', 'Change Log', 'Issue Log', 'Statistics', 'Compatible Agents', 'Pending Changes', and 'Settings'. The 'History' tab is selected. It displays a table of build results:

Results	Artifacts	Changes	Started	Duration	Agent	Tags
#16 Tests passed: 1   <a href="#">▼</a>	None   <a href="#">▼</a>	No changes   <a href="#">▼</a>	22 Mar 16 04:40	11s	WIN-50GP30FG075-1	None
#15 Success   <a href="#">▼</a>	None   <a href="#">▼</a>	alashro (1)   <a href="#">▼</a>	21 Mar 16 23:59	7s	WIN-50GP30FG075-1	None
#14 Failed to start build   <a href="#">▼</a>	None   <a href="#">▼</a>	No changes   <a href="#">▼</a>	21 Mar 16 23:57	27s	WIN-50GP30FG075-1	None
#13 VSTest.console execution failure (new)   <a href="#">▼</a>	None   <a href="#">▼</a>	No changes   <a href="#">▼</a>	21 Mar 16 23:45	10s	WIN-50GP30FG075-1	None
#12 Success   <a href="#">▼</a>	None   <a href="#">▼</a>	alashro (1)   <a href="#">▼</a>	21 Mar 16 23:24	8s	WIN-50GP30FG075-1	None

A checked checkbox at the top right of the table says 'Show canceled and failed to start builds'.

**Step 14:** Then go to Build Step -> MS Build and click Add build step as depicted in the following screenshot.

Build Step	Parameters Description
Build	MSBuild Build file: Simple\Simple.csproj Targets: default Execute: If all previous steps finished successfully

In the next screen that comes up, add the following values –

- Choose the runner type as Visual Studio Tests.
- Enter an optional Test step name.
- Choose the Test Engine type as **VSTest**.
- Choose the Test Engine version as **VSTest2013**.
- In the Test files name, provide the location as **DemoTest\bin\Debug\DemoTest.dll** – Remember that **DemoTest** is the name of our project which contains our Unit Tests. The **DemoTest .dll** will be generated by our first build step.
- Click Save which will be available at the end of the screen.

The screenshot shows a web-based interface for managing build configurations. The URL in the browser is `localhost:8080/admin/editRunType.html?id=buildType:DemoBuild&runnerId=_NEW_RUNNER_`. The page title is "DemoBuild Configuration". The navigation path is "Administration > <Root project> > Demo > DemoBuild". The main content area is titled "New Build Step" under "Build Step: MSBuild".  
  
On the left, there is a sidebar with the following options:

- Build Configuration Settings
- General Settings
- Version Control Settings (1)
- Build Step: MSBuild** (selected)
- Triggers (1)
- Failure Conditions
- Build Features
- Dependencies
- Parameters
- Agent Requirements

A note at the bottom of the sidebar says "Last edited one minute ago by admin (view history)".  
  
The main form fields are:

- Runner type:** Visual Studio Tests (Visual Studio Tests runner)
- Step name:** TestStep (Optional, specify to distinguish this build step from other steps.)
- Test engine type:** VSTest
- Test engine version:** VSTest 2013
- Test file names:** \*Edit included assemblies:  
DemoTest\bin\Debug\DemoTest.dll

Below the assembly list, there is a note: "Newline-separated list of assemblies to be included in test run. Wildcards are supported. Paths to the assemblies must be relative to the build checkout directory." There is also a link "Edit excluded assemblies".  
  
At the bottom, there is a section for ".NET Coverage":

- .NET Coverage tool:** <No .NET Coverage> (Choose a .NET coverage tool.)
- Show advanced options** (link)

Now you will have 2 build steps for your project. The first is the Build step which will build your application code and your test project. And the next will be used to run your test cases.

The screenshot shows the Jenkins administration interface for a build configuration named 'DemoBuild'. The left sidebar has a tree view with 'Build Configuration Settings' expanded, showing 'General Settings', 'Version Control Settings', 'Build Steps' (which is selected), 'Triggers', 'Failure Conditions', 'Build Features', 'Dependencies', 'Parameters', and 'Agent Requirements'. A note at the bottom of this sidebar says 'Last edited moments ago by admin (view history)'. The main content area has a heading 'Build Steps' with a sub-instruction: 'In this section you can configure the sequence of build steps to be executed. Each build step is represented by a build runner and provides integration with a specific build or test tool.' Below this are three buttons: '+ Add build step', 'Reorder build steps', and 'Auto-detect build steps'. A table lists the build steps:

Build Step	Parameters Description	Edit	More
Build	MSBuild Build file: Simple\Simple.csproj Targets: default Execute: If all previous steps finished successfully	Edit	More
TestStep	Visual Studio Tests Test engine: VSTest Included assemblies: DemoTest\bin\Debug\DemoTest.dll Execute: If all previous steps finished successfully	Edit	More

**Step 15:** Now it is time to check-in all your code in Git, so that the entire build process can be triggered. The only difference is this time, you need to run the **git add** and **git commit** command from the **Demo parent folder** as shown in the following screenshot.

```
C:\Demo>git add .
C:\Demo>git commit -m "Test Commit"
```

Now when the build is triggered, you will see an initial output which will say that the test passed.

The screenshot shows the TeamCity web interface. At the top, there's a navigation bar with 'Projects' (selected), 'Changes', 'Agents 1', 'Build Queue 0', and user 'admin'. Below the navigation is a search bar and a 'Hide Successful Configurations' checkbox. The main area displays a project named 'Demo' with a build named 'DemoBuild #16'. The build status is 'Tests passed: 1' (green checkmark). There are buttons for 'Run ...' and 'no hidden | x'. At the bottom, there are links for 'Help', 'Feedback', 'TeamCity Professional 9.1.6 (build 37459)', and 'License agreement'.

**Step 16:** If you click on the Test passed result and go to the Test tab, you will now see that the UnitTest1 was executed and that it is passed.

This screenshot shows the 'Tests' tab for build #16 of the 'DemoBuild' configuration. The tab title is 'Demo > DemoBuild > #16 (22 Mar 16 04:40)'. It displays a table of test results:

Status	Test	Duration	Order#
OK	UnitTest1.TestMethod1   (VSTest.DemoTest)	6ms	1

Below the table, there are filters for 'View: tests', 'containing: [ ]', 'with: any', and a 'Filter' button. At the bottom, there are links for 'Help', 'Feedback', 'TeamCity Professional 9.1.6 (build 37459)', and 'License agreement'.

# 16. CI – Continuous Inspection

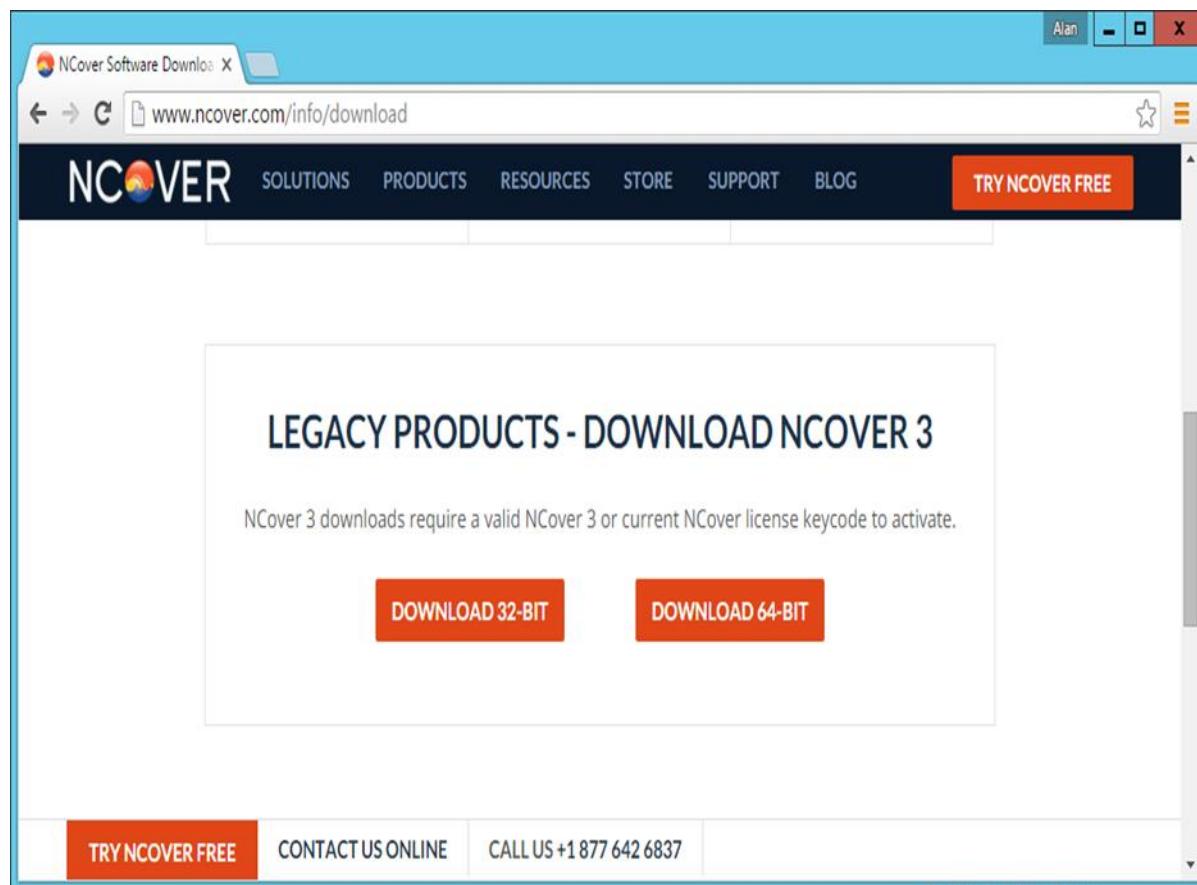
Continuous Inspection is the process of an automated code review or inspection conducted for your code before the actual tests are run. There are subtle differences between inspecting and testing software. Testing is dynamic and executes the software in order to test the functionality. Inspection analyzes the code based on a set of predefined rules.

Inspectors (or static and dynamic analysis tools) are directed by identified standards that teams should adhere to (usually coding or design metrics). Examples of inspection targets include coding “grammar” standards, architectural layering adherence, code duplication, and many others.

Continuous Inspection reduces the time between a discovery and a fix. There are a number of Continuous Inspection tools available. For this example, we are going to be using **NCover 3.x** which has an integration with TeamCity. Let’s see how we can carry out Continuous Inspection and what it can do for us.

## Download and Install NCover

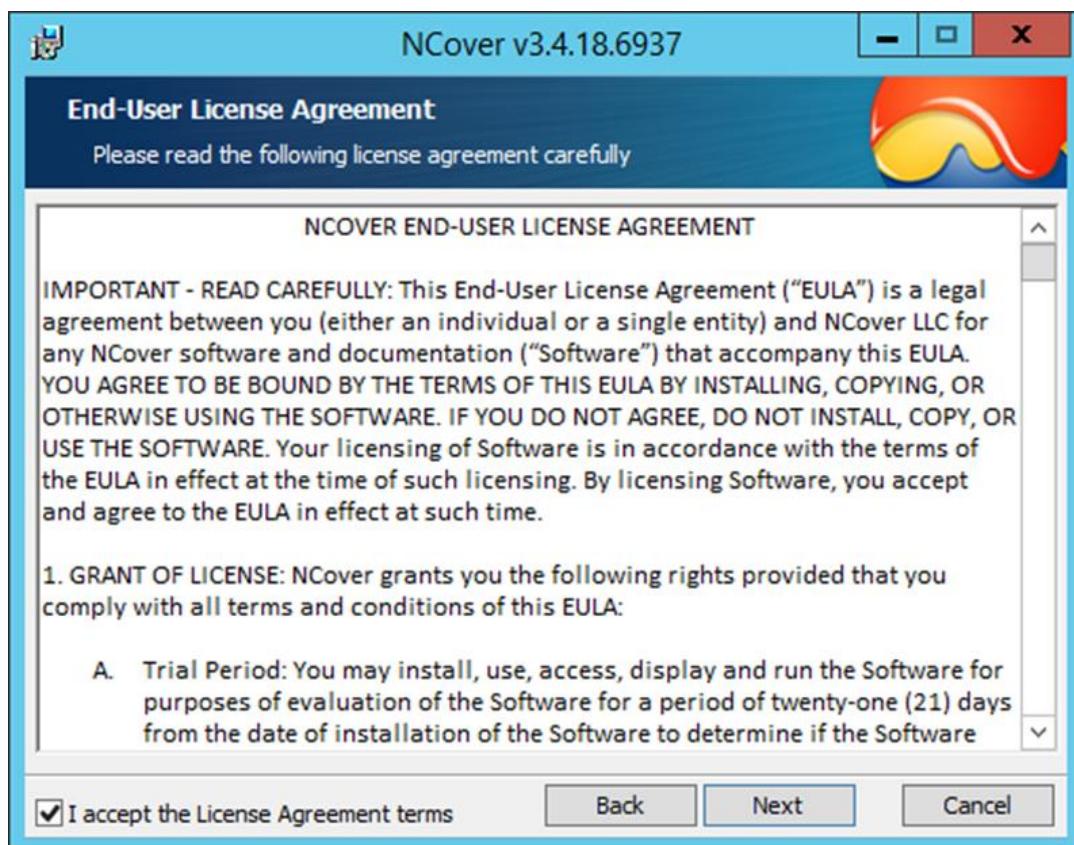
NCover is a separate product which needs to be downloaded and installed. To Download NCover, please click on the following link and download the 32-bit installer – <http://www.ncover.com/info/download>



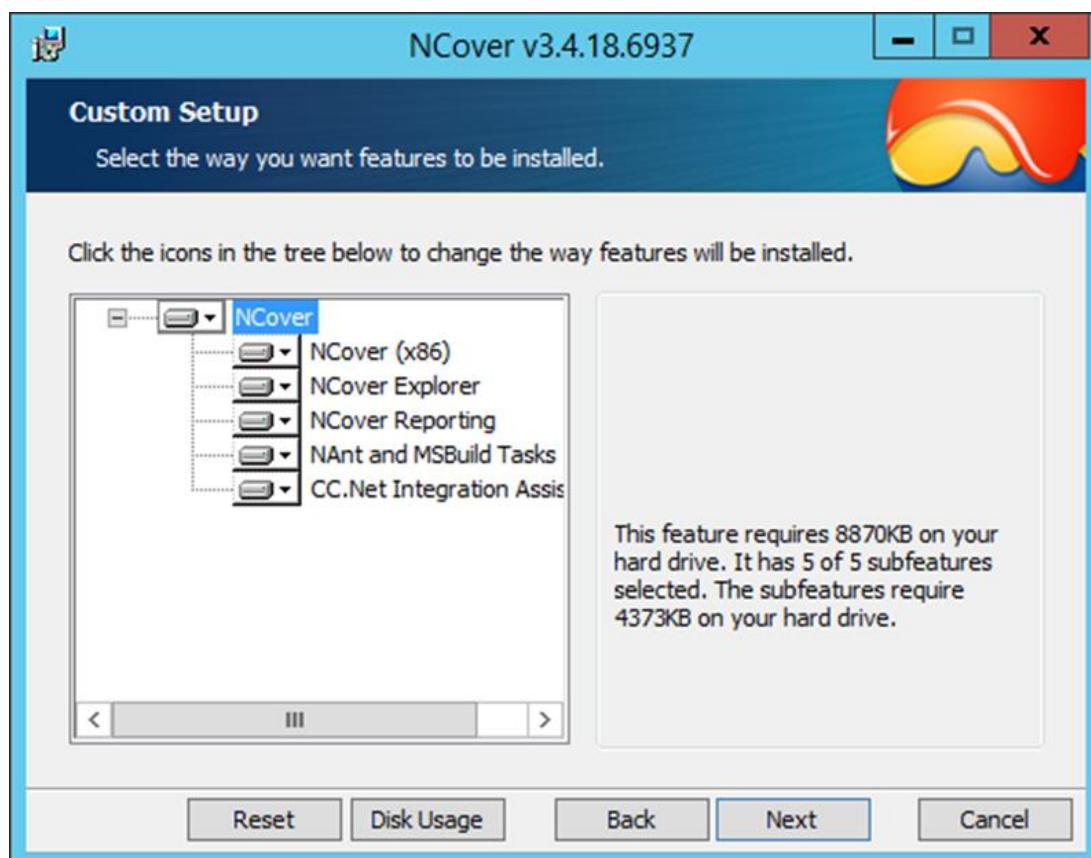
Run the downloaded installer and then click Next after the installer is started.



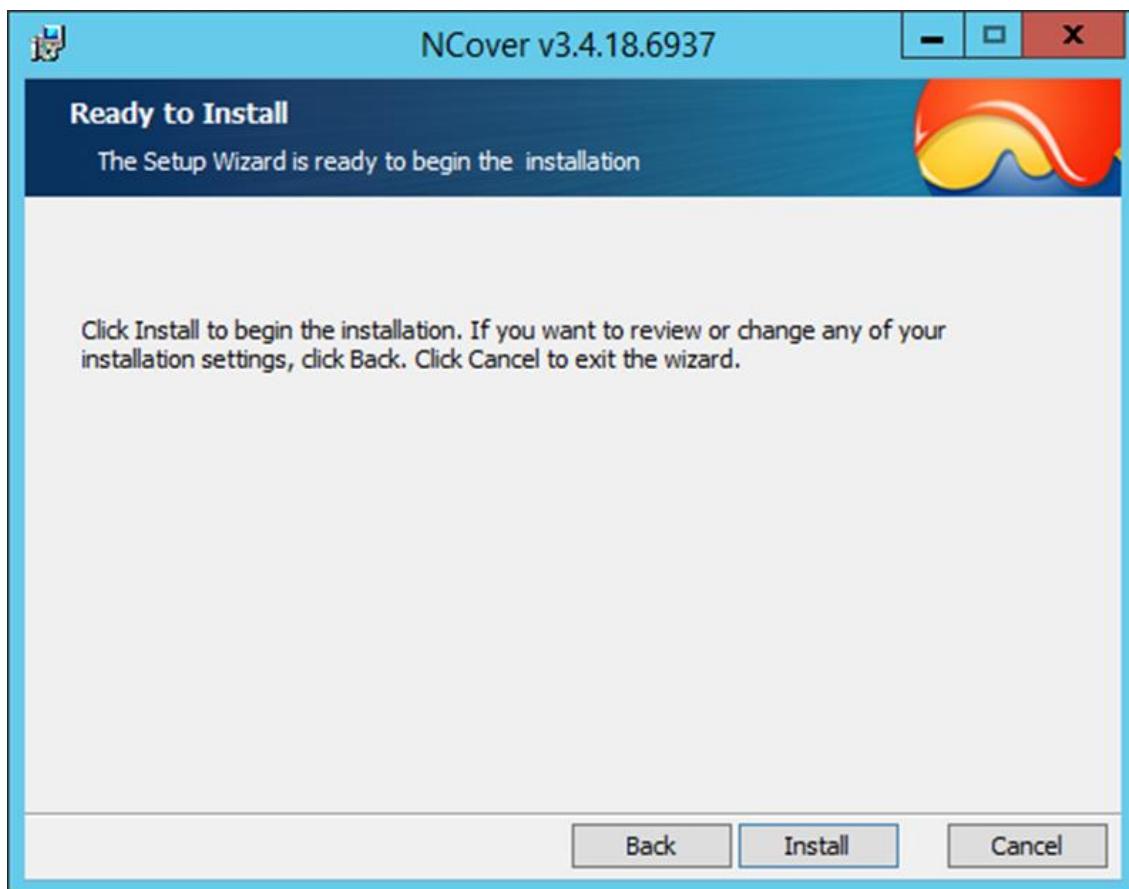
Accept the License agreement and then click Next.



