Microsoft Team Foundation Server 2015 Cookbook

Team Foundation Server (TFS) allows you to manage code repositories, build processes, test infrastructure, and deploy labs. TFS supports your Team, enabling you to connect, collaborate, and deliver on time. Microsoft's approach to Application Lifecycle Management (ALM) provides a flexible and agile environment that adapts to the needs of your Team, removing barriers between roles, and streamlines processes.

The book introduces you to creating and setting up Team Projects for Scrum Teams. You’ll explore various source control repositories, branching, and merging activities, along with a demonstration of how to embed quality into every code check-in. Then, you’ll discover Agile project planning and management tools. Later, emphasis is given to the testing and release management features of TFS which facilitate the automation of the release pipeline in order to create potentially shippable increments. By the end of the book, you’ll have learned to extend and customize TFS plugins to incorporate them into other platforms and enable Teams to manage the software life cycle effectively.

Who this book is written for

This book is aimed at software professionals including developers, testers, architects, configuration analysts, and release managers who want to understand the capabilities of TFS to deliver better quality software faster. A working setup of TFS 2015 and some familiarity with the concepts of software lifecycle management is assumed.

What you will learn from this book

- Creating a Team Project with Dashboards, assigning license, adding users, and auditing access
- Setting up a Git repository in an existing TFVC-based Team Project
- Setting up branch policies and conducting Pull requests with code reviews
- Mapping, assigning, and tracking Work Items shared by multiple Teams
- Setting up and customizing Backlogs, Kanban board, Sprint Task board, and Dashboards
- Creating a continuous integration, continuous build, and release pipeline
- Integrating SonarQube with TFBuild to manage technical debt
- Triggering Selenium Web Tests on a Selenium Test Grid using TFBuild
- Using Visual Studio Team Services Cloud load testing capability with new build framework
- Extending and customizing the capabilities of Team Foundation Server using API and Process Editor

In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 5 'Testing Your Application'
- A synopsis of the book’s content
- More information on Microsoft Team Foundation Server 2015 Cookbook
About the Author

Tarun Arora is obsessed with high-quality working software, continuous delivery, and Agile practices. He has experience managing technical programs, implementing digital strategy, and delivering quality at scale. Tarun has worked on various industry-leading programs for fortune 500 companies in the financial and energy sector.

He is one of the many geeks working for Avanade in the United Kingdom. Avanade helps clients and their customers realize results in a digital world through business technology solutions, cloud, and managed services that combine insight, innovation, and expertise in Microsoft technologies.

For the past 5 years, Tarun has been a Microsoft Most Valuable Professional in Visual Studio and Development Technologies. His core strengths are enterprise architecture, .NET, WPF, SQL, and PowerShell. He was awarded the MVP of the year award by Microsoft in 2014 for going over and above in supporting the product teams and the community with his contributions. He is also an ALM Ranger and has contributed to key guidance and tooling projects focused on Azure, Team Foundation Server, Visual Studio Team Services, and Visual Studio Extensibility. Tarun is an active open source community contributor, speaker, and blogger. Follow him on twitter at @arora_tarun and his blog at http://www.visualstudiogeeks.com for the latest and greatest in technology trends and solutions on DevOps and ALM.

Tarun loves photography and travel. He is a very active traveler and has travelled to more than 21 countries in the last few months. Parts of this book have been written on his journeys across three continents. While some chapters were written on the beaches of Mauritius, others were written in transit, airport lounges, and taxis. Follow his adventures on his travel blog (https://outofofficetraveller.wordpress.com).
Preface

Visual Studio is a suite of Microsoft Developer Tools and Services, a few key ones being Visual Studio IDE, Visual Studio Code, Visual Studio Team Services, and Visual Studio Team Foundation Server (TFS). Back in November 2004, Microsoft released its first version of integrated Application Lifecycle Management (ALM) tool, called "Microsoft Visual Studio Team Systems". Over the last 15 years, the product has gone through several evolutions, each enriching the developer experience and the scope of tooling:

Visual Studio – Any App Any Developer

The Visual Studio family of tools and services now enables heterogeneous software development across various platforms. The experience of using open source tooling within the product has improved tremendously. Open source solutions are being given first class citizen status, and more of these solutions are being pre-packaged into the product. This gives us a clear indication that Microsoft wants to become the platform of choice for every developer, independent of the technology or platform. There is a huge overlap between the tools and services within the Visual Studio family of tools. This book focuses entirely on Visual Studio Team Foundation Server 2015.

Microsoft Visual Studio Team Foundation Server 2015 is at the center of Microsoft's ALM solution, providing core services such as version control, Work Item tracking, reporting, and automated builds. TFS helps organizations communicate and collaborate more effectively throughout the process of designing, building, testing, and deploying software, ultimately leading to increased productivity and team output, improved quality, and greater visibility of an application's life cycle.
Software delivery itself has gone through a revolution in the last decade. The introduction of Agile practices and lean frameworks, such as Scrum, Kanban, XP, and RUP, among others, have demonstrated that iterative feedback-driven development helps to cope with changes in the marketplace, business, and user requirements. Lean processes also help minimize waste and maximize value delivery to end users. Better DevOps practices encouraging continuous integration, continuous deployment, continuous delivery, and continuous feedback along with better tooling are enabling organizations to break the silos between teams. Mission-critical applications may still choose to deliver using Waterfall, while a line of business applications may find more success choosing lean methodologies. There is no right or wrong in this; choose the process and tools that are most appropriate to your delivery scenario. Visual Studio TFS supports most processes out of the box, and gives you the flexibility to customize and define processes that work best for your organization.

Visual Studio Team Foundation Server 2015, henceforth referred to as TFS in this book, is Microsoft’s on-premise offering of ALM Tooling. Microsoft also offers a cloud-hosted service called Visual Studio Team Services (VSTS). Do not confuse VSTS for being Visual Studio IDE in the cloud; it is instead a collection of developer services comparable to TFS that run on Microsoft Azure and extend the development experience in the cloud. Microsoft is really committed to its hosted service, and has moved it into a 3-week cadence. All features are released in VSTS first, and then, most features are rolled into TFS via quarterly updates. A timeline of features released and those planned in the future releases can be found at https://www.visualstudio.com/en-us/news/release-archive-vso.aspx. The product teams solicit new feature requests via user voice. If you have a burning idea for a feature, be sure to log your request at https://visualstudio.uservoice.com/forums/121579-visual-studio-2015/category/30925-team-foundation-server-visual-studio-online. VSTS now offers a lot of enterprise features such as guaranteed uptime, single sign on using ADFS and AAD, and compliance to US, European, and Australian data sovereignty laws by offering tenants hosted in those regions. Though VSTS boasts of having over 3 million active users, organizations that need more control of the environment and their data will still prefer TFS over VSTS.