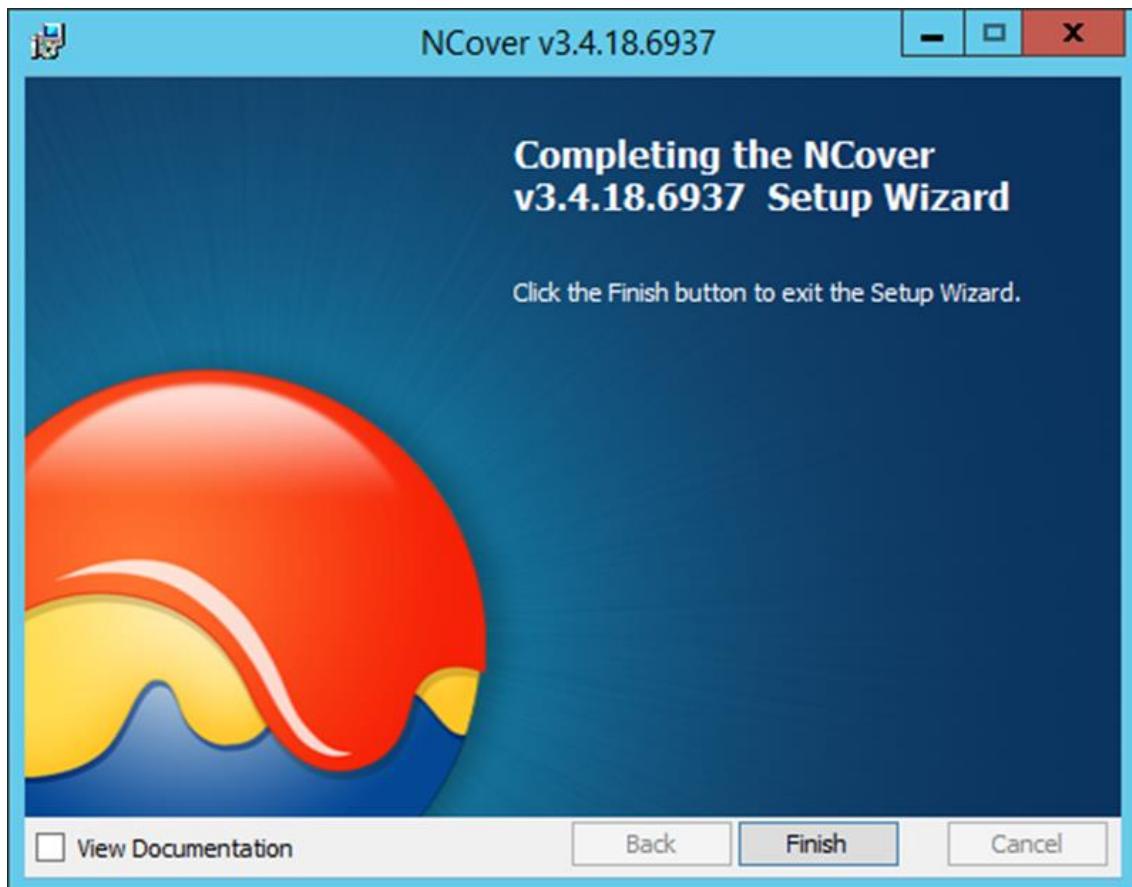
Accept the default components and click Next.



Click on the Install button to begin the installation.



Click the Finish button to complete the installation.



Launch the NCover installation for the first time by going to **C:\Program Files (x86)\NCover\ NCover.Explorer.exe**. You will just need to install a trial key for the first time, which is a straightforward process.

## Configure the Project in TeamCity to Use NCover

**Step 1:** Go to your project home screen and click Edit Configuration Settings.

The screenshot shows the TeamCity interface for the 'DemoBuild' project. At the top, there's a navigation bar with links for 'Projects', 'Changes', 'Agents', 'Build Queue', and 'Administration'. Below that is a breadcrumb trail: 'Demo > DemoBuild'. On the right side of the header, there are buttons for 'Run ...', 'Actions', and 'Edit Configuration Settings'. The main content area has tabs for 'Overview', 'History', 'Change Log', 'Issue Log', 'Statistics', 'Compatible Agents', 'Pending Changes', and 'Settings'. Under 'Pending changes', it says 'No pending changes'. Under 'Current status', it says 'Idle'. The 'Recent history' section contains a table of build logs:

Results	Artifacts	Changes	Started	Duration	Agent	Tags
#18 Tests passed: 1	None	No changes	22 Mar 16 05:41	19s	WIN-50GP30FG075-1	None
#17 Tests passed: 1	None	No changes	22 Mar 16 05:35	10s	WIN-50GP30FG075-1	None
#16 Tests passed: 1	None	No changes	22 Mar 16 04:40	11s	WIN-50GP30FG075-1	None
#15 Success	None	alashro (1)	21 Mar 16 23:59	7s	WIN-50GP30FG075-1	None

**Step 2:** Go to Build Steps and click Edit for the **TestStep**. Continuous Inspection needs to run along with the Unit tests which are defined.

Build Step	Parameters Description	Edit	More
Build	MSBuild Build file: Simple\Simple.csproj Targets: default Execute: If all previous steps finished successfully	Edit	More
TestStep	Visual Studio Tests Test engine: VSTest Included assemblies: DemoTest\bin\Debug\DemoTest.dll Collect .NET code coverage data with NCover (3.x) Execute: If all previous steps finished successfully	Edit	More

**Step 3:** In the .Net Coverage section, click on **.Net Coverage Tool**. And then choose the following settings –

- Choose the .Net Coverage tool as NCover(3.x)
- Platform as x86
- Version as v4.0
- Path to NCover as C:\Program Files (x86)\NCover
- Leave the other settings as they are

**Step 4:** Click Save.

The screenshot shows a web browser window titled "DemoBuild Configuration" with the URL "localhost:8080/admin/editRunType.html?id=buildType:DemoBuild&runnerId=RUNNER\_4". The page contains configuration settings for a .NET Coverage tool:

- .NET Coverage tool:** NCover (3.x) - A dropdown menu with a placeholder "Choose a .NET coverage tool".
- .NET Runtime:** Platform: x86 - A dropdown menu.
- Version:** v4.0 - A dropdown menu.
- Path to NCover 3:** C:\Program Files (x86)\NCover - An input field with a tooltip explaining the path to the NCover 3 installation folder.
- NCover Arguments:** //ias .\* - An input field containing the command-line arguments for the NCover tool.
- Additional NCover Arguments:** An empty input field for additional arguments.

At the top right of the browser window, there is a user name "Alan" and standard window control buttons (minimize, maximize, close).

The screenshot shows a web browser window titled "DemoBuild Configuration" with the URL "localhost:8080/admin/editRunType.html?id=buildType:DemoBuild&runnerId=RUNNER\_4". The page contains several configuration sections:

- NCover Arguments:** A text input field containing the argument `//ias .*`. Below it is a note: "Write additional coverage tool specific arguments. Use //ias .\* to profile all assemblies".
- NCover Reporting Arguments:** A text input field containing the argument `//or FullCoverageReport:Html:{teamcity.report.path}`. Below it is a note: "Write additional NCover.Reporting tool arguments. Use {teamcity.report.path} as path report folder in the reporting commandline arguments. Try //or FullCoverageReport:{teamcity.report.path}" to create full coverage report."
- NCover Report Index File:** An empty text input field. Below it is a note: "Write the name of the index file (i.e. fullcoverage.html) in generated HTML report".

At the bottom left is an orange link " Show advanced options". At the bottom right are two buttons: "Save" and "Cancel".

**Step 5:** Now go to the main screen of your project and click Run.

The screenshot shows the 'DemoBuild Configuration' window in TeamCity. The URL in the browser is `localhost:8080/admin/editBuild.html?id=buildType:DemoBuild`. The navigation path is Administration > <Root project> > Demo > DemoBuild. The 'General Settings' tab is selected. The build configuration is named 'DemoBuild'. The 'Build configuration ID' is also set to 'DemoBuild'. There is a note explaining that this ID is used in URLs, REST API, HTTP requests to the server, and configuration sets in the TeamCity Data Directory. The 'Artifact paths' field is empty. The sidebar on the left lists other settings like Version Control, Build Steps, Triggers, etc. A message at the bottom indicates the configuration was last edited 17 minutes ago by admin. At the bottom right are 'Save' and 'Cancel' buttons. The footer includes links for Help, Feedback, and License agreement.

The screenshot shows the TeamCity web interface at [localhost:8080/overview.html](http://localhost:8080/overview.html). The top navigation bar includes links for Projects, Changes, Agents (1), Build Queue (0), and user admin. The main content area displays the 'Demo' project, which contains the 'DemoBuild' configuration. A successful build (#18) is listed, showing 1 test passed, no artifacts, and no changes, completed 17 minutes ago (19s). Navigation icons for up, down, and left are visible on the left, and filter options 'Hide Successful Configurations' and 'Configure Visible F...' are at the top right. The bottom footer includes links for Help, Feedback, TeamCity Professional 9.1.6 (build 37459), and License agreement.

**Step 6:** Once the build is run, click on the Test passed. You will now see a Code Coverage screen and you will see many metric indicators.

Demo :: DemoBuild > #18 X

localhost:8080/viewLog.html?buildId=20&tab=buildResultsDiv&buildTypeId=DemoBuild

Projects | Changes Agents 1 Build Queue 0 admin | Administration

Demo > DemoBuild > #18 (22 Mar 16 05:41)

Run ... Actions | Edit Configuration Settings |

Overview Changes Tests Build Log Parameters Artifacts Code Coverage

Result: Tests passed: 1 Agent: WIN-50GP30FG075-1

Time: 22 Mar 16 05:41:15 - 05:41:34 (19s) Triggered by: you on 22 Mar 16 05:41

Code coverage summary View full report »

Classes: 57.4% 31/54 Methods: 39.4% 145/368 Blocks: 35% 288/822 Lines: 66.6% 6/9

1 test passed

Help Feedback TeamCity Professional 9.1.6 (build 37459) License agreement

**Step 7:** You can now click the Code Coverage tab to get more information on the Code Analysis.

Demo :: DemoBuild > #18 X TC NCover Code Coverage Report X

localhost:8080/viewLog.html?buildId=20&buildTypeId=DemoBuild&tab=coverage\_dotnet

Projects | Changes Agents 1 Build Queue 0 admin | Administration

Demo > DemoBuild > #18 (22 Mar 16 05:41)

Run ... Actions | Edit Configuration Settings |

Overview Changes Tests Build Log Parameters Artifacts Code Coverage

This is an autogenerated index file (there was no index.html found in the generated report).

- classes.html
- classes\_full.html
- covveragesummary.html
- fullcoveragereport.html
- methods\_full.html
- modules.html
- modules\_full.html
- namespaces.html
- namespaces\_full.html
- sources.html
- sources\_full.html
- uncoveredreport.html
- files10POM355TZohYrEPTXuptnElVX0.html
- files10elQadJZbOCdeiGalBRP25acYrg.html
- files1240TIGX5dalOUFAeanOqenpxA.html
- files14FuxOfUcmMULnUKXnmvV8EzF0M.html
- files177DnD-A-TUW-L-EUTQ9LIP-0Q1uL

**Step 8:** Click the **fullcoverage.html**. You will now get a full comprehensive report on the inspection carried out for the **.Net code**.

The screenshot shows a web browser window with the URL `localhost:8080/viewLog.html?buildId=20&buildTypeId=DemoBuild&tab=coverage_dotnet`. The page title is "Demo :: DemoBuild > #18". The main content is titled "Demo :: DemoBuild Coverage Summary". It displays the following coverage statistics:

Symbol	Branch	Module
100.00%	100.00%	DemoTest.dll
50.00%	50.00%	Simple.dll
N/A	34.84%	vstest.console.exe

Below this, there are sections for Modules (3 modules), Namespaces (7 namespaces), Classes (54 classes), Methods (368 methods), and Documents (3 documents). The overall coverage summary is as follows:

Symbol Coverage: **66.67%** (6 of 9)  
Branch Coverage: **35.04%** (288 of 822)  
Method Coverage: **39.40%** (145 of 368)  
Cyclomatic Complexity Avg: **1.44** Max:**14**

# 17. CI – Continuous Database Integration

Continuous Database Integration is the process of rebuilding your database and test data any time a change is applied to a project's version control repository.

In Database Integration, generally all the artifacts related to the database integration –

- Should reside in a version control system.
- Can be tested for rigor and inspected for policy compliance.
- Can be generated using your build scripts.

Activities that can be involved in Continuous Database Integration can be any one of the following –

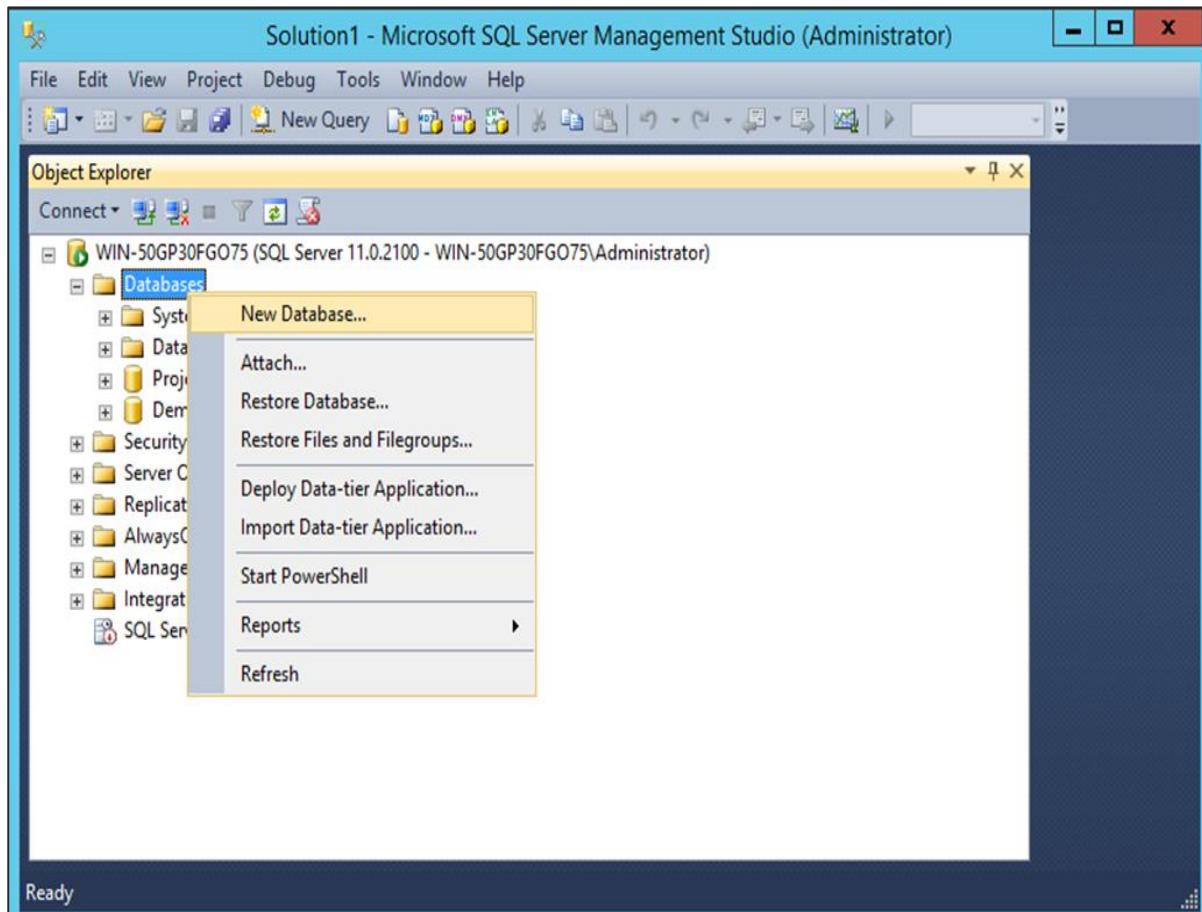
- **Drop a Database** – Drop the database and remove the associated data, so that you can create a new database with the same name.
- **Create a new Database** – Create a new database using Data Definition Language (DDL).
- **Insert the Initial Data** – Insert any initial data (e.g., lookup tables) that your system is expected to contain when delivered.
- **Migrate Database and Data** – Migrate the database schema and data on a periodic basis (if you are creating a system based on an existing database).
- **Modify Column Attributes** – Modify table column attributes and constraints based on requirements and refactoring.
- **Modify Test Data** – Alter test data as needed for multiple environments.

So in our Continuous Database example, we are going to do the following steps –

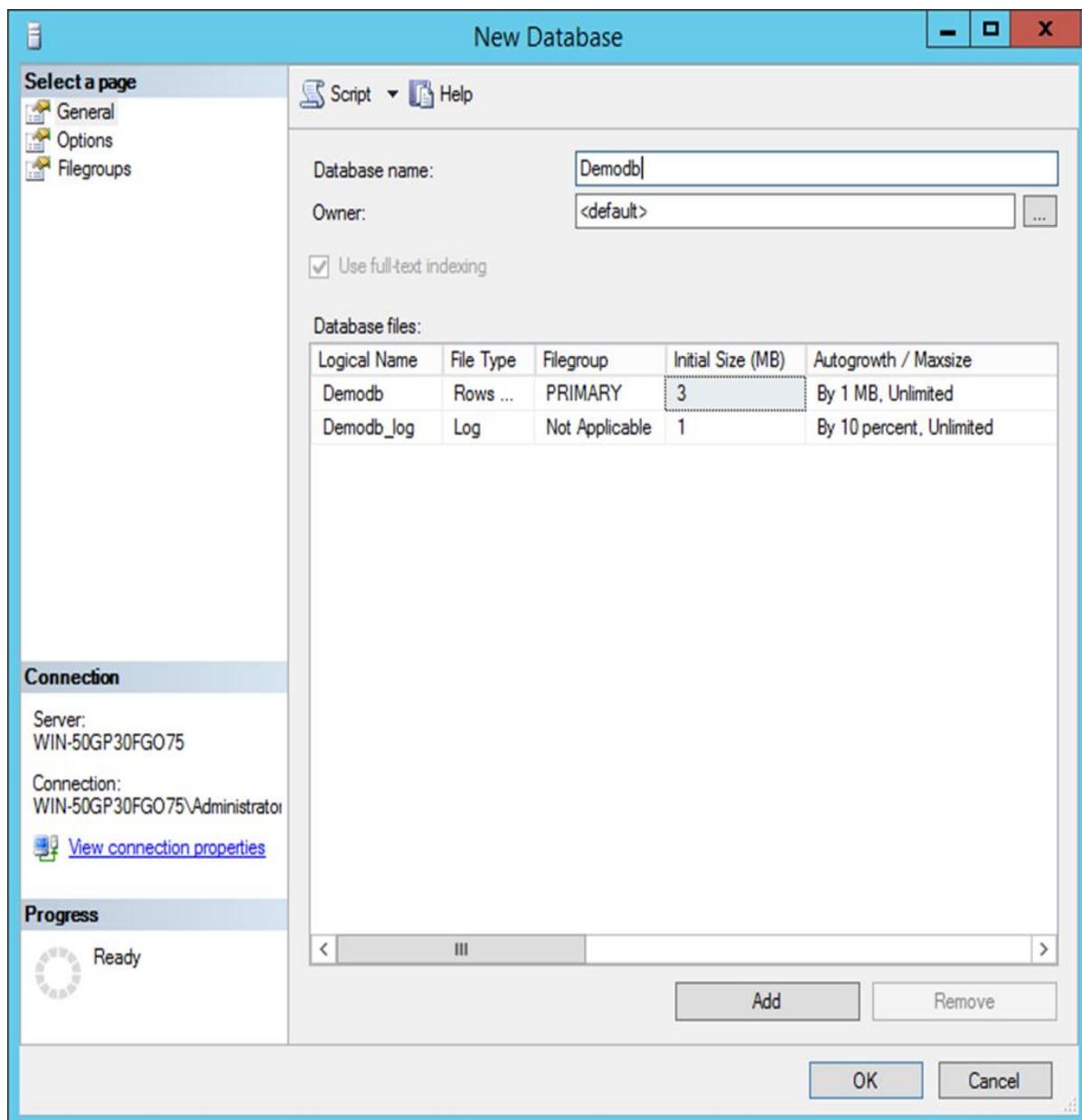
- We will create a MS SQL Server database and a corresponding table.
- We will create a script out of SQL Server Management Studio. This database script will be used to set up our table in the database.
- We will write a code in our ASP.Net project to access this database.
- We will create a step in our project in TeamCity to run this script.
- We will check in our script into Git.