All recipes in this book are designed for TFS; however, because of the overlap between VSTS and TFS, most of what you learn in this book is applicable to VSTS.
The various clients that can be used to connect to TFS can be broadly divided into two
groups—primary clients, and task-specific clients, as shown in the following screenshot. A full
list of the functions that can be performed using these clients can be found at https://

If you're setting up TFS for personal use, or to evaluate the core features, such as version
control, build, and Work Item tracking, use TFS Express. It's free, it's simple to set up, and it can
be installed on both client and server operating systems. Express does not support integration
with SharePoint or Reporting Services. If you are setting up TFS for your organization, use the
standard version of TFS. You can set up TFS on a single computer, in a dual server configuration,
or in a multi-server configuration. Use the following handy reference to check the compatibility
matrix for TFS 2015:

https://msdn.microsoft.com/Library/vs/alm/TFS/administer/requirements

The TFS architecture setup and network and port requirements can be found at
https://msdn.microsoft.com/en-us/library/ma252473(v=vs.120).aspx. The product setup documentation can be found at https://msdn.microsoft.com/en-us/Library/vs/alm/TFS/setup/overview. The planning and disaster recovery guidance (http://vsarplanningguide.codeplex.com/) from ALM Rangers is very useful when planning an enterprise grade TFS setup. TFS 2015-specific license updates will be covered
in the Assigning a license, adding users, and auditing user access recipe in Chapter 1, Team

The recipes in this book require a standard one machine setup of TFS. You can set up a
standalone single server using the preceding installation instructions, or, alternatively, use
a preconfigured TFS 2015 Virtual Machine. Instructions to download and set this up can be
found at http://vsalmvm.azurewebsites.net/.
What this book covers

Chapter 1, Team Project Setup, covers Team Project, which is a logical container isolating all tools and artifacts associated with a software application together in a single namespace. Features such as Welcome pages, Dashboards, Team Rooms, and many more enable better collaboration within Teams, whereas the ability to rename Team Projects and scripting Team Project creation empowers you to better administer a Team Project. In this chapter, we'll learn the different features of a Team Project and how to set up these features to leverage them to their full potential.

Chapter 2, Setting Up and Managing Code Repositories, introduces TFS, which is the only product to offer a centralized as well as distributed version control system. In this chapter, we'll learn how to set up both TFVC and Git repositories in a single project and how to tackle technical debt by enforcing code reviews and code analysis into the development workflows.

Chapter 3, Planning and Tracking Work, explains the requirements that are implemented but never used, or those that are used just long enough to identify that they don't satisfy the needs of the users cause and waste, re-work, and dissatisfaction. In this chapter, we'll learn how to set up and customize multiple backlogs, Kanban, and Sprint Task Board. We'll also learn how to integrate with external planning tools using Service Hooks, and how to improve the feedback loop by leveraging the feedback features in TFS.

Chapter 4, Building Your Application, introduces the new build system (TFBuild), which is a cross platform, open, and extensible task-based execution system with a rich web interface that allows the authoring, queuing, and monitoring of builds. In this chapter, we'll learn how to set up and use TFBuild for continuous integration. We'll also learn how to integrate TFBuild with SonarQube and GitHub. We'll also review features that help lay the foundations for continuous delivery of software.

Chapter 5, Testing Your Application, states that low quality software just isn't acceptable. But you may ask "what is the right level of quality?" In this chapter, we'll learn how to plan, track, and automate using the testing tools available in TFS. We'll also learn how to leverage the new build system to integrate non-Microsoft testing frameworks, such as Selenium and NUnit, into the automation testing workflows.

Chapter 6, Releasing Your Application, explains the new web-based Release Manager in TFS that uses the same agent and task infrastructure offered by TFBuild. In this chapter, we'll learn how to set up, secure, and deploy to multiple environments using release pipelines. We'll also learn how to track and report on releases delivered through the release pipeline. The techniques in this chapter enable you to set up your software for continuous delivery.

Chapter 7, Managing Team Foundation Server, teaches you how to update, maintain, and optimize your TFS, enabling high availability for geo-distributed Teams and reducing the administration overheads.
Chapter 8, *Extending and Customizing Team Foundation Server*, explains that it is not uncommon for organizations to have different tools to manage different parts of the life cycle, for example, Jira for Agile project management, TeamCity for builds, Jenkins for release management, and ServiceNow for service management. In this chapter, we’ll learn about the TFS object model and TFS REST APIs to programmatically access and integrate with systems. In this chapter, we’ll also cover how to customize Team Projects by leveraging Process Template customization.
Testing Your Application

"Walking on water and developing software from a specification are easy...If both are frozen!"

-- Edward Berard

In this chapter, we will cover the following:

- Running NUnit tests in the CI Pipeline using TFBuild
- Creating and setting up a Machine Group
- Deploying a test agent through the TFBuild task
- Distributing test execution on a Lab Machine Group
- Triggering Selenium Web Tests on a Selenium Test Grid using TFBuild
- Integrating the Cloud Load Testing Service in TFBuild
- Analyzing test execution results from the Runs view
- Exporting and importing test cases in Excel from TFS
- Copying and cloning test suites and test cases
- Exporting test artifacts and test results from the test hub
- Charting testing status on Dashboards in Team Portal

Introduction

Software teams are constantly under pressure to deliver more... faster. End users expect software to simply work. Low quality software just isn't acceptable. But, you may ask what the right level of quality is? Quality is a very subjective term; it is therefore important for Teams to agree to a definition of quality for their software. Teams that are unable to define quality usually end up testing for coverage rather than testing for quality.
Testing Your Application

The toolkit in Team Foundation Server provides tooling for both manual and automation testing. Microsoft Test Manager (MTM), first introduced with TFS 2010, enables testers to plan, track, and run manual, exploratory, and automated tests. While Test Manager fully integrates with TFS, it does not offer any integration with other testing platforms. The Test Manager architecture does not lend itself to extensibility. Microsoft has ambitions to support every developer and every app; however, it isn't possible with tooling that can't be run on non-window platforms. The test tooling is gradually moving out from the Test Manager client into the web-based Team Web Portal to enable extensibility, cross-platform availability, and integration with other testing platforms. The following screenshot provides a comparison of testing features already in test hub, those that will gradually move, and those that will remain in MTM. For a full comparison of all the features, download Microsoft Test Manager Vs Test Hub Feature Comparison.png from the course material:

<table>
<thead>
<tr>
<th>Test Plan Management</th>
<th>TH</th>
<th>MTM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create/Edit Test Plans</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Assign Build</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Create/Edit/Assign Config</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Assign Environments</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Assign Test Settings</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Clone Test Plan</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Explore Test Plan</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test Suite Management</td>
<td>TH</td>
<td>MTM</td>
</tr>
<tr>
<td>Create/Edit Test Suites</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Add/Remove Tests</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Assign Individual Testers</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Assign configurations</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add tests from other suites</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Clone Suites</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Export Test Suites</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

To speed up the software delivery loop, software testing needs to be incorporated into the Continuous Integration (CI) Pipeline. In order to do this, software testing needs to shift left in the development processes. Test-driven development (TDD) enables developers to write code that's maintainable, flexible, and easily extensible. Code backed by unit tests helps identify change impact and empowers developers to make changes confidently. In addition to this, functional testing needs to be automated. This enables software testers to focus on high-value exploratory testing rather than just coverage of test matrix.

The DevOps movement at large supports bringing testing into the CI Pipeline. The tooling in TFS has evolved to enable this. The pre 2015 version of TFS shipped with a separate test controller and test agent, with the introduction of the new build framework in TFS 2015 test runner has been distilled down to a task in the build definition. TFS now provides a task based open and extensible build framework that allows you to pick not only a build framework of your choice, but also provides you full autonomy to pick and assemble the test frameworks of your choice too. The new build system enables the agent to be a generic task orchestrator. The agent is capable of executing the entire CI Pipeline. This new approach removes the need for managing multiple single purpose agents. This takes away the friction from the setup process and enables a scale up option without the need for tedious configuration. This also enables integrating other testing frameworks and platforms into the same pipeline:
Chapter 5

Through the recipes in this chapter, you'll learn how to leverage tasks in the build definition to provision test agents on demand, perform a distributed test execution that includes a large number of diverse automated tests such as unit, functional UI, Selenium and, coded UI on remote Machine Groups. We'll also look at integrating with Visual Studio Team Systems to run cloud load tests. Last but not least, we will look at the rich test reports and test result analysis from the test executions.

Look at the richness of testing features within the bigger DevOps capabilities available in TFS; should you decide to move from your existing Test Case Management Solutions to the Microsoft Test Manager, you may wonder what the possibilities for migration are. We'll briefly cover a few tools that are available today to help you with the migration. In the end, we'll look at how to export and visualize test execution results with the help of light weight charts.

Along with providing world-class tooling for manual and automation testers, TFS also provides rich, integrated test tooling to stakeholders. The web-based testing extension allows you to perform exploratory testing of your web or mobile application right from the browser on any platform (Windows, Mac, or Linux). Your stakeholders now do not need to follow predefined test cases or test steps. Stakeholders can capture and annotate screenshots, log bugs, and share notes. The testing session records and logs every step providing developers rich action logs for diagnosing application and user experience issues. Read more on this at http://bit.ly/1I82pfK.

These investments are taking the testing tooling from good to great. It is a very exciting time to be in the testing space!

**Running NUnit tests in the CI Pipeline using TFBuild**

Traditionally, developers using the NUnit framework had to install the NUnit Test Adapter on the build machines. While this approach worked if you were dealing with a small number of build servers, it quickly became tedious when dealing with large number of build servers. An alternative to installing the NUnit adapter was to inject the NUnit adapter DLLs to the build machines using the custom assembly field available in the build controller properties. While this worked with TFVC-based repositories, there were challenges using this approach for Git-based repositories. This is a classic example of configuration hell when using non-Microsoft testing frameworks.
Testing Your Application

The new build framework makes running non-Microsoft unit test frameworks completely configuration free. In this recipe, you'll learn how to use the Visual Studio Test task in the build definition to run NUnit tests or generally any non-Microsoft unit testing frameworks.

Getting ready

In order to create a build definition, you'll need to be a member of the Project Build Administrator Group.

How to do it...

1. Launch Team Explorer and connect to the FabrikamTFVC Team Project. From the Source Control Explorer, open the FabrikamFiber.CallCenter.sln in Visual Studio:

2. Follow the instructions at http://bit.ly/1XgG0UG to create a new NUnit test project. Call the project FabrikamFiber.Web.NUnit.Tests.csproj.

3. Right-click on the FabrikamFiber.Web.NUnit.Tests.csproj project and select Manage NuGet Packages... from the context menu:

4. From the NuGet package manager, search and install the NUnit test adapter.
5. Open the folder location of the solution and you’ll notice that the NUnitTestAdapter NuGet package has been added under the packages folder:

![Folder Location Image]

6. Check in the code changes from the Pending Changes view in Team Explorer:

![Pending Changes Image]

7. In the FabrikamTFVC Team Web Portal, navigate to the BUILD hub and click on the + icon to add a new build definition. Create it using Visual Studio Build Template:

![Build Definition Template]

8. In the Visual Studio Build Task, ensure that the Restore NuGet Packages flag is checked. You don’t need to make any other changes in the build or test task. Ensure that the code repository is mapped as FabrikamTFVC in the repository tab. Save the build definition as FabrikamTFVC NUnit CI. You can learn more about setting up a build definition in the *Creating a continuous integration build definition in TFSBuild* recipe in *Chapter 4, Building Your Application*. 
Testing Your Application

9. Queue a build. Once the build completes, you will see a summary of the test results in the build summary section. The build successfully executes all NUnit tests without any manual configuration specific to NUnit:

![Test results](image)

**How it works**

Let's now understand how TFBuild manages to identify the test runner for NUnit tests. Open the FabrikamTFVC NUnit CI build definition. In the Advanced section of the Visual Studio Test task, you'll see a field for specifying the path to custom adapters:

![Advanced](image)

With the restore NuGet packages flag checked in the Visual Studio Build task, all NuGet packages that the solution has dependency on are downloaded into the agent's working directory. The Visual Studio Test task scans the agent working directory for all test adapter packages. In this instance, the NUnit Test Adapter NuGet package was checked into source control. The package would have been restored into the agent working directory during the package restore process; as a result the Visual Studio Test task loaded up the NUnit Test Adapter during the scan.

TFBuild publishes the test and coverage results into TFS independent of the testing framework used. This is automatically handled by the TFBuild framework.

This approach scales really well, as you add more machines to carry out testing, you do not need to worry about manually adding the NUnit Test Adapter on the test machines. This also simplifies upgrading test adapters. The test task can load multiple versions of one test adapter and use the adapter referenced by the unit tests.
The **Path to Custom Test Adapters** field wasn't really put to use in this case. However, this field is useful for specifying the test adapter path if your solution uses a unit test framework that does not have a NuGet package associated with it. The path to the test adapter needs to be constructed using the build agent's source directory variable. The `Build.SourceDirectory` variable points to the directory in the build agent where the source code has been synced. If your custom test adapter was in the `\FabrikamTFVC\myAdapters` folder, you would need to specify the path: `$(Build.SourcesDirectory)\src\myAdapters\TestAdapter.1.2`.

---

## Creating and setting up a Machine Group

Simply put, Machine Group is a logical grouping of machines. The Machine Group holds metadata, connectivity, and login details of the machines in the group. Machine Group can directly be referenced from build and release definitions. In this recipe, you'll learn how to create and set up a Machine Group.

### Getting ready

**Scenario:** The FabrikamTFVC Team has a lab environment in the Fabrikam.lab domain. Fabrikam.lab comprises of five servers that serve different roles. The FabrikamTFVC Team wants the ability to directly reference these machines from the build definition and release definition to deploy test agents on all the machines and trigger a distributed test run. Fabrikam.Lab is managed by the Fabrikam Environments Team who cannot share environment credentials with the FabrikamTFVC Team. In this recipe, we'll walk through the process followed by the Fabrikam Environments Team to set up and configure the Machine Group Fabrikam-QA for the FabrikamTFVC Team:

---

![Diagram showing five servers and their IP addresses and tags]

- **Machine Group:** Fabrikam-QA
- **Domain:** Fabrikam.lab
- **Administration Credentials**
  - **Username:** FabLab-Admin
  - **Password:** xxxxxxxxxxxxx

- **Server 1**
  - **FQDN:** QA-Web1.Fabrikam.lab
  - **Tags:** Win12R2, WebServer, QA

- **Server 2**
  - **FQDN:** QA-Web2.Fabrikam.lab
  - **Tags:** Win12R2, WebServer, QA

- **Server 3**
  - **FQDN:** QA-Db1.Fabrikam.lab
  - **Tags:** Win12R2, DbServer, QA

- **Server 4**
  - **FQDN:** QA-App1.Fabrikam.lab
  - **Tags:** Win12R2, AppServer, QA

- **Server 5**
  - **FQDN:** QA-App2.Fabrikam.lab
  - **Tags:** Win12R2, AppServer, QA
The Machine Group will be accessed by a remote host; the remote host will likely be playing the role of a build agent or release agent. As illustrated in the following figure, the remote host is in the same network as the Machine Group and has a trust relationship with the Machine Group:

The build agent uses Windows PowerShell remoting that requires the Windows Remote Management (WinRM) protocol to connect to the machines in the Machine Groups. WinRM needs to be enabled on a machine as a prerequisite before it can be added into the Machine Group. In this case, Kerberos will be used as the mode of authentication since the agent and Machine Group are in the same corp network.