Steps to do this in the AWS database which was created in an earlier section.

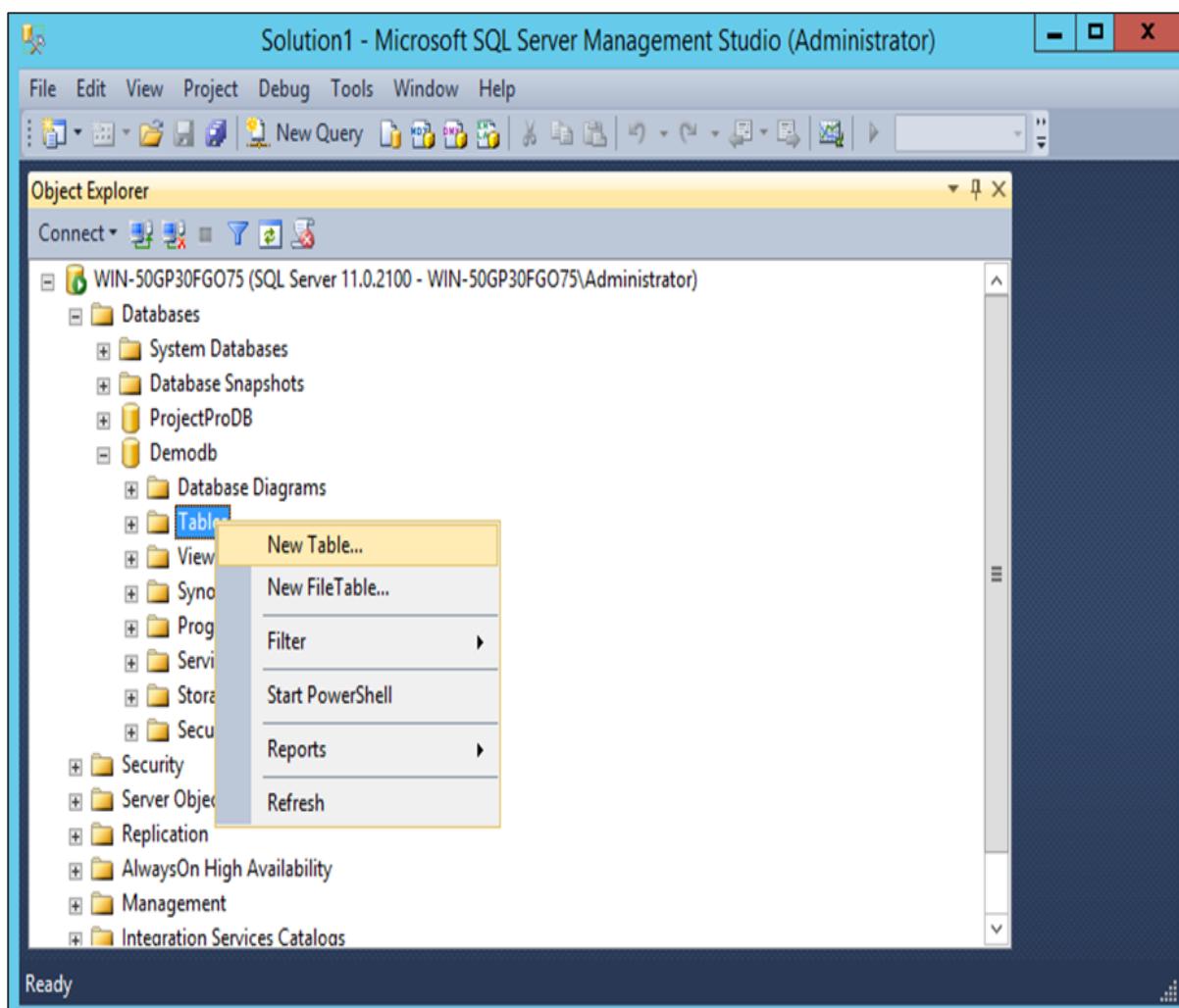
**Step 1:** Create an MS SQL Server database and a corresponding table. Let's open SQL Server Management Studio and create a simple database and table. Right-click databases and click on **New Database**.



**Step 2:** Name it as **Demodb** and click OK.



**Step 3:** In the new database, right-click and create a new table.

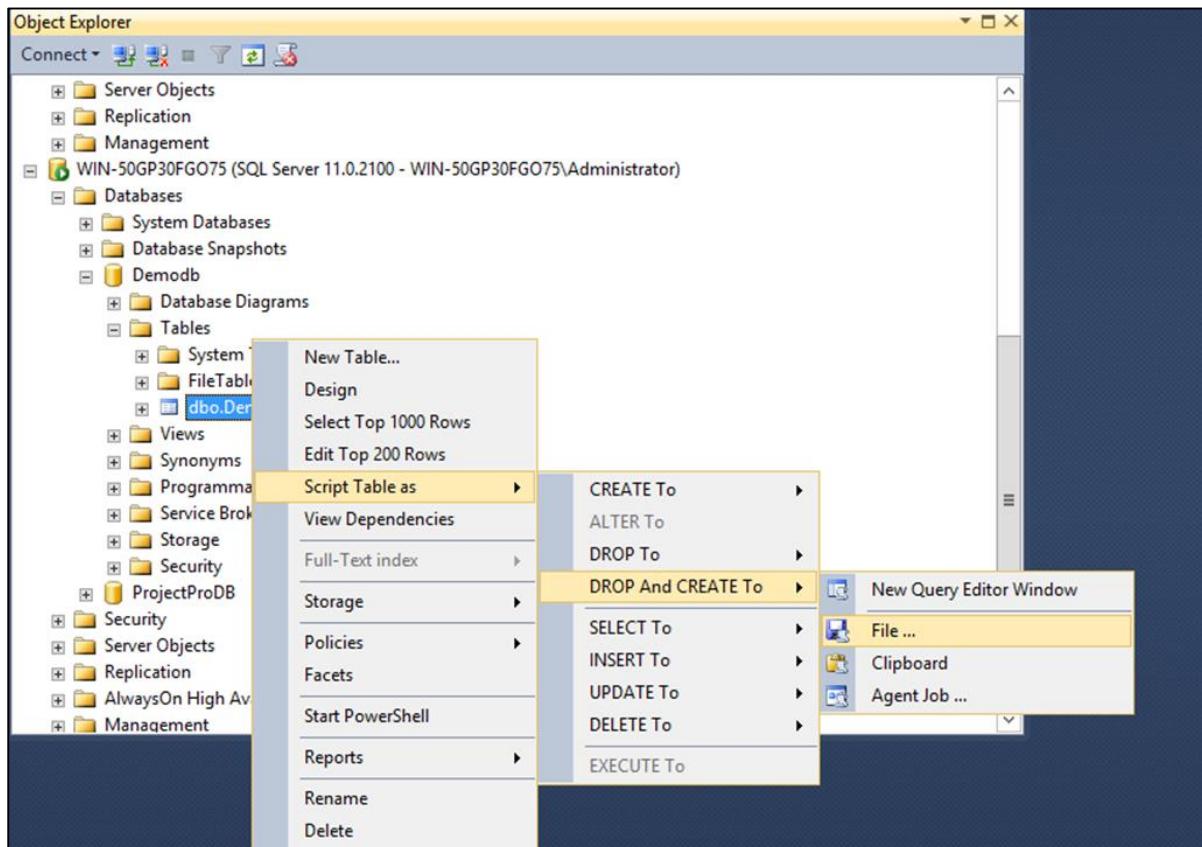


**Step 4:** You can add your desired columns to the table.

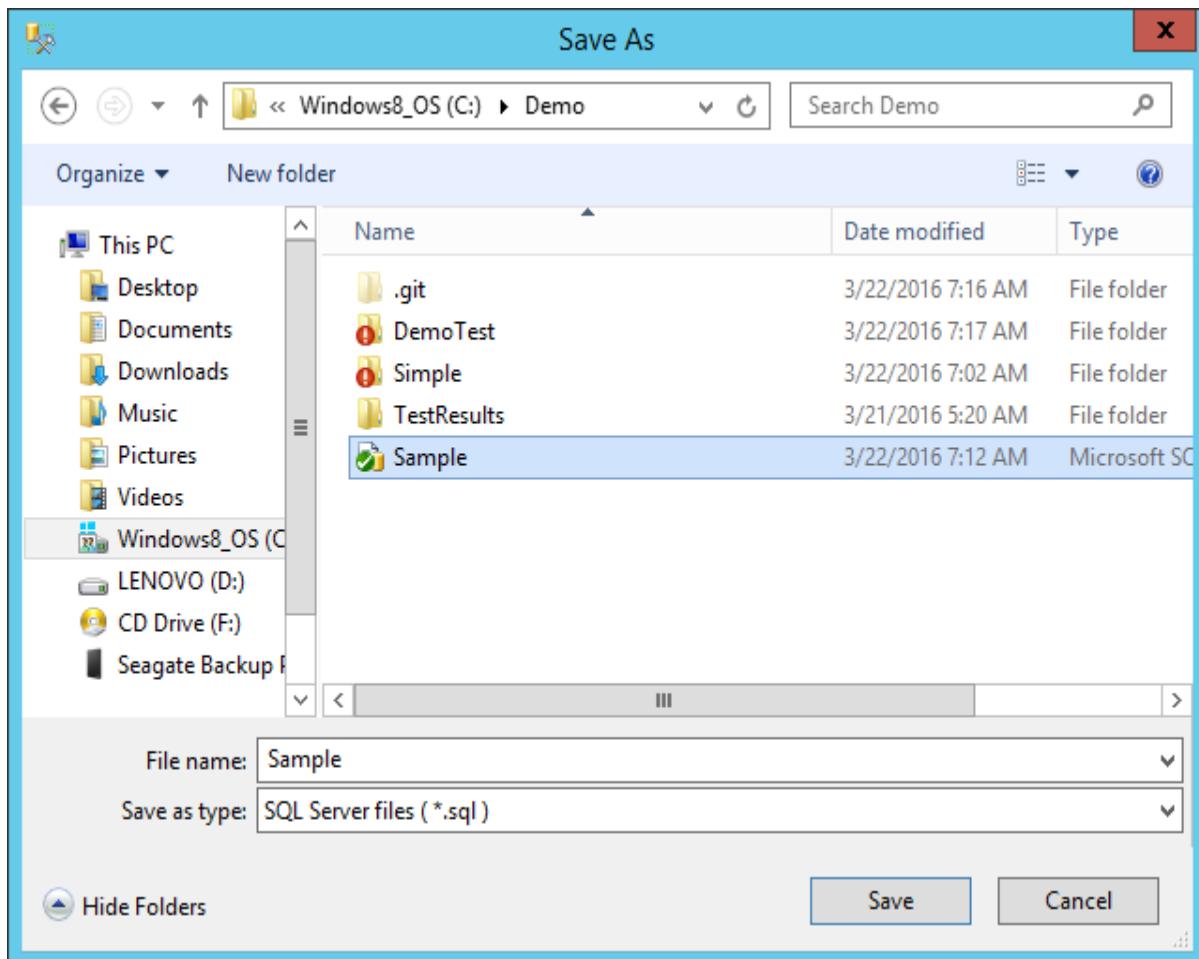
The screenshot shows the Microsoft SQL Server Management Studio interface. The title bar reads "WIN-50GP30FG075.Demodb - dbo.Table\_1\* - Microsoft SQL Server Management Studio (Administrator)". The main window is titled "WIN-50GP30FG075....odb - dbo.Table\_1\*". On the left, the Object Explorer pane shows the database structure, including the "Demodb" database and its tables. The central area displays the "Table Designer" for the "Table\_1" table. A single column named "TutorialName" is defined, with the data type set to "nvarchar(MAX)" and the "Allow Nulls" option checked. Below the table definition, the "Column Properties" pane is open, showing the properties for the "TutorialName" column: Name (TutorialName), Allow Nulls (Yes), Data Type (nvarchar(MAX)), and Default Value or Binding (empty). The "Table Designer" pane at the bottom is currently empty.

**Step 5:** Save the table and name it as **Demotb**.

**Step 6:** Now right-click on the table and choose the menu option **Script Table as -> Drop and Create to -> File.**



**Step 7:** Save the file to the demo project folder as **Sample.sql**.



This is what the database script would look like. It would first drop an existing table if present and then re-create the table.

```
USE [Demodb]
GO

/******** Object: Table [dbo].[Demotb]      Script Date: 3/22/2016 7:03:25 AM
***** 

DROP TABLE [dbo].[Demotb]
GO

/******** Object: Table [dbo].[Demotb]      Script Date: 3/22/2016 7:03:25 AM
***** 
SET ANSI_NULLS ON
```

```

GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Demotb](
    [TutorialName] [nvarchar](max) NULL,
    [TutorialID] [smallint] NULL
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]

GO

```

**Step 8:** Now let's quickly change our **ASP.Net code** to refer to the new database.

**Step 9:** In the **Tutorial.cs** file in your **Demo project**, add the following lines of code. These lines of code will connect to your database, take the Server version and store the version name in the Name variable. We can display this Name variable in our **Demo.aspx.cs** file through a **Response.write** command.

```

using System;
using System.Collections.Generic;
using System.Data.SqlClient;
using System.Linq;
using System.Web;

namespace Simple
{
    public class Tutorial
    {
        public String Name;
        public Tutorial()
        {
            string connectionString = "Data Source=WIN-50GP30FG075;Initial Catalog=Demodb;Integrated Security=true;";

            using (SqlConnection connection = new SqlConnection())
            {
                connection.ConnectionString = connectionString;

```

```
        connection.Open();
        Name = connection.ServerVersion;
        connection.Close();
    }

}

}

}
```

**Step 10:** Add the following code to the **Demo.aspx.cs** file to ensure that it displays the SQL Server version.

```
using System;
using System.Collections.Generic;
using System.Data.SqlClient;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

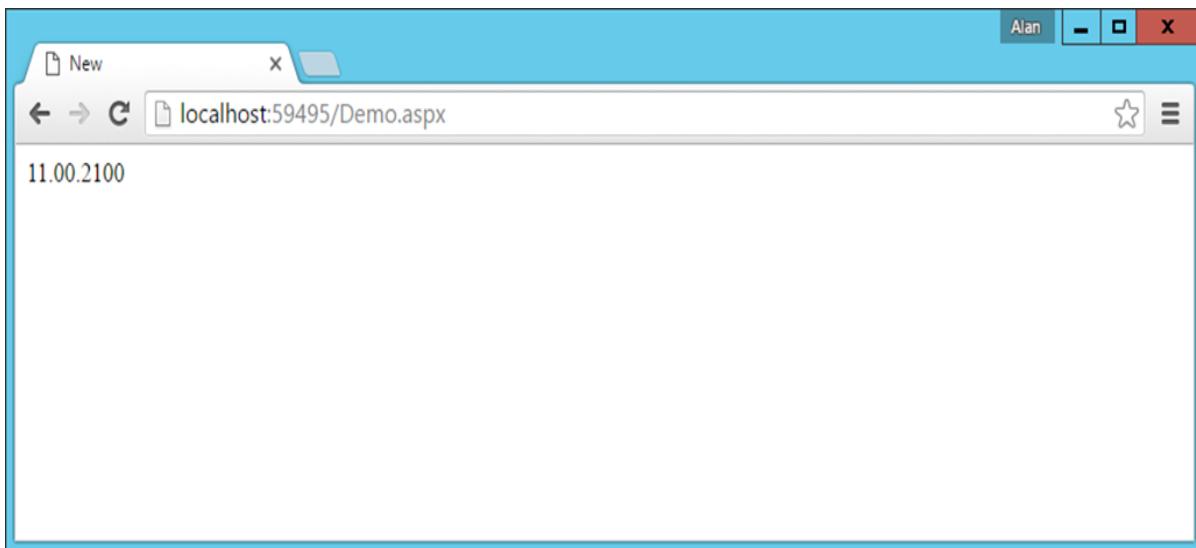
namespace Simple
{

    public partial class Demo : System.Web.UI.Page
    {
        Tutorial tp = new Tutorial();

        protected void Page_Load(object sender, EventArgs e)
        {
            Response.Write(tp.Name);

        }
    }
}
```

Now if we run the code, you will get the following output in the browser.



**Step 11:** Now let us add our step in TeamCity which will invoke the database script. Go to your project dashboard and click **Edit Configuration Settings**.

Results	Artifacts	Changes	Started	Duration	Agent	Tags
#18 Tests passed: 1	None	No changes	22 Mar 16 05:41	19s	WIN-50GP30FG075-1	None
#17 Tests passed: 1	None	No changes	22 Mar 16 05:35	10s	WIN-50GP30FG075-1	None
#16 Tests passed: 1	None	No changes	22 Mar 16 04:40	11s	WIN-50GP30FG075-1	None
#15 Success	None	alashro (1)	21 Mar 16 23:59	7s	WIN-50GP30FG075-1	None

**Step 12:** Go to Build Steps and click **Add build step**.

Build Step	Parameters Description	Edit	More
Build	MSBuild Build file: Simple\Simple.csproj Targets: default Execute: If all previous steps finished successfully	Edit	More
TestStep	Visual Studio Tests Test engine: VSTest Included assemblies: DemoTest\bin\Debug\DemoTest.dll Collect .NET code coverage data with NCover (3.x) Execute: If all previous steps finished successfully	Edit	More

Choose the following options (Note that MS SQL Server client should be installed on the CI Server).