<table>
<thead>
<tr>
<th>Target Machine state</th>
<th>Target Machine trust with automation agent</th>
<th>Machine Identity</th>
<th>Auth Account</th>
<th>Auth Mode</th>
<th>Auth Account permission on target machine</th>
<th>Conn Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain-joined machine in the corp network</td>
<td>Trusted</td>
<td>DNS name</td>
<td>Domain account</td>
<td>Kerberos</td>
<td>Machine admin</td>
<td>WinRM HTTP</td>
</tr>
</tbody>
</table>

In the next few steps, we'll walk through how to configure WinRM on a machine, and you'll learn how to test connectivity through WinRM:

1. PowerShell 2.0 and Windows Management Framework 4.0 (http://bit.ly/1kNlxuW) are required to be installed on both the agent and machines in the Machine Group.

2. Log into the QA-Web1.Farbikam.lab machine, start Windows PowerShell as an administrator by right-clicking on the Windows PowerShell shortcut and selecting Run as Administrator.
3. By default, the WinRM service is configured for manual startup and stopped. Executing the `winrmquickconfig -q` command performs a series of actions:
   1. Starts the WinRM service.
   2. Sets the startup type on the WinRM service to **Automatic**.
   3. Creates a listener to accept requests on any IP address.
   4. Enables a firewall exception for WS-Management communications.
   5. Registers the `Microsoft.PowerShell` and `Microsoft.PowerShell.Workflow` session configurations, if they are not already registered.
   6. Registers the `Microsoft.PowerShell32` session configuration on 64-bit computers, if it is not already registered.
   7. Enables all session configurations.
   8. Changes the security descriptor of all session configurations to allow remote access.
   9. Restarts the WinRM service to make the preceding changes effective.

4. The next few commands will prepare WinRM for Kerberos authentication.

5. Increase the maximum memory allocation per session:
   ```
   winrm set winrm/config/winrs '@{MaxMemoryPerShellMB="300"}'
   ```

6. Next, increase the session timeout period:
   ```
   winrm set winrm/config '@{MaxTimeoutms="1800000"}'
   ```

7. Allow the traffic between agent and Machine Group to be unencrypted:
   ```
   winrm set winrm/config/service '@{AllowUnencrypted="true"}''
   ```

8. Disable basic authentication:
   ```
   winrm set winrm/config/service/auth '@{Basic="false"}''
   ```

9. Setup a firewall exception to allow inbound traffic on port 5985; this is the default port used by WinRM when using HTTP:
   ```
   netshadvfirewall firewall set rule name="Windows Remote Management (HTTP-In)" profile=public protocol=tcp localport=5985 new remoteip=any
   ```

10. Disable digest for client authentication:
    ```
        winrm set winrm/config/client/auth '@{Digest="false"}''
    ```

11. Set service authentication to use Kerberos:
    ```
        winrm set winrm/config/service/auth '@{Kerberos="true"}''
    ```

12. Trust all connections between agent and Machine Group:
    ```
        winrm set winrm/config/client '@{TrustedHosts="*"}''
        Set-Item WSM:\localhost\Client\TrustedHosts *
    ```
Testing Your Application

13. Restart the `winrm` service:
   
   ```
   Restart-Service winrm-Force
   ```

14. To ensure Kerberos authentication is enabled on WinRM, run the following command:
   
   ```
   winrm get winrm/config/service/auth
   ```

   ```
   PS C:\Users\tarun> winrm get winrm/config/service/auth
   Auth
   Basic = false
   Kerberos = true
   Negotiate = true
   Certificate = false
   CredSSP = false
   CbtHardeningLevel = Relaxed
   ```

15. Now, let's validate whether WinRM has correctly been set up on QA-Web1.Fabrikam.lab. Log into another VM in the lab, in this case QA-Web2.Fabrikam.lab. Launch PowerShell as an administrator by right-clicking on the Windows PowerShell shortcut and selecting Run as administrator. Execute the following command:
   
   ```
   Test-Wsman -computerName QA-Web1.Fabrikam.lab
   ```

   ```
   wsmanid : http://schemas.dmtf.org/wbem/wsman/identity/1/wsmanidentity.xsd
   ProductVendor : Microsoft Corporation
   ProductVersion : 0.1.0.0 SP: 0.0 Stack: 2.0
   ```

16. Execute the following command to check the port WinRM is listing on:
   
   ```
   winrm e winrm/config/listener
   ```

   ```
   PS C:\Users\tarun\Desktop> winrm e winrm/config/listener
   Address = *
   Transport = HTTP
   Port = 5985
   Hostname
   Enabled = true
   UserProfile = wsman
   CertificateThumbprint
   ListeningOn = 127.0.0.1, 192.168.110.54, :84, fe80:50f:fe1c:92.168.110.54:11, fe80:50f:fe1c:92.168.110.54:11
   ```

   Execute the following command should you want to change the port WinRM is currently configured to listen on:

   ```
   Set-Item WSMan:\localhost\listener\*\Port 8888
   ```

17. Most importantly, validate that you are able to invoke the Pssession on QA-Web1.Fabrikam.lab by manually running the following command from QA-Web2.Fabrikam.lab. Once you execute the first statement, you'll receive a prompt to enter your credentials. Enter your domain account that has admin permissions:
   
   ```
   $cred = Get-Credential
   ```
18. Executing the next command will use your domain account to connect to the destination server; DNS will be used to resolve the destination name:

```
Enter-Pssession -ComputerName QA-Web1.Fabrikam.lab -Credential $cred
```

```
PS C:\Users\tarun> $cred = Get-Credential
```

```
PS C:\Users\tarun> Enter-Pssession -ComputerName QA-Web1.Fabrikam.lab -Credential $cred
```

```
PS C:\Users\tarun> Enter-Pssession -ComputerName QA-Web2.Fabrikam.lab -Credential $cred
```

19. Follow steps 1 to 5 to configure WinRM on other machines in the lab. Follow step 6 to validate WinRM connectivity before moving forward.

**How to do it...**

1. Navigating to the test hub in the FabrikamTFVC Team Web Portal, on the **Machines** page, click on the + icon to create a new Machine Group:

```
<table>
<thead>
<tr>
<th>Test plan</th>
<th>Parameters</th>
<th>Runs</th>
<th>Machines</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>&lt;</td>
</tr>
</tbody>
</table>
```

2. Enter the details as illustrated in the following screenshot:

![Machine Group Configuration](image-url)
3. The WinRM protocol in Fabrikam.lab will use HTTP since the remote machine has a trust relationship with Fabrikam.lab. Add and the details for all the machines. Now, click on Done to complete the setup:

<table>
<thead>
<tr>
<th>Group Name</th>
<th>Provider</th>
<th>Created By</th>
<th>Date Created</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fabrikam-QA</td>
<td>Pre-existing machines</td>
<td>Tarun Arora</td>
<td>less than a minute ago</td>
</tr>
</tbody>
</table>

### How it works...

The Fabrikam-QA Machine Group setup uses a common administrator credentials for all machines in the Machine Group. It is alternatively possible to specify different credentials for the individual machines added in the Machine Group:

1. To enter credentials per machine, check the option **Use custom credentials for each machine along with global credentials**.

2. The password field is masked in the user interface. In addition to this, the value of this field is not printed in any of the log files either.

3. The tags provide a great way to query for machines with in the Machine Group. For example, when using the test agent deployment task in build definition, you can specify a Machine Group and use Tags to filter the execution of the action on machines that include the Tag.


### Deploying a test agent through the TFBuild task

In previous versions of TFS, the test controller and agent used to ship as separate installers. While the installers supported unattended installation, the configuration needed manual intervention. This limited the ability to scale out the test agents on demand. In TFS 2015, the test controller and agent do not ship as separate installers; instead, the capability offered by the test agent has been distilled into one build task. In this recipe, you’ll learn how to deploy a test agent on multiple machines in a Machine Group.
### Getting ready

Scenario: The FabrikamTFVC Team has a dependency on the cucumber framework, since cucumber conflicts with the existing framework on the build agent, it cannot be installed on the build agent. The FabrikamTFVC Team instead wants the test agent to be deployed on the QA-App1.Fabrikam.lab server. This server is already part of the Fabrikam-QA Machine Group:

![Diagram showing Fabrikam Corp Network, Build Agent, Trust Relationship, Machine Group, and Framework: Cucumber]

In this recipe, we’ll be using the Fabrikam-QA Machine Group. If you don’t already have Machine Group set up, follow the instructions in the recipe *Creating and setting up a Machine Group*.

### How to do it...


2. Navigate to the build hub and create a new build definition using the empty build template. Ensure that the code repository is mapped as FabrikamTFVC in the repository tab. Add the **Visual Studio Test Agent Deployment** task from the test section. The task comprises of three sections; configure the Fabrikam-QA Machine Group, select the machine by machine names or tags, and filter the machine using the Tag **Framework: Cucumber**:

   ![Test Machine Group]

   Test Machine Group / Azure Resource Group: Fabrikam-QA
   Select Machines By: Machine Names Tags
   Filter Criteria: Framework: Cucumber
3. The **Agent Configuration** section accepts the credentials and the test agent service will run under it once installed. Create a variable for the test agent username and the test agent password from the **Variables** tab. Add the variables to the **Agent Configuration** section. Check the **Interactive Process** option to run the agent as an interactive process. The agent is installed as a Windows service if it's not requested to be run as an interactive process:

![Agent Configuration](image)

4. The **Advanced** section allows you to specify the location for the installer of the test agent. This is useful if you want to control the version of the agent getting installer. In the absence of a location, the latest version of the test agent is downloaded from the Internet. Check the option to enable data collection:

![Advanced](image)

5. Save the build definition as FabrikamTFVC Agent Deployment. Queue a new build from this build definition. Once the build is complete, navigate to the QA-App1.Fabrikam.lab server. The test agent has successfully been installed as a Windows service.

**How it works...**

1. The build agent downloads the test agent installer from the Internet into a temp location. The installer is then pushed for installation into the remote QA-App1.Fabrikam.lab machine:

```plaintext
DistributedTests: Machine Group name: Fabrikam-QA
DistributedTests: Run as process: True
DistributedTests: Login Automatically: True
DistributedTests: Disable Screen saver: True
DistributedTests: Agent would be downloaded from http://go.microsoft.com/fwlink/?LinkId=536423
DistributedTests: Update test agent: True
DistributedTests: Run test agent at DataCollection only : True
```
2. The agent is installed as a Windows service on the machines in the Machine Group:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Status</th>
<th>Startup Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visual Studio Test Agent</td>
<td>Provides di...</td>
<td>Started</td>
<td>Automatic</td>
</tr>
</tbody>
</table>

## Distributing test execution on a Lab Machine Group

Earlier in this chapter, you learned how to create a Machine Group and deploy the test agent on selected machines in the Machine Group. The TFBuild framework ships a task to distribute the test execution across multiple test agents. In this recipe, you'll learn how to trigger a distributed test run across a farm of test runners deployed in a Machine Group.

### Getting ready

Scenario: The FabrikamTFVC Team has over 2,000 automated coded UI tests that need to be executed as part of the functional testing workflow. FabrikamTFVC Team would like the ability to parallelize the test execution. In order to do this, the Team wants to deploy the Visual Studio test agent to all machines in the Fabrikam-QA Machine Group and then parallelize by distributing the test execution at the test assembly level:

In this recipe, you'll be using the Fabrikam-QA Machine Group. If you don't already have Machine Group set up, follow the instructions in the Creating and setting up a Machine Group recipe. Since test agents need to be deployed as a prerequisite to the distribution of the tests, follow the steps in the Deploying a test agent through the TFBuild task recipe to learn how TFBuild can be used to deploy the test agent in a Machine Group.
Testing Your Application

How to do it...

1. Navigating to the build hub in the FabrikamTFVC Team Web Portal, click on Edit to open FabrikamTFVC Agent Deployment in edit mode.

2. Click on + to add a build step and from the Add Task window, select the Build tab and add the Visual Studio Build task. This task will be used to compile the test projects:

   Visual Studio Build
   Build with MSBuild and set the Visual Studio version property

3. From the Deploy tab, add the Windows Machine File Copy task. This task will copy the test assemblies across to the machines the test need to be executed:

   Windows Machine File Copy
   Copy files to remote machine(s)

4. From the Test tab, add the Visual Studio Test using Test Agent task. This task will be used to manage the test distribution across the farm of test agents:

   Visual Studio Test using Test Agent
   Run tests across multiple test agents on a lab machine group in a distributed fashion

5. Click on close icon to close the Add Task window. The tasks in the build definition need to be ordered as:

   Visual Studio Build
   Build solution "**\*.sln"
   Windows Machine File Copy *
   Copy files to
   Visual Studio Test Agent Deployment
   Deploy TestAgent on CLD-CL
   Visual Studio Test using Test Agent *
   Run Tests "**\test*.dll" on

6. Configure the Visual Studio Build task to build the solution the functional tests are part of:

7. Configure the Windows Machine File Copy task to copy the test assemblies into a directory on the machines in the Machine Group. It is always recommended to parameterize the test locations with configuration variables. The preconfigured build variables can be used to construct the path to the assemblies on the build agent working folder that need to be copied across. If the Clean Target flag is checked, the build engine will delete the contents of the destination folder before copying the new files. The advanced section in this task also includes a flag to enable copying files in parallel and use this flag to allow the build system to copy the binaries across the machines on the Machine Group in parallel:

- Source: $(Build.Repository.LocalPath):code
- Machine Group: Fabrikam-QA
- Select Machines By: Machine Names
- Destination Folder: C:\Fabrikam\FunctionalTests\$(Build.BuildNumber)
- Clean Target: ✓

8. No changes need to be made to the Visual Studio test agent deployment task. The configuration set up in the Deploying a test agent through the TFBuild task recipe will carry out the test agent deployment across all machines in the Machine Group. The Visual Studio Tests using test agent task should immediately follow this task. Select the Fabrikam-QA Machine Group as the destination Machine Group, where the tests need to be executed. In addition to this, specify the test binaries location, the pattern of the test assemblies, the configuration, and the platform the tests need to be executed in. Check the code coverage flag in the Advanced section. You can inject the test settings file to optionally override the configuration in the test settings file through the Override Test Run Parameters field:

- Test Machine Group
  - Test Machine Group / Azure Resource Group: Fabrikam-QA

- Test Settings
  - Test Drop Location: C:\Fabrikam\FunctionalTests\$(Build.BuildNumber)
  - Test Assembly: **$(BuildConfiguration)\"tests\".*; **\"obj\".*
9. Save the changes to the build definition and queue a new build. Once the build completes, navigate to the build summary page; the logs for the Visual Studio Tests using test agent step shows the details of the test distribution:

```
#(debug)Importing cmdlet 'Invoke-DeployTestAgent'.
#(debug)Importing cmdlet 'Invoke-RunDistributedTests'
#(debug)Getting the connection object
#(debug)Unregister-TestAgent script Path -
#(debug)Calling Invoke-RunDistributedTests
#(debug)DistributedTests: Creating DLL client
```

**How it works...**

Let's look at the log files of the Visual Studio Test using a test agent task through the build summary. As illustrated in the preceding screenshot, the test DLLs have been distributed to multiple test agents in the group. The distribution is done at a DLL level rather than a test level. The test distribution significantly reduces the execution time if your tests are in multiple projects since each DLL is distributed to a test agent to execute.