- Runner type should be the Command Line.
- Give an optional Step Name.
- Run should be Executable with parameters.
- Command executable should be **C:\Program Files\Microsoft SQL Server\110\Tools\Binn\sqlcmd.exe**
- Command parameters should be **-S WIN-50GP30FG075 -i Sample.sql**. Where -S gives the name of the SQL Server instance.

**Step 13:** Click Save.

DemoBuild Configuration

localhost:8080/admin/editRunType.html?id=buildType:DemoBuild&runnerId=RUNNER\_5

Projects | Changes | Agents | Build Queue | admin | Administration | Search

Administration > <Root project> > Demo > DemoBuild

Build Step (1 of 3): Databasesetup

Runner type: Command Line  
Simple command execution

Step name: Databasesetup  
Optional, specify to distinguish this build step from other steps.

Run: Executable with parameters

Command executable: \* C:\Program Files\Microsoft SQL Server\110\Tools\Binn\sqlcmd

Command parameters: -S WIN-50GP30FG075 -i Sample.sql

Last edited 36 minutes ago by admin (view history)

Show advanced options

Save Cancel

Now what needs to be ensured is the build order. You have to ensure the build order is as follows.

**Step 14:** You can change the build order by choosing the option to reorder build steps.

- The database setup should be first – So this will be used to recreate your database from fresh.
- Next is the build of your application.
- Finally your test setup.

**Build Steps**

In this section you can configure the sequence of build steps to be executed. Each build step is represented by a build runner and provides integration with a specific build or test tool.

Build Step	Parameters Description
Databasesetup	Command Line Command: C:\Program Files\Microsoft SQL Server\110\Tools\Binn\sqlcmd.exe -S WIN-50GP30FG075 -i Sample.sql Execute: If all previous steps finished successfully
Build	MSBuild Build file: Simple\Simple.csproj Targets: default Execute: If all previous steps finished successfully
TestStep	Visual Studio Tests Test engine: VSTest Included assemblies: DemoTest\bin\Debug\DemoTest.dll Collect .NET code coverage data with NCover (3.x) Execute: If all previous steps finished successfully

**Step 15:** Now run the **git add** and **git commit** command so that the **Sample.sql** file is checked into Git. This will trigger a build automatically. And this build should pass.

```
C:\Demo>git add .
C:\Demo>git commit -m "Database commit"
```

You now have a full-fledged build cycle with a continuous database integration aspect as well in your cycle. In the next section, let's take this further and look at Continuous Deployment.

Now that you have done this with a local SQL Server, we can repeat the same steps for a **AWS MS SQL Server** which was created in one of the earlier sections. To connect to a Microsoft SQL Server, you need to connect via the following convention.

**Step 16:** First see what is the name assigned to your database instance in AWS. When you log-in to the AWS, go to the RDS section under the database section.

The screenshot shows the AWS Management Console home page with a light blue header bar. The URL in the address bar is <https://ap-southeast-1.console.aws.amazon.com/console/home?refId=1>. The top right corner shows the user name "Alan" and standard window control buttons.

The main content area is organized into several sections:

- Import/Export:** Snowball, Large Scale Data Transport.
- Storage:** Storage Gateway, Hybrid Storage Integration.
- Database:** RDS, Managed Relational Database Service; DynamoDB, Managed NoSQL Database; ElastiCache, In-Memory Cache; Redshift, Fast, Simple, Cost-Effective Data Warehousing; DMS, Managed Database Migration Service.
- Networking:** VPC, Isolated Cloud Resources.
- Service Catalog:** Create and Use Standardized Products.
- Trusted Advisor:** Optimize Performance and Security.
- Application Services:**
  - API Gateway:** Build, Deploy and Manage APIs.
  - AppStream:** Low Latency Application Streaming.
  - CloudSearch:** Managed Search Service.
  - Elastic Transcoder:** Easy-to-Use Scalable Media Transcoding.
  - SES:** Email Sending and Receiving Service.
  - SQS:** Message Queue Service.
  - SWF:** Workflow Service for Coordinating Application Components.
- Security & Identity:**
  - Identity & Access Management:** Manage User Access and Encryption Keys.
  - Directory Service:** Host and Manage Active Directory.
  - Inspector (PREVIEW):** Analyze Application Security.
  - WAF:** Filter Malicious Web Traffic.
  - Certificate Manager:** Provision, Manage, and Deploy SSL/TLS Applications.
- Enterprise Applications:**

**Service Health:** All services operating | Updated: Mar 22 2016 22:54:01 | Service Health Dashboard

**Step 17:** Click on DB Instances in the next screen that comes up.

The screenshot shows the AWS RDS Dashboard. On the left sidebar, under the 'Instances' section, there is a link labeled 'Click here to increase DB instances limit'. The main content area is divided into two columns: 'Resources' and 'Additional Information'. The 'Resources' column lists various Amazon RDS resources: DB Instances (2/40), Parameter Groups (2), Allocated Storage (40.00 GB/100.00 TB), Reserved DB Purchases (0/40), Snapshots (19), Manual (0/50), Automated (19), and Subnet Groups (1/20). The 'Additional Information' column provides links to 'Getting Started with RDS', 'Overview and Features', 'Documentation', 'Articles and Tutorials', 'Data import guide for MySQL', 'Data import guide for Oracle', 'Data import guide for SQL Server', 'Pricing', and 'Forums'. At the bottom of the page, there are links for 'Feedback', 'English', 'Privacy Policy', and 'Terms of Use', along with a copyright notice: '© 2008 - 2016, Amazon Web Services, Inc. or its affiliates. All rights reserved.'

**Step 18:** Click on your database and make a note of the endpoint. In the following screenshot, it is **demodb.cypphcv1d87e.ap-southeast-1.rds.amazonaws.com:1433**

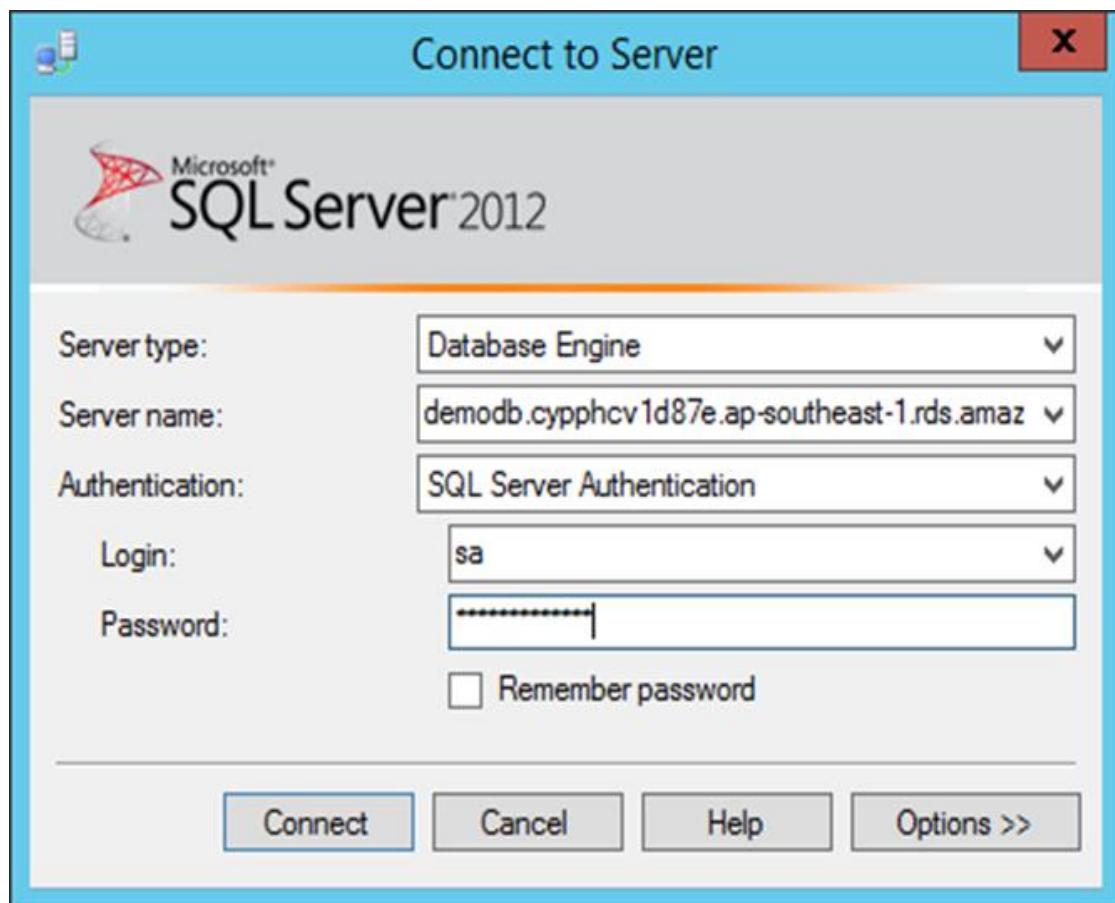
The screenshot shows the AWS RDS Dashboard. On the left, there's a sidebar with links like Instances, Reserved Purchases, Snapshots, Security Groups, Parameter Groups, Option Groups, Subnet Groups, Events, Event Subscriptions, and Notifications. The main area displays two DB instances. A table provides details for one instance:

Engine	DB Instance	Status	CPU
SQL Server Express	demodb	available	1.17%

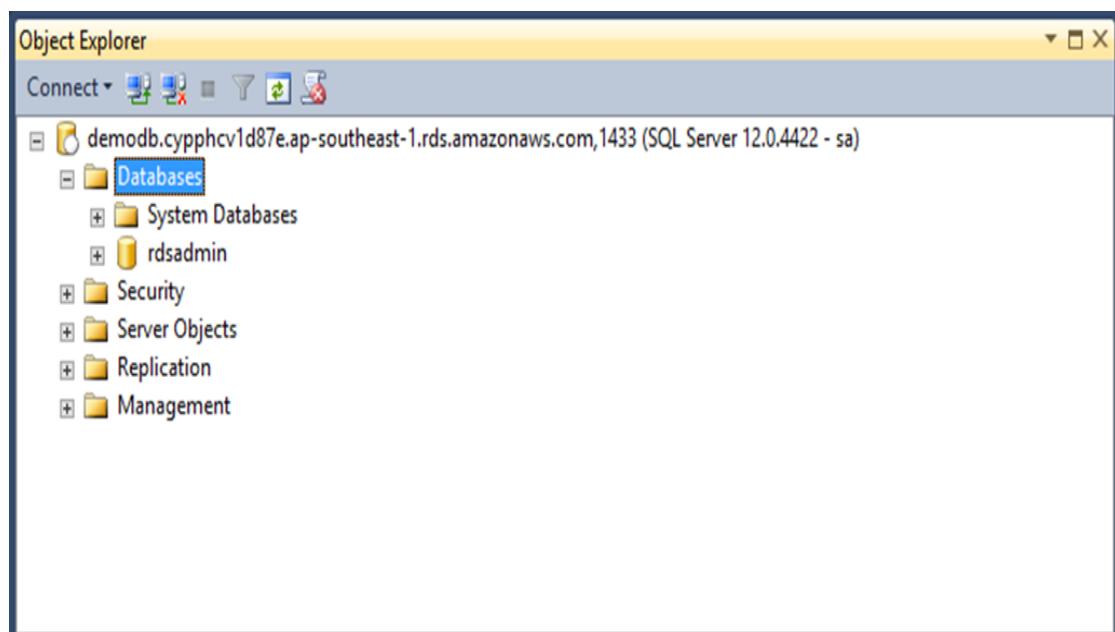
Below the table, the endpoint is listed as **Endpoint: demodb.cypphcv1d87e.ap-southeast-1.rds.amazonaws.com:1433 (authorized)**. There are sections for **Alarms and Recent Events** and **Monitoring**.

Feedback English Privacy Policy Terms of Use © 2008 - 2016, Amazon Web Services, Inc. or its affiliates. All rights reserved.

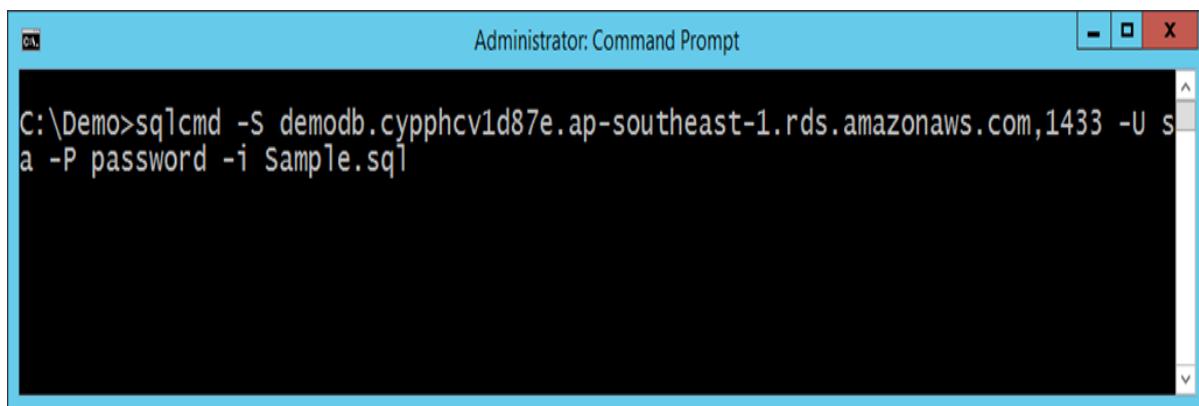
**Step 19:** Now to connect to the database from **SQL Server Management Studio**, you need to specify the connection as **demodb.cypphcv1d87e.ap-southeast-1.rds.amazonaws.com,1433** (Note the comma used between instance name and port no)



The following screenshot shows a successful connection to the database.

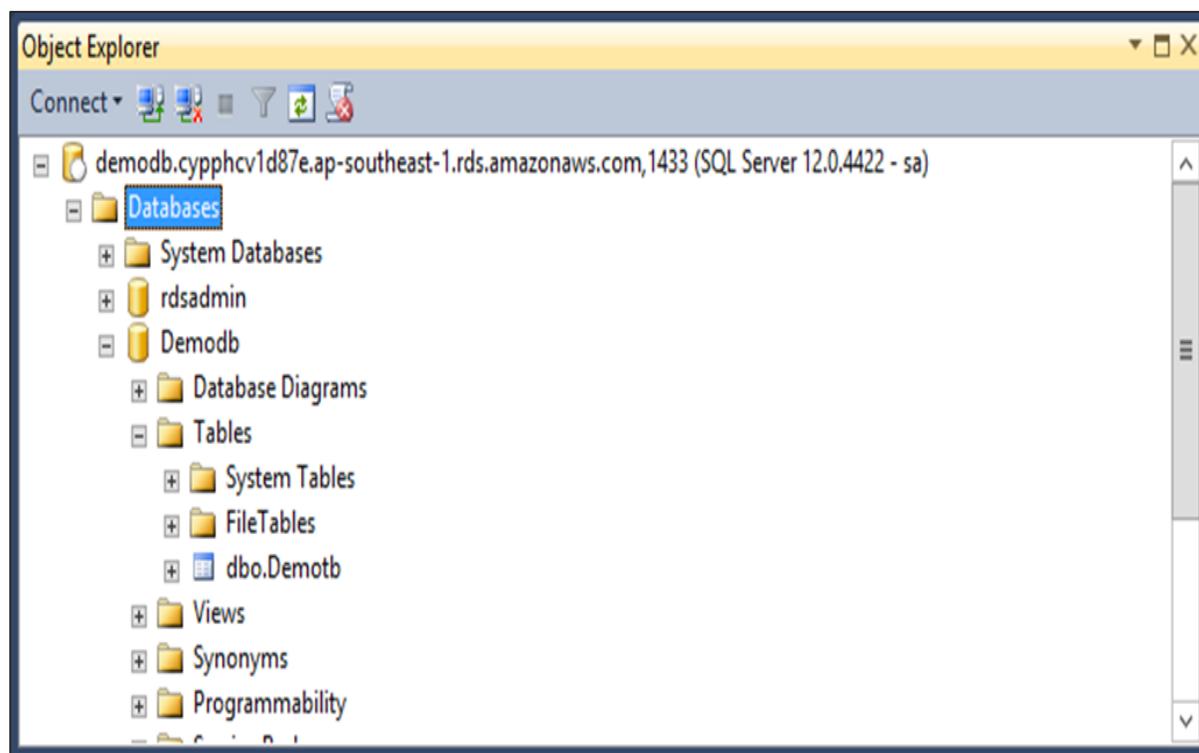


Then you can repeat all the same steps. The **Sqlcmd command** will be as follows:



```
C:\Demo>sqlcmd -S demodb.cypphcv1d87e.ap-southeast-1.rds.amazonaws.com,1433 -U sa -P password -i Sample.sql
```

This same command can be replaced in the Database build step in TeamCity. When you execute the **sqlcmd command**, the table will be created automatically in your SQL Server database in AWS.



# 18. CI – Continuous Deployment

Automated builds and repeatable builds. Automated tests and repeatable tests. Test categories and test frequencies. Continuous inspections. Continuous database integration. These string of tasks in creating an effective CI environment primarily enables one key benefit: releasing working software at any point in time, in any environment.

In our previous chapters, we have accomplished all of the following segments:

- Created our code.
- Ensured a proper build in TeamCity.
- Created a Database Integration process.
- Conducted successful testing.

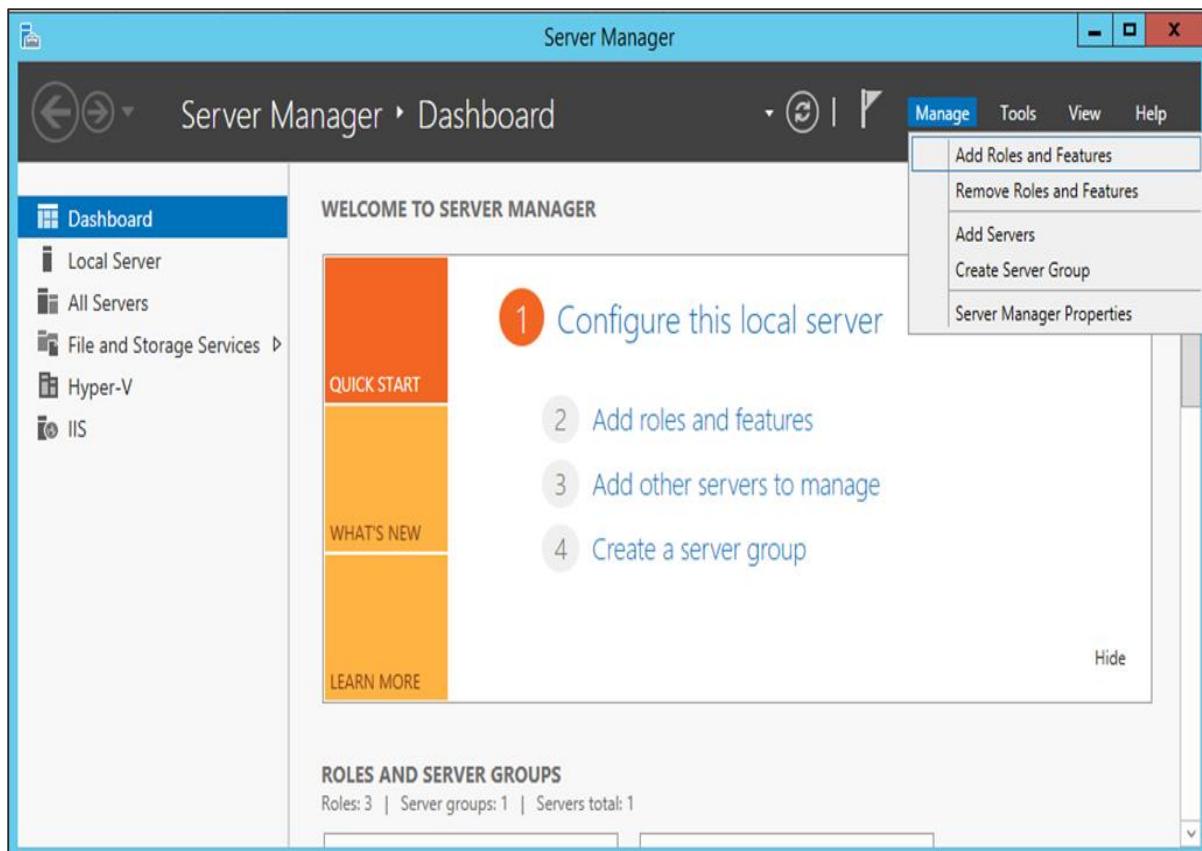
Now the only thing remaining is to carry out an automated deployment, so that our entire process is complete.