It is also worth noting that the Windows Machine File Copy task successfully moves the files from the agent working directory across to the C:\Fabrikam\FunctionalTests\$(Build.BuildNumber) folder in the Machine Group:

```
\ SYSTEM (C:) \ Fabrikam \ FunctionalTests \$
Name ~
2015:120:14
```

You'll learn more about the Windows Machine File Copy task in *Chapter 6, Releasing Your Application*, as it'll be used extensively for deployments in release management.

**Triggering Selenium Web Tests on a Selenium Test Grid using TFBuild**

The cost of testing can significantly increase if your software needs to support multiple devices and browsers. The rise in client-side scripting frameworks and responsive design technologies enable each browser to offer a unique immersive user experience. With the matrix of devices and browsers rapidly increasing, it is virtually impossible to cover the matrix with manual testing. There are a number of commercial and open source tools available for assisting with the development of test automation. Selenium is possibly the most widely used open source solution. TFS works better together with open source tools. In this recipe, you'll learn how to execute Selenium Tests from TFBuild.
Getting ready

Scenario: The FabrikamGit Team has automated functional tests using Selenium. The Team now wants to validate the functionality by running the tests in Chrome, Firefox, and Internet Explorer. The FabrikamGit Team would like to trigger the execution of these tests on a Selenium grid via TFBuild.

Follow the instructions in the blog post at http://bit.ly/1PSjfwy to set up a Selenium grid on a Windows machine. For the purposes of this recipe, we'll follow a simple setup that comprises of both the Selenium hub and node on the same machine:

The course material provided with this book includes a Selenium Test Project. Download the SeleniumHelloWorld.zip folder; alternatively, you can use your own Selenium Test Project. Commit and publish Selenium.Web.Test.sln into the master branch in the FabrikamGit repository:
More information on how to commit and publish changes to a Git repository is available in the introduction section of Chapter 2, Setting Up and Managing Code Repositories. The SeleniumHQ website (http://bit.ly/1YmqRDJ) includes great C# tutorials for authoring Selenium Tests.

**How to do it...**

1. In the FarbikamGit Team Web Portal, navigate to Build hub. Click on the + icon to add a new build definition. Select the Visual Studio Build Template and click on Create. Navigate to the Repository tab in the build definition and map the repository to the master branch of the FabrikamGit repository:

   ![Repository configuration](image)

2. In the build definition navigate to the Build tab, select the Visual Studio Build task. Set the name of the solution containing the Selenium Test Project. Ensure that the Restore NuGet Packages flag is checked:

   ![Build configuration](image)

3. Select the Visual Studio Test task, enter the path to the test settings file, and inject the URL of the Selenium grid via the Override TestRun Parameters field:

   ![Test settings configuration](image)
4. Save the build definition as FabrikamGitSelenium. Queue a new build for this definition. Once the build execution completes, you can see the test results from the build summary view:

<table>
<thead>
<tr>
<th>Time</th>
<th>Status</th>
<th>Test Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>21:07:975961Z</td>
<td>Starting</td>
<td>test execution, please wait...</td>
</tr>
<tr>
<td>21:22:8677541Z</td>
<td>Passed</td>
<td>Selenium_ClickApplicationName_FireFox</td>
</tr>
<tr>
<td>21:28:3767143Z</td>
<td>Passed</td>
<td>Selenium_ClickApplicationName_Chrome</td>
</tr>
<tr>
<td>21:36:2059993Z</td>
<td>Passed</td>
<td>Selenium_ClickAbout_FireFox</td>
</tr>
<tr>
<td>21:43:5909377Z</td>
<td>Passed</td>
<td>Selenium_ClickAbout_Chrome</td>
</tr>
<tr>
<td>21:49:8614017Z</td>
<td>Passed</td>
<td>Selenium_ClickContact_FireFox</td>
</tr>
<tr>
<td>21:55:5099186Z</td>
<td>Passed</td>
<td>Selenium_ClickContact_Chrome</td>
</tr>
</tbody>
</table>

**How it works...**

Let's start by understanding why the Selenium grid URL was injected in the Visual Studio Test task via the **Override TestRun Parameters** field. The solution contains Selenium Test Project and a Run settings file. This is the configuration file holding test settings used by the Test Project. The new build system provides the ability to overwrite the values in the Test settings file by directly passing the values through the **Override TestRun Parameters** field, in this specific case, the value of SeleniumTestGridURL. This value is used by the Selenium Test to execute the test:

```xml
<TestRunParameters>
  <Parameter name="gridHub" value="http://fab-selenium-grid.cloudapp.net:4440/wd/hub" />
</TestRunParameters>
```

Next, let's look at the build execution log file for this task; this provides a clear summary of the test executed, duration, and update on the result publishing:

<table>
<thead>
<tr>
<th>Time</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Test execution time: 41.6065 Seconds</td>
</tr>
<tr>
<td>21:55:9473514Z</td>
<td>Publishing Test Results...</td>
</tr>
</tbody>
</table>

Last but not least, the TFBuild system automatically handles the publishing of test results back to the TFS, independent of the testing framework used:

<table>
<thead>
<tr>
<th>Test Results</th>
<th>Release: any cpu</th>
<th>Run completed 4 days ago</th>
<th>All 6 tests passed</th>
</tr>
</thead>
<tbody>
<tr>
<td>864/VSTest Run</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

As you can see in the preceding screenshot, the Selenium Test Project execution results can directly be viewed in the build summary section.
Testing Your Application

Integrating the Cloud Load Testing Service in TFBuild

"Tests at Amazon revealed: every 100 ms increase in load time of Amazon.com decreased sales by 1% (Kohavi and Longbotham 2007)."

Performance Testing can't be an afterthought! Mature software teams test for performance early in the development life cycle. The biggest entry barrier to performance testing is the high cost and low utilization of infrastructure required to generate sufficient load on the application. Today, a wide range of pay-as-you-go Cloud Load Test Services are available. Microsoft also offers a Load Test Service with Visual Studio Team Services. Each VSTS account receives 20,000 virtual user minutes of Load Test Quota free every month. In this recipe, you'll learn how to integrate the VSTS Cloud Load Test Service into your CI Pipeline in TFS.

Getting ready

Visual Studio Team Services is free for Teams of up to five people. You can create a free account at http://bit.ly/1lANwhA. In this recipe, we'll be using the account https://tfs2015cookbook.visualstudio.com.

For TFS to authenticate with the VSTS account using your identity, you will need to generate a personal access token in the VSTS account. Follow the instructions at http://bit.ly/1I3kzVk to generate a personal access token.

Now that we have the Cloud Load Test service in VSTS and personal access token, the Cloud Load Test service needs to be added as an endpoint in TFS. Adding an endpoint in TFS requires that you have Edit project level information permissions. You can acquire this permission by being added to the Team Project Administrator Group. Follow the instructions here to add the Cloud Load Test Service endpoint to TFS.

Navigate to FabrikamTFVC Team Administration Console. In the Services tab, click on the + icon to add a new endpoint. Select the Generic endpoint option and fill out the details as illustrated in the following screenshot. Name the endpoint CLD-VSTS.
How to do it...

1. Navigate to the Build hub in the FabrikamTFVC Team Web Portal. Edit the FabrikamTFVC CI build definition.

2. Click on the + icon to launch the Add New Task window. From the Test tab, select and add the Cloud-based Web Performance Test task:

3. The Cloud Load Test Service was added as a generic endpoint in the Fabrikam Team Administration Console. Select the CLD-VSTS endpoint from the registered connection dropdown. Enter the endpoint you would like to run the test against. Specify the location to generate the load from:

4. Click on Save and trigger queue a new build. Once the build execution completes, the results of the load test execution is available as part of the build result:
How it works...

You can start to load test your application with simple configuration. It is easy to get started and utilize the free Cloud Load Test Virtual User Minutes available in your VSTS account. TFBuild also includes a Cloud-based Load Test task that allows you to run multistep performance tests.

Analyzing test execution results from the Runs view

In Team Foundation Server 2015, the test execution results of both manual and automated testing are surfaced in the Runs page. This page is a new addition to the Test hub in Team Web Portal. The Runs page offers a unified experience for analyzing the results of test executed using any framework. In this recipe, you’ll learn how to analyze and action the test execution results in the Runs view in Team Web Portal.

How to do it...

1. Navigate to the Test hub and click on Runs to load the Runs page. The Runs page displays the recent test runs. At first glance, you can see the test execution status, test configuration, build number, number of failed tests, and the pass rate:

<table>
<thead>
<tr>
<th>State</th>
<th>Run Id</th>
<th>Title</th>
<th>Completed Date</th>
<th>Build Number</th>
<th>Failed</th>
<th>Pass Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>✔</td>
<td>15</td>
<td>VSTest Test Run debug any cpu</td>
<td>11/20/2015 12:27:05 AM</td>
<td>20151120.3</td>
<td>1</td>
<td>83.3%</td>
</tr>
<tr>
<td>✔</td>
<td>14</td>
<td>VSTest Test Run debug any cpu</td>
<td>11/18/2015 12:21:55 AM</td>
<td>1.0.15322.61</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>✔</td>
<td>13</td>
<td>VSTest Test Run debug any cpu</td>
<td>11/18/2015 12:13:48 AM</td>
<td>1.0.15322.60</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>✔</td>
<td>12</td>
<td>VSTest Test Run debug any cpu</td>
<td>11/16/2015 2:43:36 PM</td>
<td>1.0.15320.55</td>
<td>38</td>
<td>97.7%</td>
</tr>
</tbody>
</table>

2. Navigate to the Filters view by clicking on the Filters tab. The query is defaulted to display the test runs from the last 7 days. Amend and add new clauses to show only the automated test runs for today:

<table>
<thead>
<tr>
<th>And/Or</th>
<th>Field</th>
<th>Operator</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>✔</td>
<td>Created date</td>
<td>&gt;=</td>
<td>@Today</td>
</tr>
<tr>
<td>✔</td>
<td>Is automated</td>
<td>=</td>
<td>True</td>
</tr>
<tr>
<td>✔</td>
<td>State</td>
<td>=</td>
<td>Completed</td>
</tr>
<tr>
<td>✔ Add new clause</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

202
3. The query narrows down the test execution results to just one run:

<table>
<thead>
<tr>
<th>State</th>
<th>Run id</th>
<th>Title</th>
<th>Completed Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>✔</td>
<td>15</td>
<td>YSTest Test Run debug any cpu</td>
<td>11/20/2015 12:27:05 AM</td>
</tr>
</tbody>
</table>

4. Double-click on test run ID to open the test run for analysis. This view shows the run summary along with charts to visualize the test results by properties, traits, configuration, failures types, and resolution. Any attachments associated to the test run are also available in this view:

5. Navigate to the **Test results** tab to see the list of all tests executed as part of this test run. Prior to TFS 2015, you would have had to download the TRX file and open it in Visual Studio to get to this information. This view provides the next level of detail among other things. You can see the test execution duration and failure error messages:
6. Select multiple tests and click on Create Bug to create a bug-type Work Item. Clicking on Update Analysis lets you add comments to the test results. You can also double-click on a test to go the next level of detail on its test execution:

<table>
<thead>
<tr>
<th>UPDATE ANALYSIS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analysis owner</td>
</tr>
<tr>
<td>Tarun Arora</td>
</tr>
<tr>
<td>Failure type</td>
</tr>
<tr>
<td>New Issue</td>
</tr>
<tr>
<td>Resolution</td>
</tr>
<tr>
<td>Configuration</td>
</tr>
<tr>
<td>The IE configuration is causing the integration tests to fail.</td>
</tr>
</tbody>
</table>

**How it works...**

This functionality gives you a unified test analysis experience irrespective of the framework you choose to execute your tests in. In summary, you can query all test runs available in your Team Project, drill down into a specific test run to get a summary view of that run, visualize test runs using charts, query/filter the test results within a run, drill down to a specific test result, download attachments, and last but not least, analyze test failures and file bugs.

**Exporting and importing test cases in Excel from TFS**

While test cases in TFS can be accessed from Microsoft Test Manager as well as Team Web Access, you may still find stakeholders wanting to access the test cases from Excel. As a matter of preference, a few of your stakeholders may find it easier to document test cases in Excel. While TFS does not offer an out-of-the-box feature to export and import test cases in Excel, it offers a good **Software Development Kit (SDK)** that can be used to create tools that enable such functionality. In this recipe, you'll learn how to use "test case export to excel" and "Test Case Migrator Plus (open source community tools)" to export test cases into Excel and import test cases from Excel into TFS.

**Getting ready**

Download and install the following utilities on your machine:

- **Test case export utility:** [https://tfstestcaseexporttoexcel.codeplex.com](https://tfstestcaseexporttoexcel.codeplex.com)
- **Test case import utility:** [http://tcminpoort.codeplex.com](http://tcminpoort.codeplex.com)
You'll need elevated permissions to install these utilities. The machine where these utilities are installed needs a version of Excel installed. These utilities do not need to be installed on the TFS application tier.

**How to do it...**

**Exporting test cases from TFS**

1. On starting the **Test Cases Export** utility, you'll be prompted to connect to TFS. Once connected, you'll see a list of test plans and test suits associated to that plan. Select the test suite and the file location for export and fill out the export output location and filename as illustrated in the following screenshot:

![Export Test Cases from TFS to Excel](image)

2. The export process gives you the following options:

   - **Export Options**
     - Export the Selected Suite Only.
       *(Test cases from the sub suits will not be exported)*
     - Export Each Test Suite into Separate Sheets
     - Export Test Results

3. Select the **Export Each Test Suite into Separate Sheets** option to export each test suite into a separate sheet. In the preceding example, regression pack, payments, and website will be exported in three separate sheets in the workbook.