For an automated deployment in our case, we need to follow these steps:

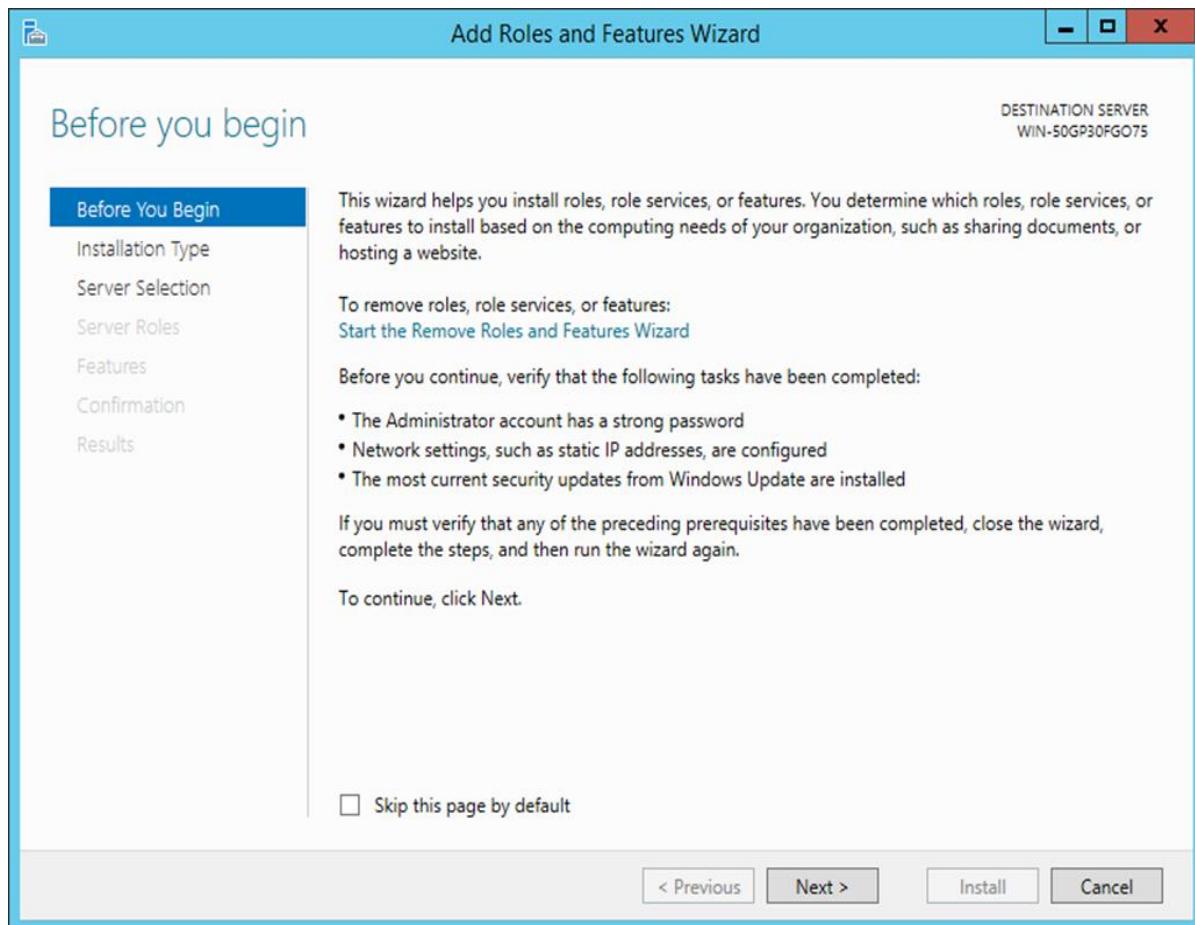
- In our deployment server, ensure that IIS is installed.
- Ensure that IIS user is given access to our database.
- Create a publish profile which will be used to publish the site when it is built.
- Ensure we change our MSBuild command to do an automatic deployment.
- Automate TeamCity to do an automatic publish.
- Do a **git commit** to ensure all your files are in Git.

**Step 1:** Configure a local IIS Server. If you have a local or remote IIS Server, the following configuration can be carried out to deploy our application. It's always a good practice to see if a deployment can be done manually before it is done in an automated fashion.

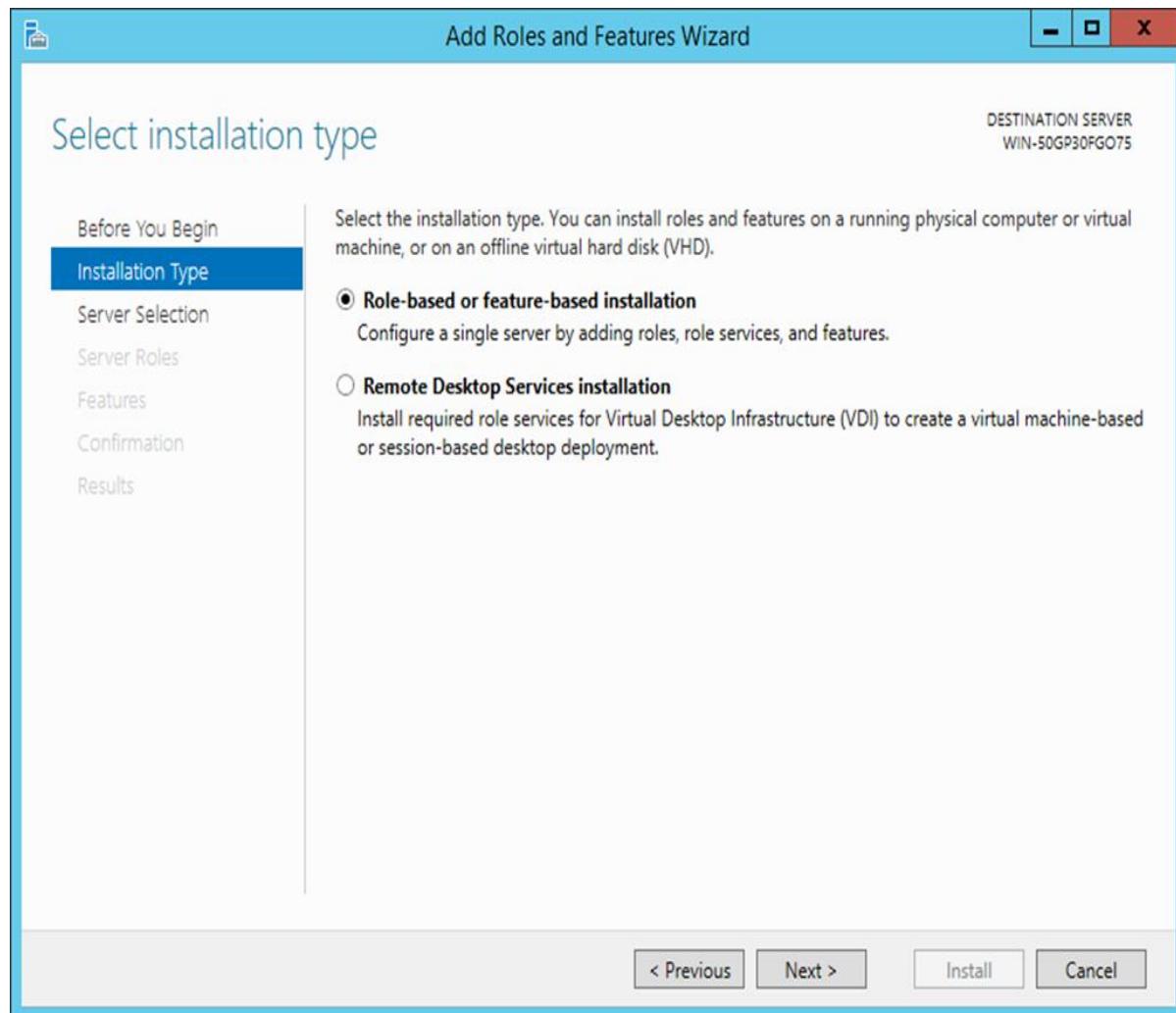
**Step 2:** On a Windows 2012 server, go to your Server Manager and click on Add Roles and Features.



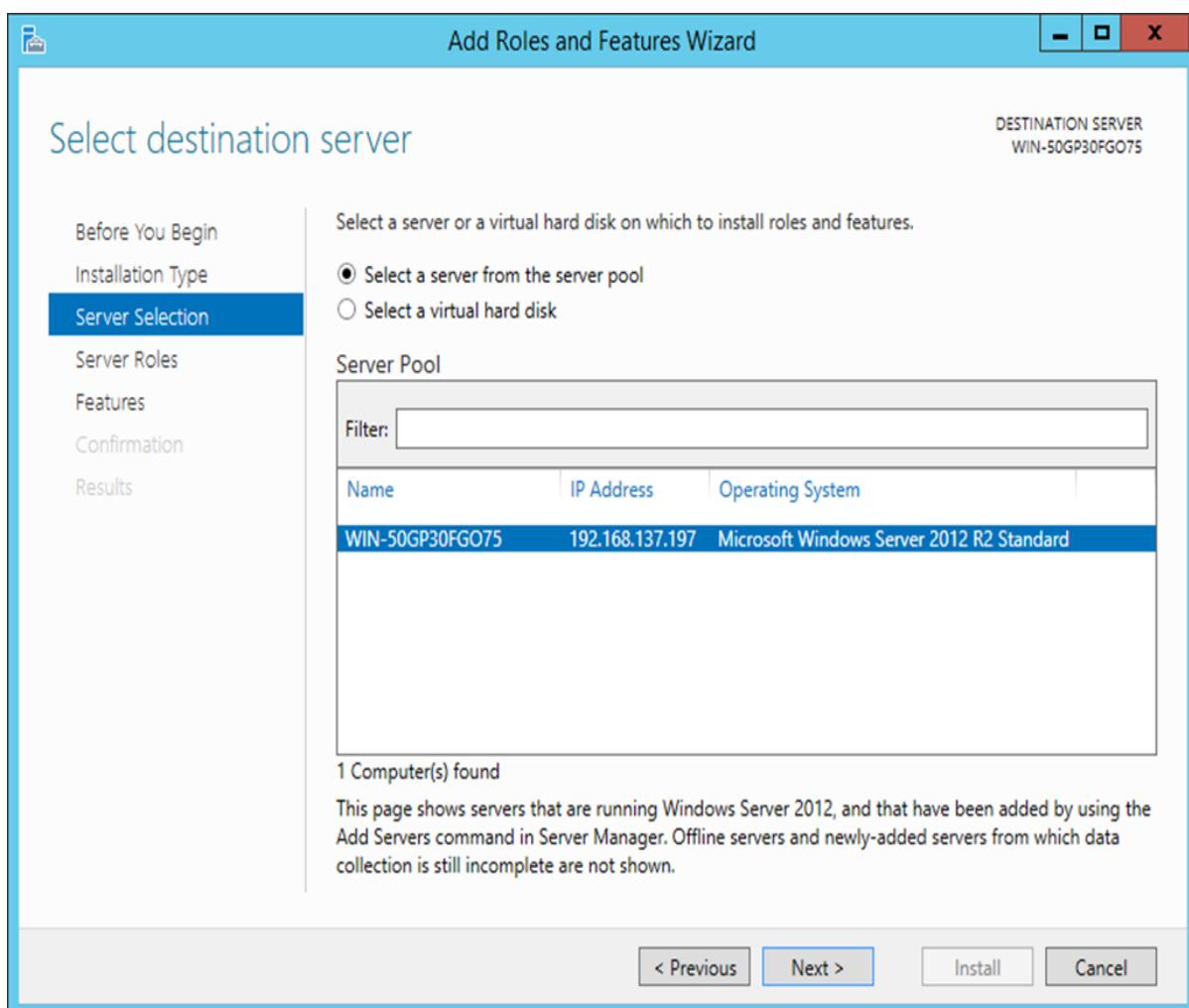
**Step 3:** Click Next on the following screen that comes up.



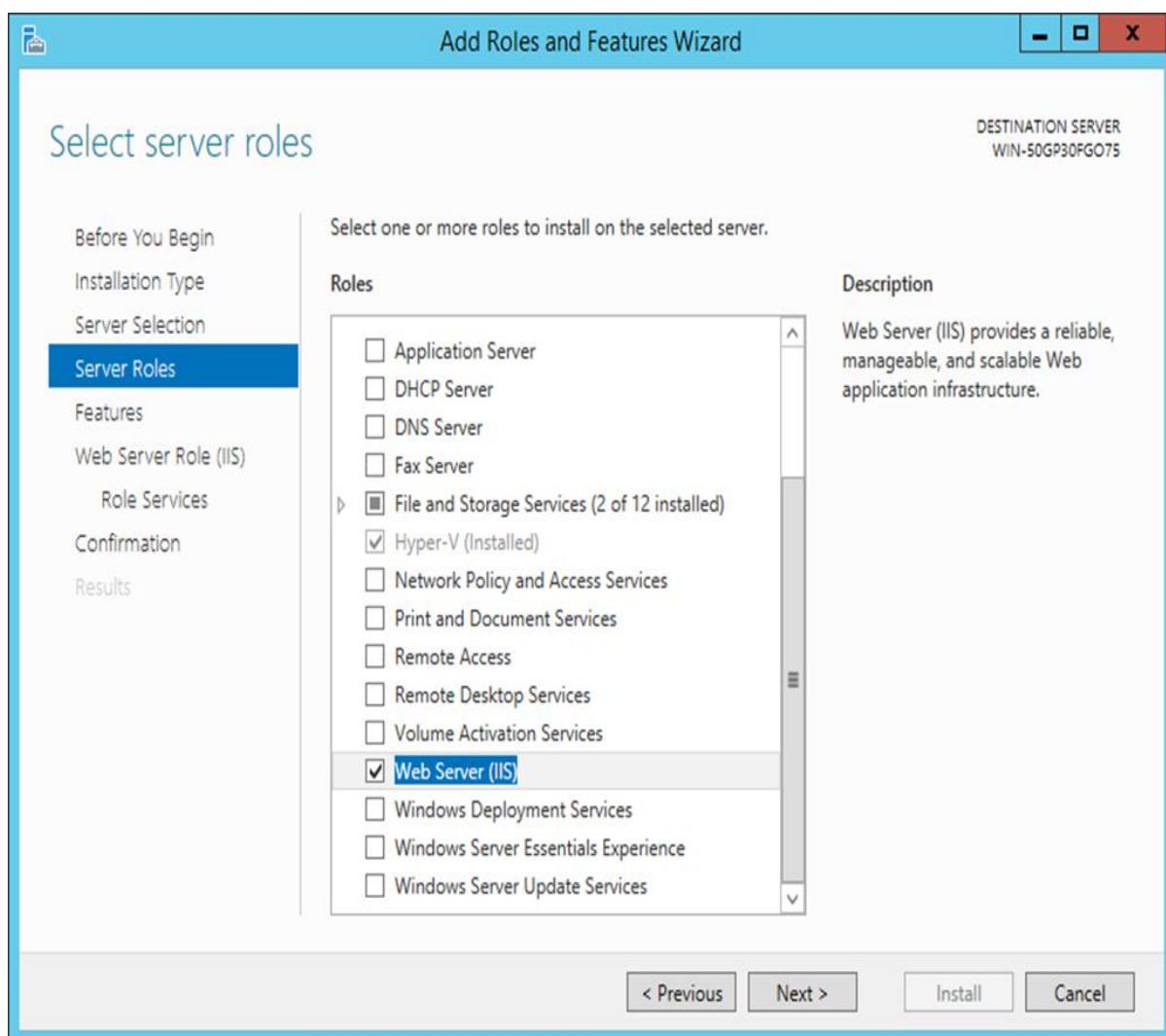
**Step 4:** Choose roles-based or feature-based installation on the next screen and click Next.



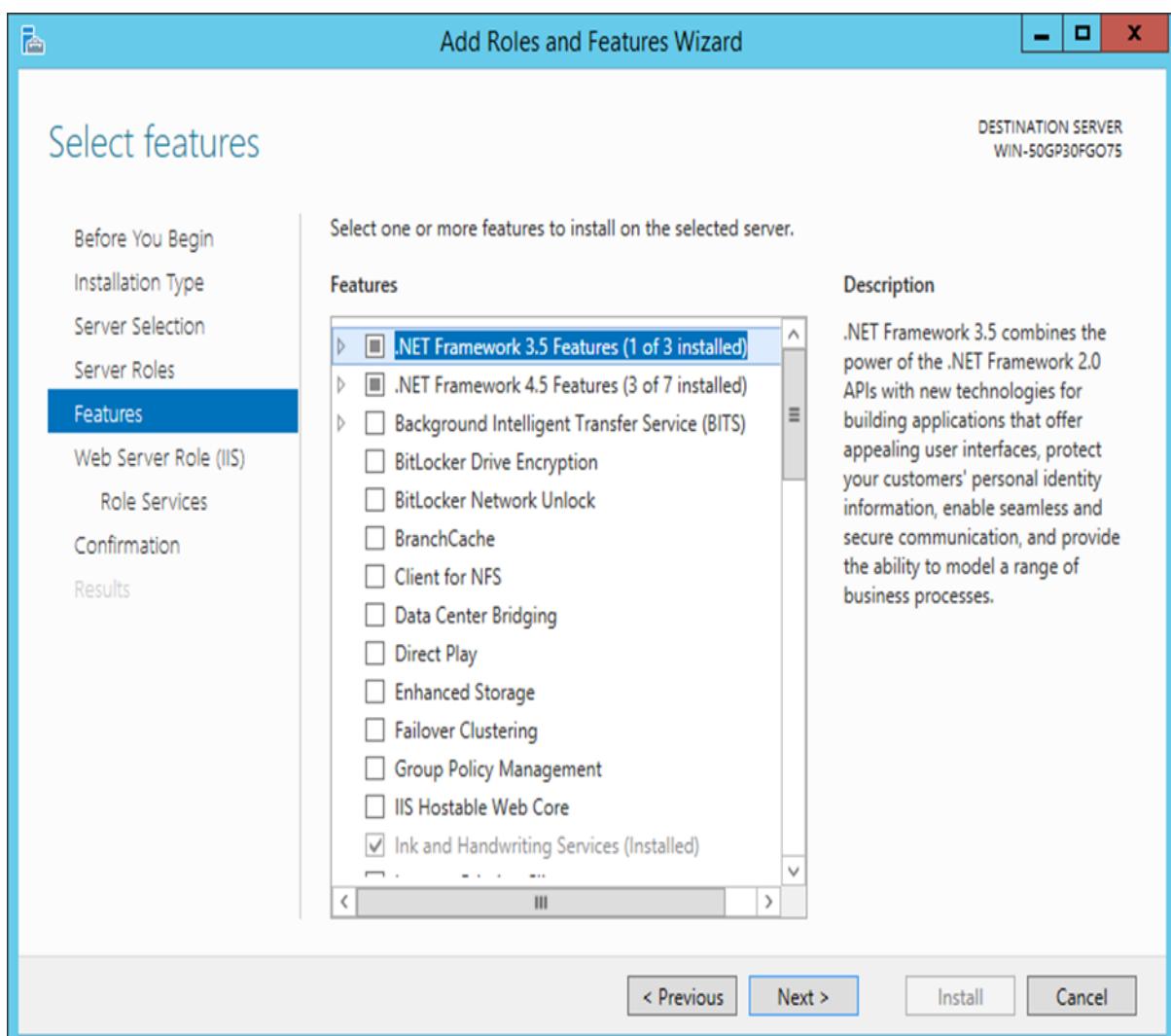
**Step 5:** Select the default server and click Next.



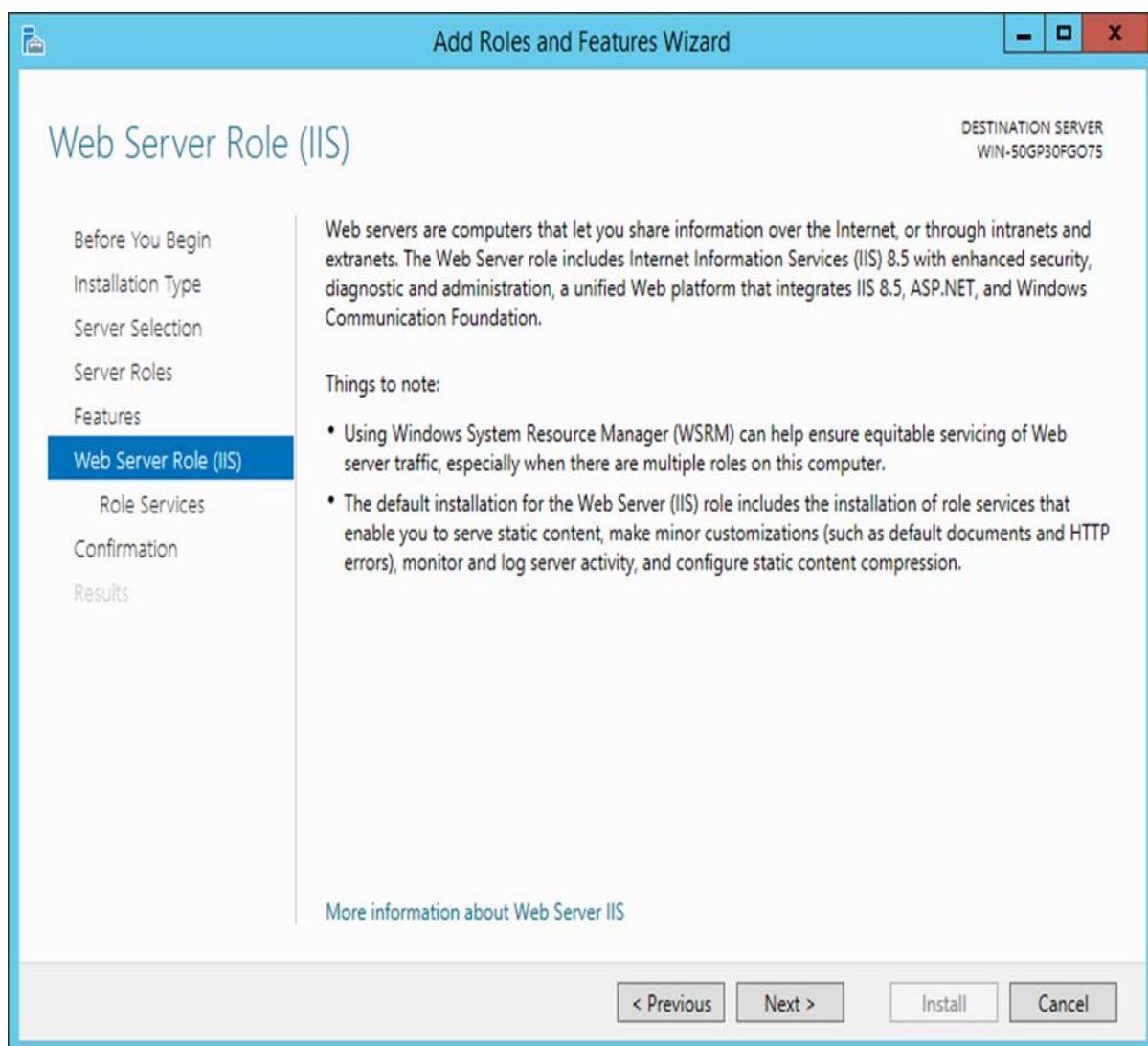
**Step 6:** Choose the Web server role and click Next.



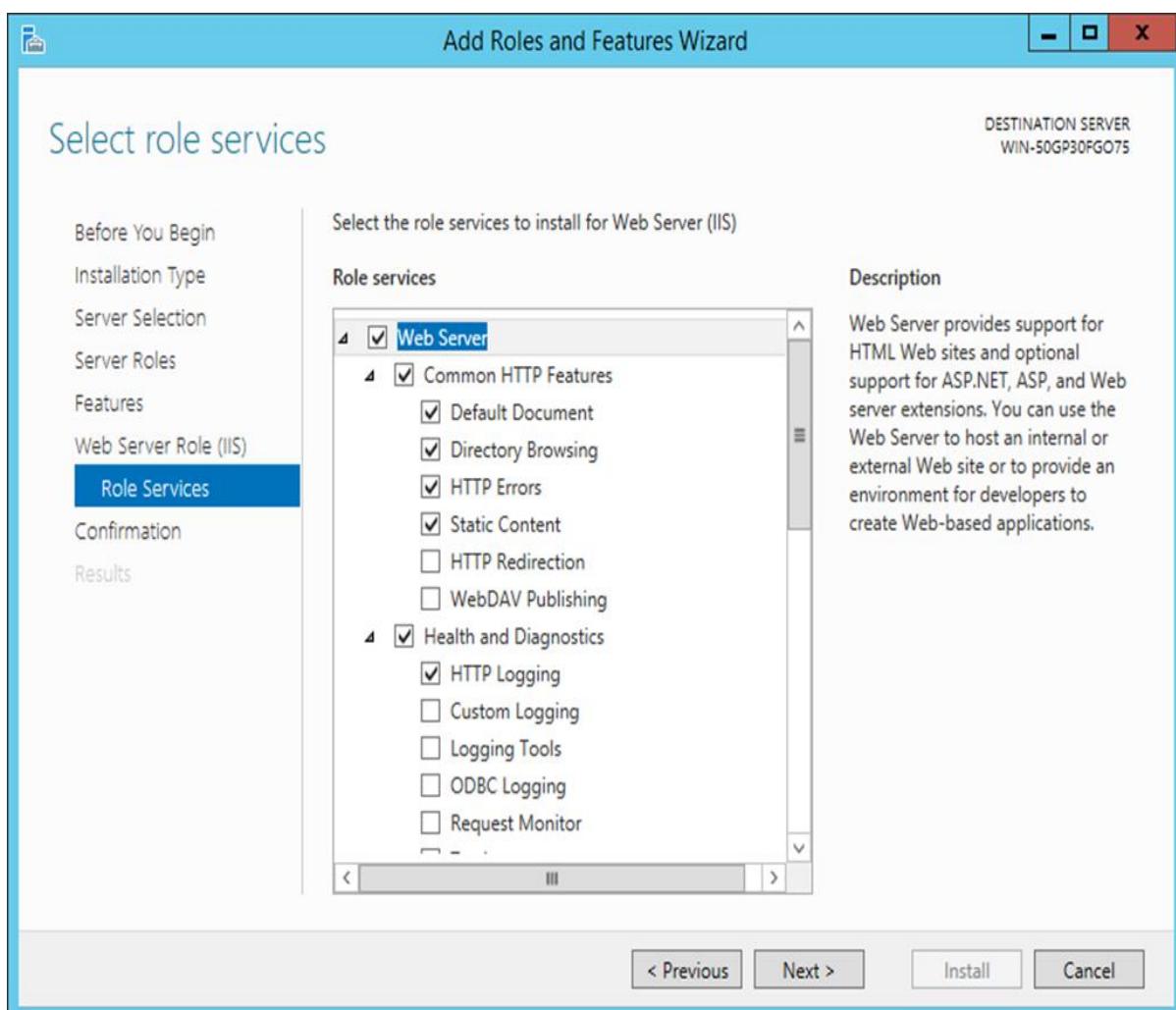
**Step 7:** In the next screen that comes up, click Next.



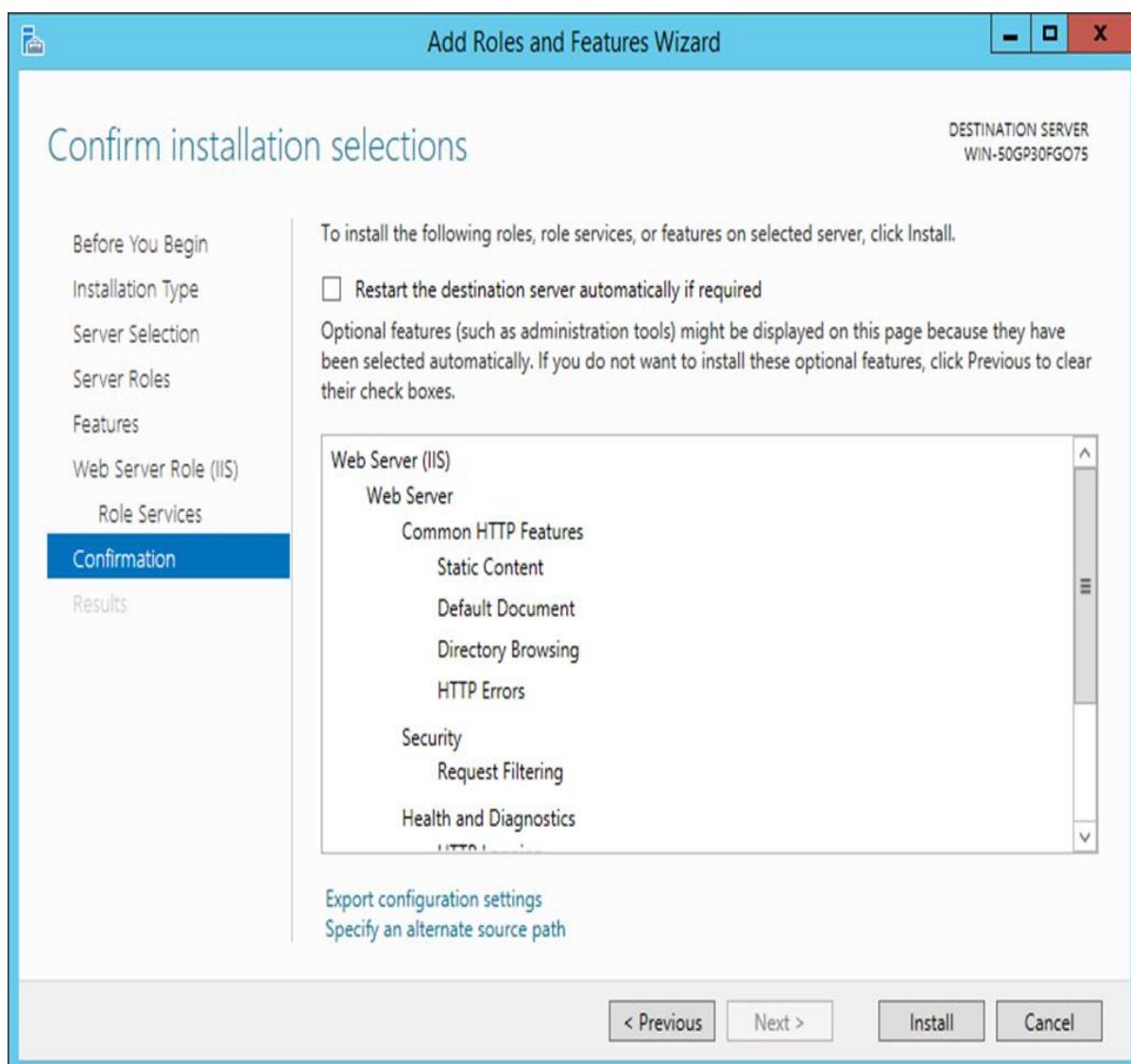
**Step 8:** Click Next again on the following screen that appears.



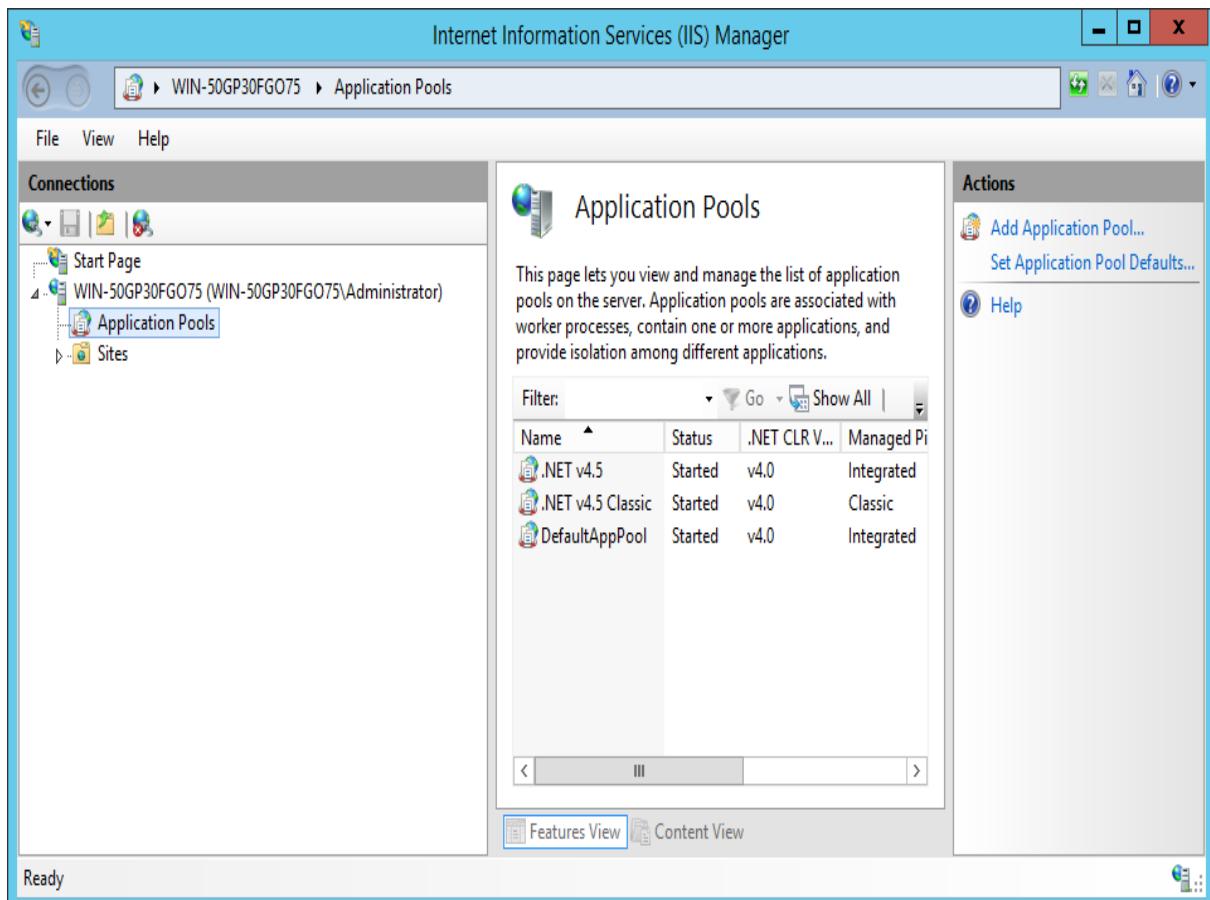
**Step 9:** In the next screen that pops up, click Next.



**Step 10:** In the final screen, you can click the Install button to install the IIS.



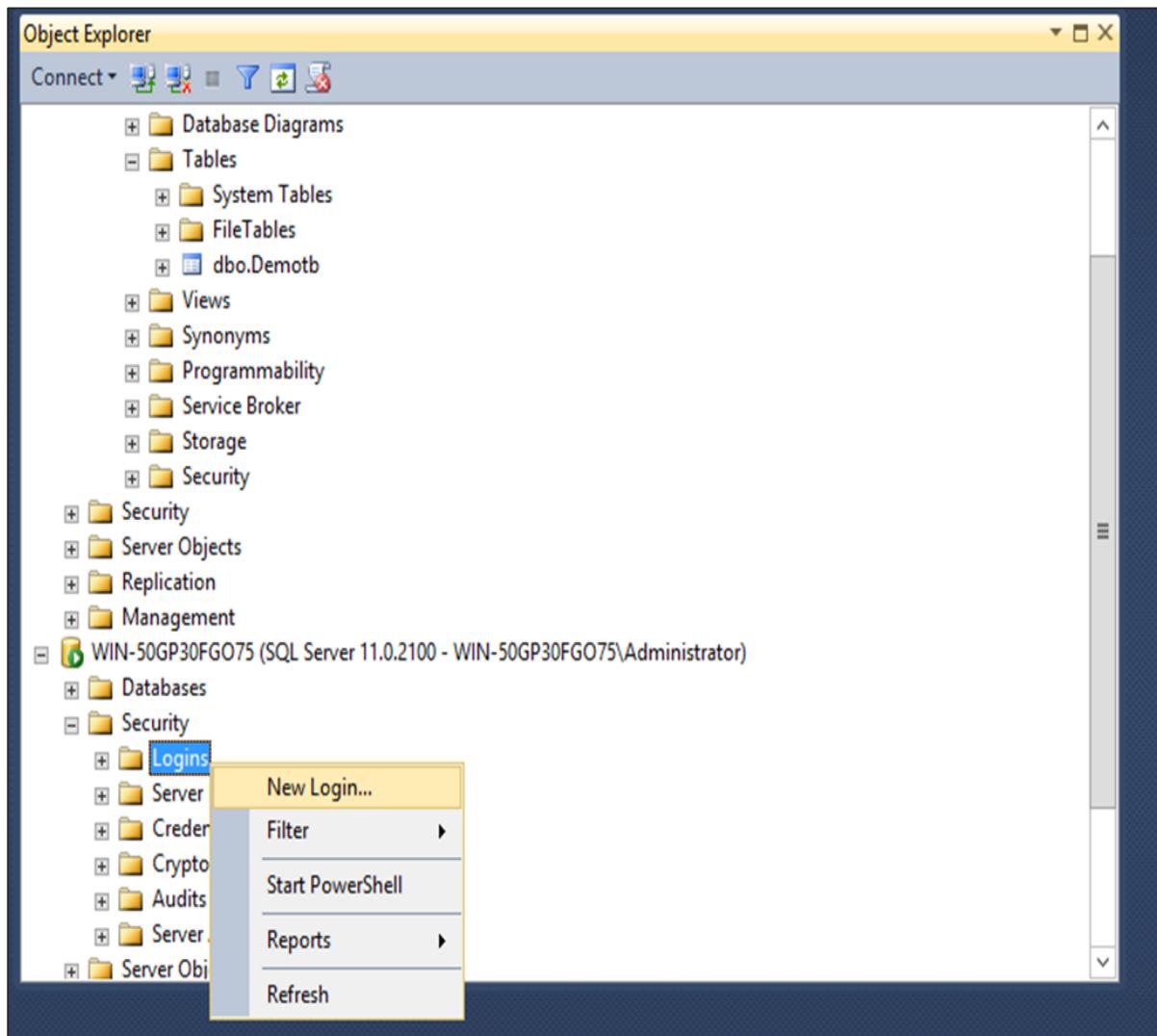
Once you have IIS installed, you can open it by opening the Internet Information Services.



**Step 11:** Click Application Pools, you will see a pool with the name of **DefaultAppPool**. This needs to have access to SQL Server in the next step.

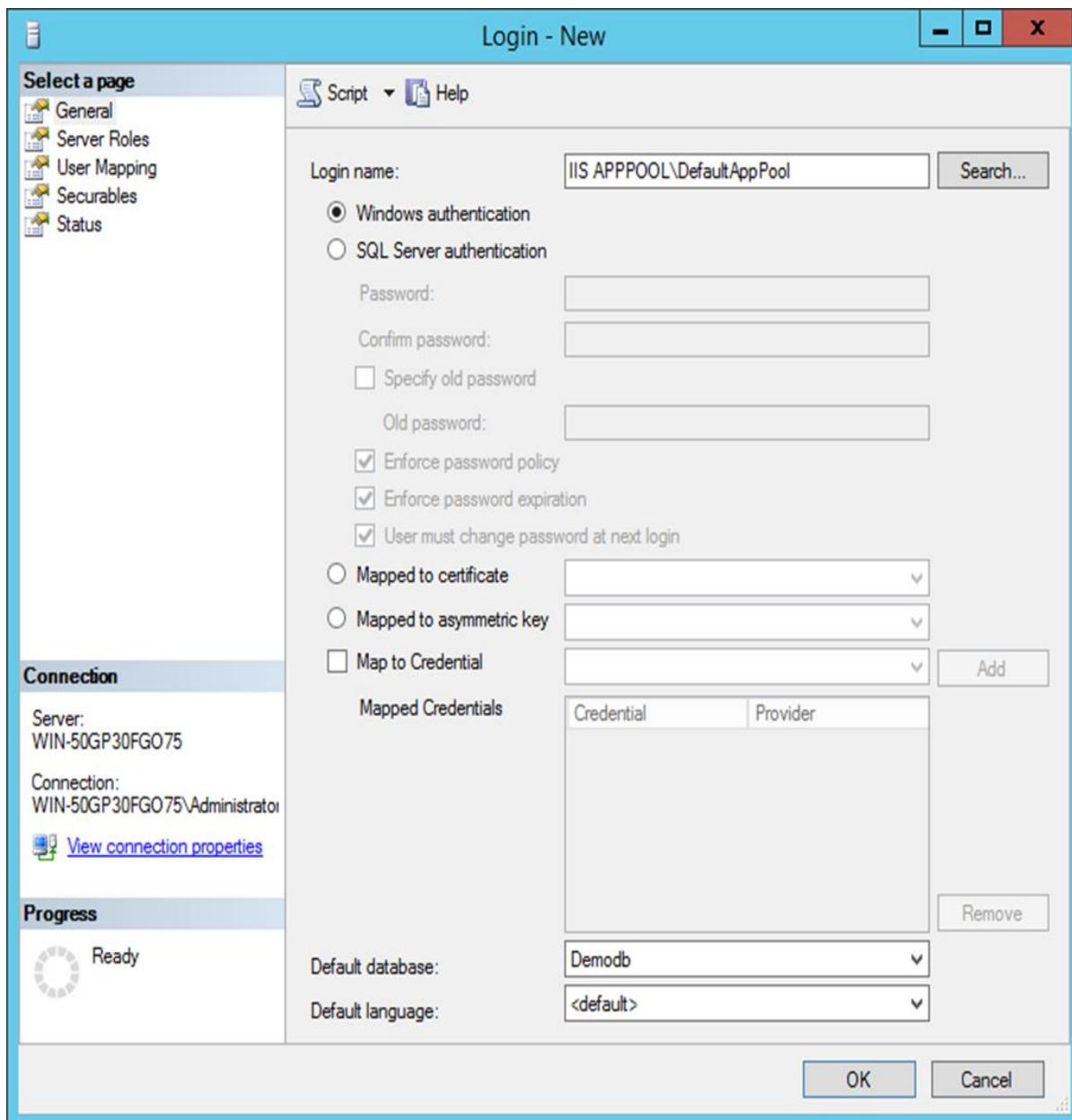
**Step 12:** If we need to connect a ASP.Net application to a MS SQL Server application, we have to give access to the default application pool to the SQL Server instance, so that it can connect to our **Demodb** database.

**Step 13:** Open SQL Server Management Studio. Go to Logins, right-click and choose the menu option **New Login**.

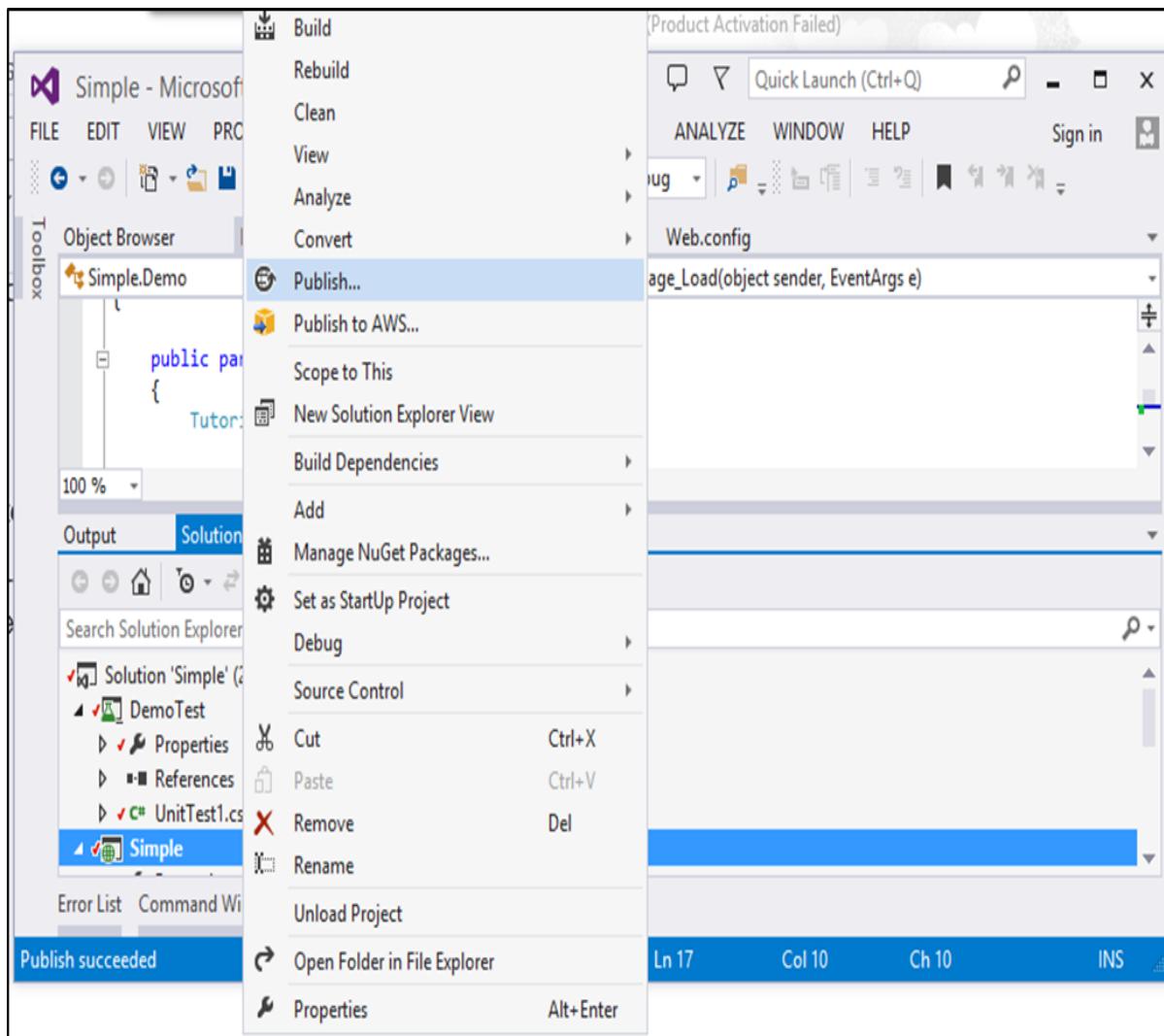


In the next screen, update the following parameters and click OK.

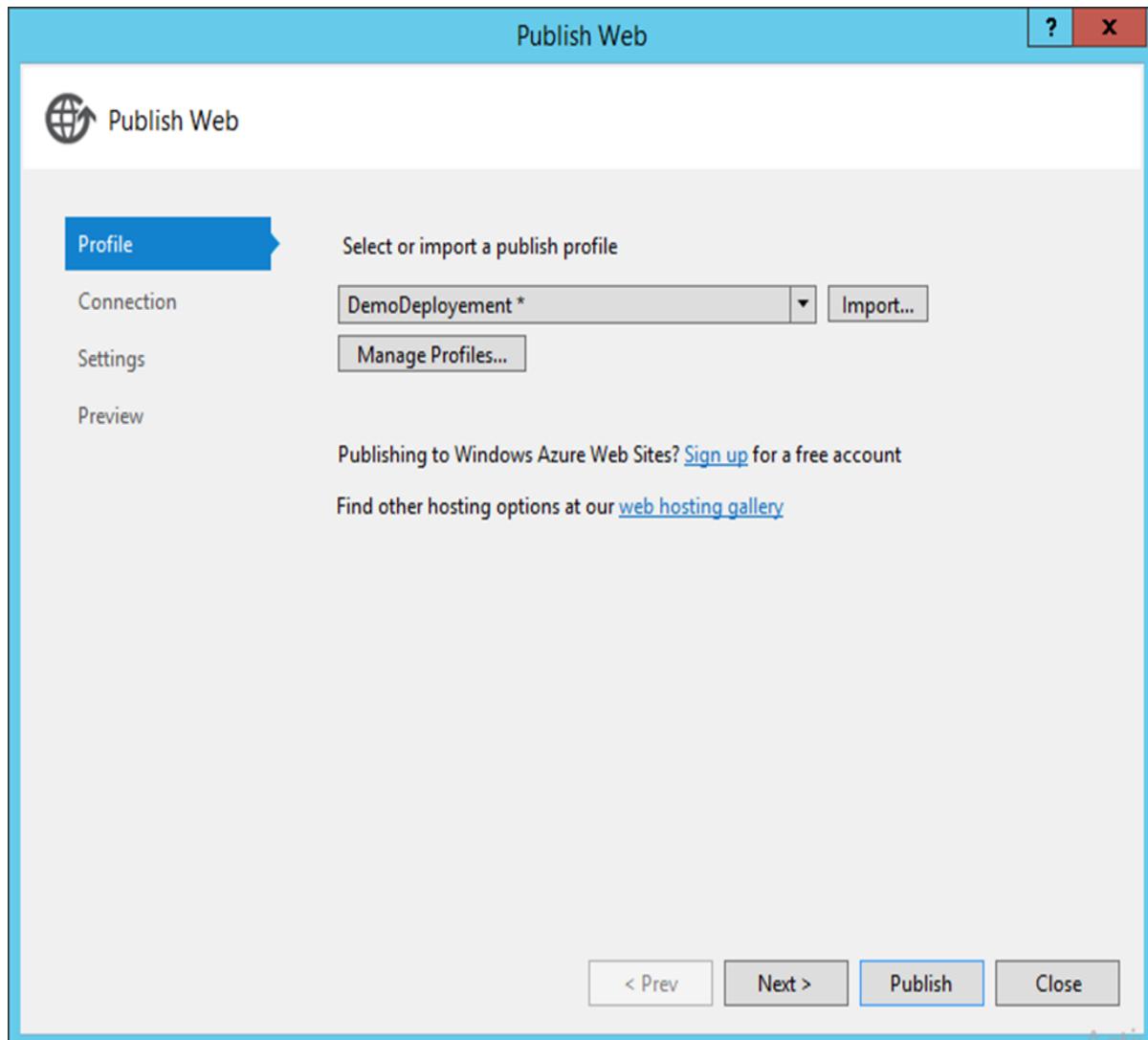
- Login name as IIS APPPOOL\DefaultAppPool.
- Default database – This should be our database, which is demodb.



**Step 14:** Creating a **Publish Profile**. The publish profile is used in Visual Studio to create a deployment package that can then be used with MS Build and in any CI Server accordingly. To do this, from Visual Studio, right-click on the project and click the menu option of Publish.



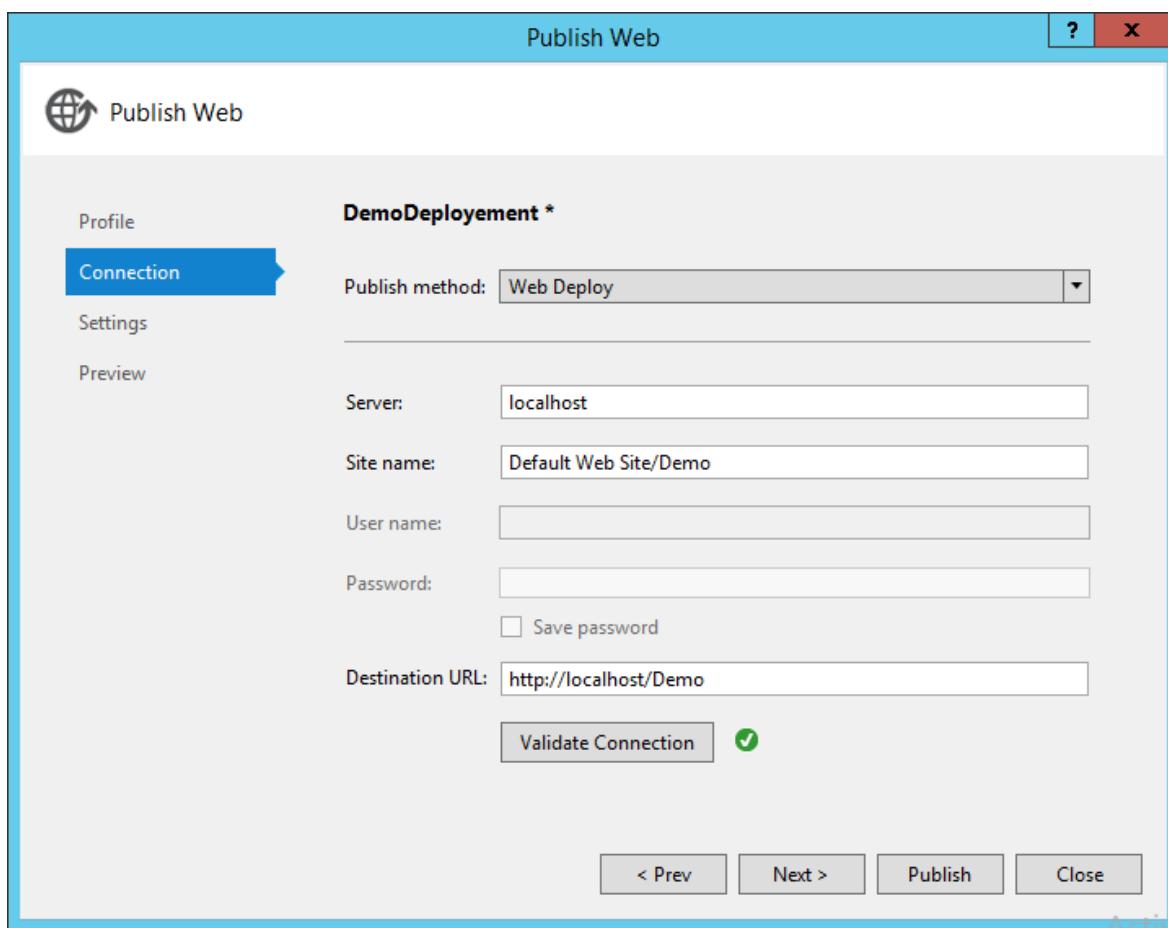
**Step 15:** In the next screen that comes up, choose to create a new Publish profile, give it a name – **DemoDeployment**. Then click the Next button.



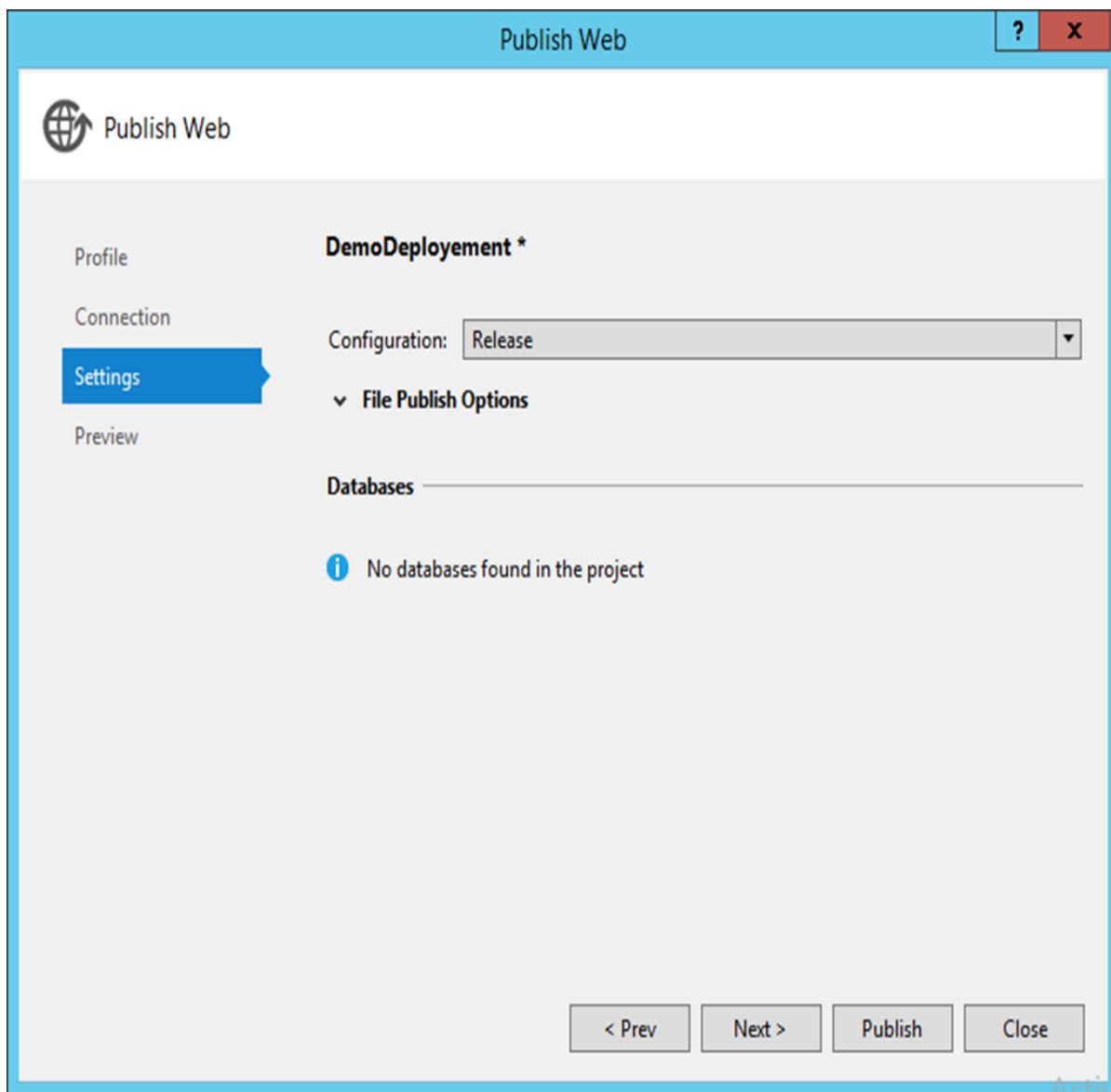
In the ensuing screen that shows up, add the following values:

- Choose the Publish method as Web Deploy.
- Enter the server as localhost.
- Enter the site name as Default Web Site/Demo.
- Put the destination url as http://localhost/Demo

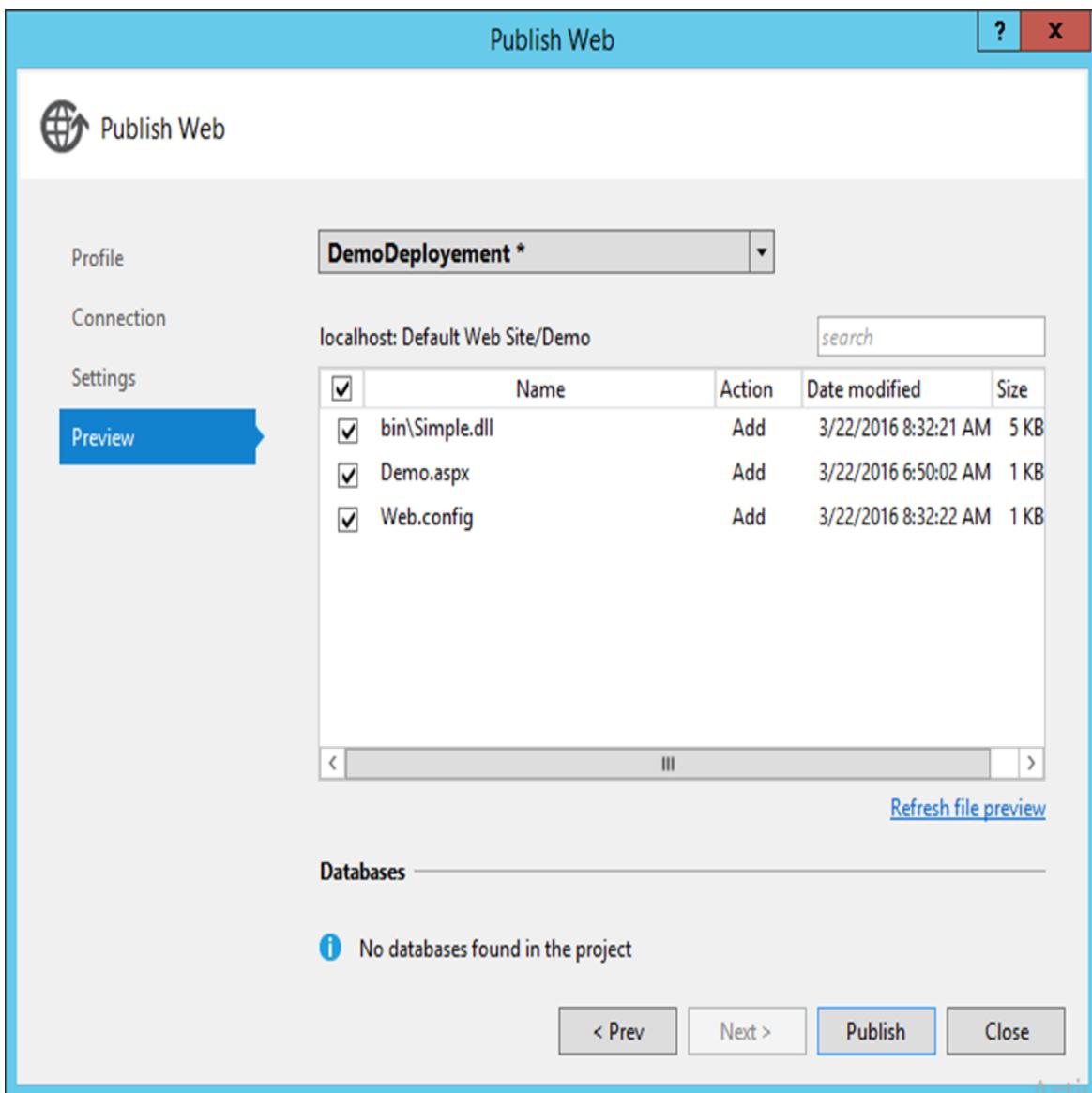
Then click the Next button.



**Step 16:** In the next screen, click Next.



**Step 17:** In the final screen that comes up, click the Publish button.



Now if you go to the **C:\Demo\Simple\Properties\PublishProfiles** location of your project, you will see a new **publish profile xml file** created. This publish profile file will have all the details required to publish your application to the local IIS server.

**Step 18:** Now let's customize our MSBuild command and use the above publish profile and see what happens. In our MSBuild command, we specify the following parameters:

- Deploy on Build is true – this will trigger an automatic deployment once a successful build is done.
- We are then mentioning to use the Publish profile which was used in the above step.
- The Visual Studio version is just to be mentioned to the MSBuild deployment capability on what is the version of the Visual Studio being used.

```
C:\Demo\Simple>msbuild simple.csproj /p:DeployOnBuild=true /p:PublishProfile=DemoDeployment /p:VisualStudioVersion=12.0
```

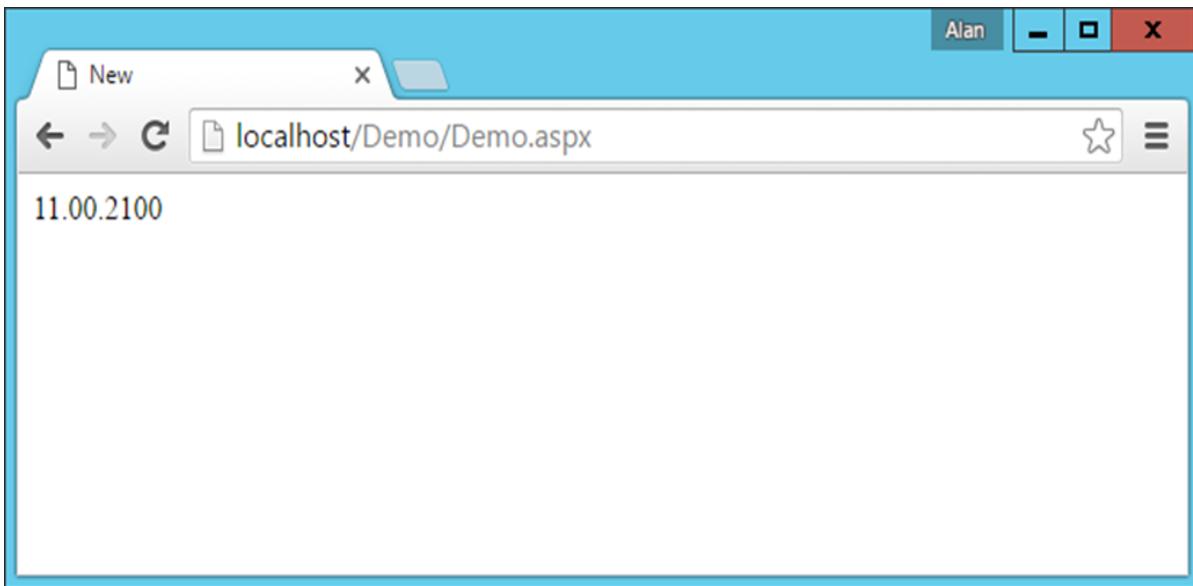
When you run the above command, MSBuild will trigger a build and deployment process. What you will note that, it is deploying it to our **Default Website** in our IIS Server.

```
Administrator: Command Prompt
GenerateMsdeployManifestFiles:
  Generate source manifest file for Web Deploy package/publish ...
MSDeployPublish:
  Starting Web deployment task from source: manifest(C:\Demo\Simple\obj\Debug\Package\Simple.SourceManifest.xml) to Destination: auto().
  Adding ACL's for path (Default Web Site/Demo)
  Adding ACL's for path (Default Web Site/Demo)
  Successfully executed Web deployment task.
  Publish Succeeded.
PipelineDeployPhase:
  Publish Pipeline Deploy Phase
Done Building Project "C:\Demo\Simple\Simple.csproj" (default targets).

Build succeeded.
  0 Warning(s)
  0 Error(s)

Time Elapsed 00:00:02.72
C:\Demo\Simple>
```

Now if we browse to the site – <http://localhost/Demo/Demo.aspx> we will see the following output, which means that the MSBuild did a successful deployment to our website.



**Step 19:** Automating through TeamCity – Now it is time to add a task to our TeamCity server to automatically use MSBuild to deploy our application, based on the above mentioned steps.

**Step 20:** Go to your project dashboard and click **Edit Configuration Settings**.

 A screenshot of the TeamCity project dashboard for 'Demo :: DemoBuild'. The top navigation bar shows 'localhost:8080/viewType.html?buildTypeId=DemoBuild'. The main content area displays the build history for 'DemoBuild'. It shows four successful builds (#18, #17, #16, #15) with green checkmarks, all completed on '22 Mar 16' with short durations (19s, 10s, 11s, 7s). The 'Pending changes' section shows 'No pending changes'. The 'Recent history' section includes a checkbox for 'Show canceled and failed to start builds'.
 

Results	Artifacts	Changes	Started	Duration	Agent	Tags
#18 Tests passed: 1	None	No changes	22 Mar 16 05:41	19s	WIN-50GP30FG075-1	None
#17 Tests passed: 1	None	No changes	22 Mar 16 05:35	10s	WIN-50GP30FG075-1	None
#16 Tests passed: 1	None	No changes	22 Mar 16 04:40	11s	WIN-50GP30FG075-1	None
#15 Success	alashro (1)		21 Mar 16 23:59	7s	WIN-50GP30FG075-1	None

**Step 21:** Go to Build Steps and click Add a Build step.

The screenshot shows the Jenkins 'Build Steps' configuration page for a build named 'DemoBuild'. The left sidebar lists various settings: General Settings, Version Control Settings (with 1 update), Build Steps (selected, with 2 steps), Triggers (1 trigger), Failure Conditions, Build Features, Dependencies, Parameters, and Agent Requirements. A note at the bottom left says 'Last edited 12 minutes ago by admin (view history)'. The main content area is titled 'Build Steps' and contains a table with two rows. The first row is for a 'Build' step using MSBuild with the target 'default'. The second row is for a 'TestStep' using VSTest with the included assembly 'DemoTest.dll'. Both rows have 'Edit' and 'More' buttons.

Build Step	Parameters Description
Build	MSBuild Build file: Simple\Simple.csproj Targets: default Execute: If all previous steps finished successfully
TestStep	Visual Studio Tests Test engine: VSTest Included assemblies: DemoTest\bin\Debug\DemoTest.dll Collect .NET code coverage data with NCover (3.x) Execute: If all previous steps finished successfully

Choose the following options –

- The runner type should be MSBuild
- Give an optional Step name
- Enter the build path as Simple/Simple.csproj
- Keep the MSBuild version as Microsoft Build Tools 2013
- Keep the MSBuild Toolsversion as 12.0
- Put the command line as /p:DeployOnBuild=true  
/p:PublishProfile=DemoDeployment /p:VisualStudioVersion=12.0

**Step 22:** Click Save.

The screenshot shows a web-based administration interface for a build configuration named 'DemoBuild'. The URL in the address bar is `localhost:8080/admin/editRunType.html?id=buildType:DemoBuild&runnerId=RUNNER_1`. The page title is 'DemoBuild Configuration'. The navigation bar includes 'Projects', 'Changes', 'Agents 1', 'Build Queue 0', 'admin | Administr', and 'Actions'. The main content area shows the 'Administration > <Root project> > Demo > DemoBuild' path. On the left, a sidebar lists 'Build Configuration Settings' (General Settings, Version Control Settings), 'Build Step: MSBuild' (Triggers 1, Failure Conditions, Build Features, Dependencies, Parameters, Agent Requirements), and a note 'Last edited one minute ago by admin (view history)'. The right side displays the 'Build Step (4 of 4): Deploy' configuration with the following settings:

Runner type:	MSBuild
Step name:	Deploy
Build file path:	Simple/Simple.csproj
MSBuild version:	Microsoft Build Tools 2013
MSBuild ToolsVersion:	12.0
Run platform:	x86
Targets:	(empty)

Last edited one minute ago by admin (view history)

Agent Requirements

MSBuild version: Microsoft Build Tools 2013

MSBuild ToolsVersion: 12.0

Run platform: x86

Targets:

Enter targets separated by space or semicolon.

Command line parameters:

```
/p:DeployOnBuild=true  
/p:PublishProfile=DemoDeployment  
/p:VisualStudioVersion=12.0
```

Enter additional command line parameters to MSBuild.exe.

.NET Coverage

.NET Coverage tool: <No .NET Coverage>

Choose a .NET coverage tool.

⚠ Test code coverage is supported only for NUnit tests run using TeamCity facilities.

Show advanced options

Save Cancel

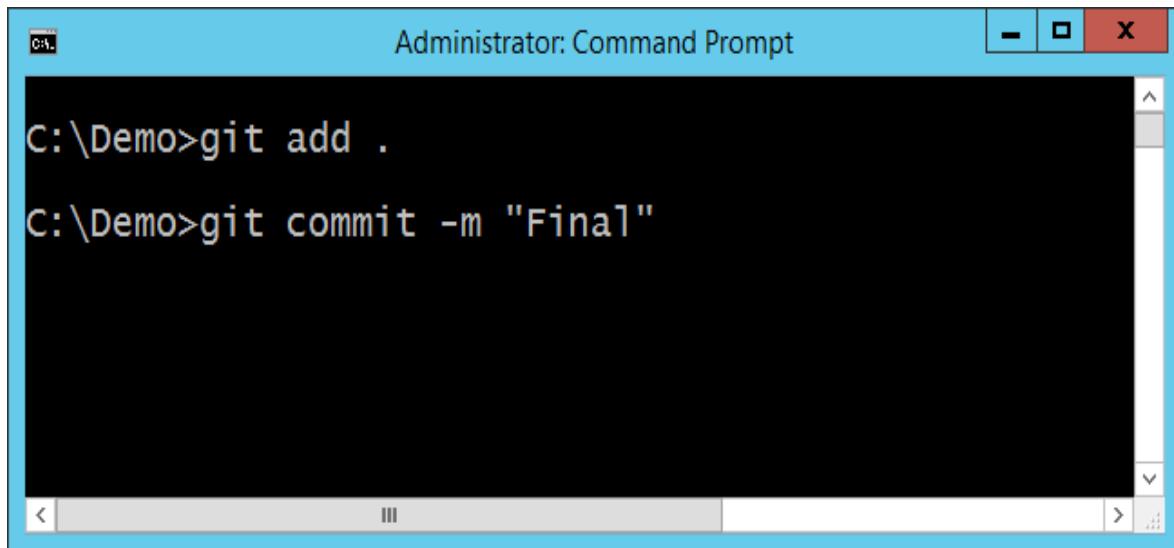
Make sure that in the build steps, the Deploy step is the last step in the chain.

The screenshot shows the 'DemoBuild Configuration' page in TeamCity. The 'Build Steps' tab is selected, displaying four steps: Databasesetup, Build, TestStep, and Deploy. A yellow bar at the top right indicates 'Build step enabled.'

Build Step	Parameters Description	Edit	More
Databasesetup	Command Line Command: C:\Program Files\Microsoft SQL Server\110\Tools\Binn\sqlcmd.exe -S WIN-50GP30FG075 -i Sample.sql Execute: If all previous steps finished successfully	Edit	More
Build	MSBuild Build file: Simple\Simple.csproj Targets: default Execute: If all previous steps finished successfully	Edit	More
TestStep	Visual Studio Tests Test engine: VSTest Included assemblies: DemoTest\bin\Debug\DemoTest.dll Collect .NET code coverage data with NCover (3.x) Execute: If all previous steps finished successfully	Edit	More
Deploy	MSBuild Build file: Simple\Simple.csproj Targets: default Execute: If all previous steps finished successfully	Edit	More

At the bottom left, a note says 'Last edited moments ago by admin (view history)'. At the bottom, there are links for Help, Feedback, TeamCity Professional 9.1.6 (build 37459), and License agreement.

**Step 23:** Now let's do a final **git commit**, to ensure all the files are in Git and can be used by TeamCity.



The screenshot shows an 'Administrator: Command Prompt' window. The command line displays two commands:  
C:\Demo>git add .  
C:\Demo>git commit -m "Final"

Congratulations, you have successfully set up a complete Continuous Integration Cycle for your application, which can be run at any point in time.

# 19. CI – Best Practices

Let's have a final review of the best practices of Continuous Integration based on all the lessons we have learnt so far –

- **Maintain a code repository** – This is the most basic step. In all our examples, everything is maintained in a Git repository right from the code base to the Publish profiles, to the database scripts. It must always be ensured that everything is kept in the code repository.
- **Automate the build** – We have seen how to use MSBuild to automate a build along with using a publish profile. This is again a key step in the continuous Integration process.
- **Make the build self-testing** – Ensure that you can test the build by keeping unit test cases in place and these test cases should be in such a way that it can be run by the Continuous Integration server.
- **Everyone commits to the baseline every day** – This is a key principle of Continuous Integration. There is no point staying till the end of the entire process to see who breaks the build.
- **Every commit (to baseline) should be built** – Every commit made to the application, needs to be successfully built. If the build fails for whatever reason, then the code needs to be changed to ensure the build passes.
- **Keep the build fast** – If the build is slow, then it would indicate a problem in the entire Continuous Integration process. Ensure that the builds are always limited to a duration, preferably should never go beyond 10 minutes.
- **Everyone can see the results of the latest build** – The TeamCity dashboard gives everyone a view of all the builds, which have either passed or failed. This gives a good insight to all the people who are involved in the Continuous Integration process.