4. Once the export completes, open the spreadsheet from the export location to view the results of the export.
Importing test cases from TFS

1. Start the Test Case Migrator Plus utility to launch the test case import wizard.
2. Select the data source from the Source tab. The input format can either be Excel Workbook or MHT/Word. In this recipe, we’ll be using the input source format as Excel Workbook. Key in the output folder location. Once the workbook has been parsed, you’ll see the list of columns in the selected worksheet. Click on Next to configure the Destination settings:

   Select Data Source Type: Excel Workbook
   Specify file path for excel workbook:
   C:\TestExport\Regression
   1 worksheets found.
   Select the worksheet to migrate:
   Regression Pack
   Enter row number that has field names: 1
   Preview:
   Field Names
   TC No

3. From the Destination tab, enter the destination details such as Team Project and Work Item type to import the test cases to. From the Settings tab, choose to create a new settings file and click on the Next button (the settings file will persist your selections for future imports). From the Field mapping tab, map the columns in the spreadsheet to the Work Item. For example, the Title field in the spreadsheet maps to the title field of the Work Item.

4. From the Links mapping tab, you can optionally enable linking between Work Items using the Miscellaneous tab to specify how the data is read from the source Excel file. Click on Save and Migrate to start the migration process based on the settings configured through the wizard. This will trigger the import process; progress on the update is shown on the user interface:

   ![In Progress Progress Table]
   Total: 189
   Migrated Successfully: 95
   Errors: 0
   Warnings: 0

   Details:
   - Status: Success
   - Action: Parse Data Source
   - Message: Success
   - Status: Success
   - Action: Save Settings
   - Message: Success
   - Status: In Progress
   - Action: Migrate Work Items
   - Message: 95 of 189 is processed
   - Status: Pending
   - Action: Publish Report
5. Once the import has been completed, exit the wizard by clicking on the Close icon button. The import utility processes the worksheet and generates a report in the same location as the source worksheet. The import settings file is also generated in the same location. The import settings file stores the settings specified during the import process. The settings file can be used for future imports:

![TestCases ▶ Report7_6_2015_9_01_PM](Report.xls)

6. Open the Report.xls file and all successfully imported test cases will be listed in the Passed sheet. Any test cases that have issues during import will be listed in the warning sheet, and any test cases that couldn't be imported will be listed in the failed sheet.

7. To view the imported test results, navigate to the FabrikamTest hub in Team Web Portal. You'll see the new plans appear in the test plan dropdown. The following screenshot illustrates a test suite and a linked test case in Team Web Portal:

![How it works...](How it works...)

This, by no means, is a Test Case Migration solution, but it is worth pointing out that if you are using other Test Case Management tools that support extracting test cases into Excel, you could use these tools to import the artifacts into TFS.
There is more

While we are on the subject of test suites, it is worth highlighting that TFS 2015 now supports assigning multiple people to a test suite. This capability can be invoked by right-clicking on a test suite in test hub and choosing Assign Individuals from the context menu. As illustrated in the following screenshot, the new form allows assigning multiple testers to the test suite:

Assign all the test cases in this test suite to be run by multiple testers. For example, you can assign the tests to all your user acceptance testers. Then send them an email to let them know that the tests are ready to be run.

Select testers
If you want to have multiple users run the same test cases in this test suite, add these users to this list. Tests are created from all the test cases for each tester.

Aaron Bjo工序X  Charles SterlingX  Chris PattersonX  Claude RemillardX  Gopinath CHX

Display Name or Microsoft Account  |  Browse  |  Check name

If at a later time you decide to remove a tester from this list for this suite, then the tests that were created for this tester are removed.

Send email

This will iterate through the test suite and create test cases for each individual. An e-mail with the link to the test cases is e-mailed out to the individuals. This is a great way to enable multiple individuals to test using the same test suite, giving you the ability to track the test activity of each individual in isolation.

Copying and cloning test suites and test cases

TFS offers two types of copy operations for test suites and test cases, namely, copy and clone. Copy uses a mechanism called shallow copy that simply creates a reference to the artifact. If any amendment is made to the artifact, it reflects into all its references. Clone uses a mechanism called deep copy; the new artifacts have no reference back to its origin and is not impacted by any updates made to the original artifact. A common scenario for using shallow copy is when testing using the same scripts across multiple iterations. On the other hand, a common scenario for deep copy is creating a regression test script by copying several existing test suites and test cases. In this recipe, you'll learn how to use the copy and clone functionality.

Getting ready

In this recipe, we'll be using Microsoft Test Manager. Microsoft Test Manager is installed with Visual Studio Enterprise or Visual Studio Test Professional. Launch Microsoft Test Manager and connect to the FabrikamTFVC Team Project.
How to do it...

Copying (shallow copy)
1. In Microsoft Test Manager, from the Plan tab, select a static test suite that will hold the new copy.
2. From the toolbar, click on the Create test suites by referencing existing test cases button.
3. From the dialog, select the test plan and test suites that you want to copy and click on Create Suite.

Clone (deep copy)
1. In the Microsoft Test Manager, from the Organize tab, select a test plan.
2. From the toolbar, click on the Clone button.
3. From the dialog, select one or more test suites that you want to clone. You can optionally choose to clone requirements that are associated with the selected test suite.
4. Enter the name of the destination test plan and set the associated Area Path and Iteration Path.
5. Next, click on the Clone button to trigger the clone operation. The clone log appears showing updates of the clone process.

Cloning test cases is supported between Team Projects, but only in the same Team Project Collection.

You can also use the /clone switch with tcm.exe from command line to clone test cases and test suites. In the following sample, you’ll notice that the /clone requirements is being used to clone the requirements. The value of the field Area Path and Iteration Path in the destination will be overridden with the specified value:

tcm suites /clone
/teampatient:FabrikamTFVC /destinationteampatient:FabrikamGit
/clonerequirements
/suiteid:234 /destinationsuiteid:567
/overridefield:"Iteration Path"="FabrikamGit\sprint3"
/overridefield:"Area Path"="FabrikamGit\catalog"
How it works...

When you clone a test suite, the following objects get copied from the source test plan to the destination test plan:

<table>
<thead>
<tr>
<th>Test plan object</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test case</td>
<td>Each new test case retains its shared steps. A link is made between the source and new test cases. The new test cases do not have test runs, bugs, test results, and build information.</td>
</tr>
<tr>
<td>Shared steps referenced by cloned test cases</td>
<td>Any shared steps referenced by the source test cases are maintained in the cloned test cases at destination.</td>
</tr>
<tr>
<td>Test suite</td>
<td>The following data is retained:</td>
</tr>
<tr>
<td></td>
<td>- Names and hierarchical structure of the test suites</td>
</tr>
<tr>
<td></td>
<td>- Order of the test cases</td>
</tr>
<tr>
<td></td>
<td>- Assigned testers</td>
</tr>
<tr>
<td></td>
<td>- Configurations</td>
</tr>
<tr>
<td>Action recordings linked from a cloned test case</td>
<td>Any action recording links are maintained in the clone test case.</td>
</tr>
<tr>
<td>Links and attachments</td>
<td>Any links and attachments in the source are also available in the cloned test case.</td>
</tr>
<tr>
<td>Test configuration</td>
<td>The test configuration is reapplied in the destination test plan.</td>
</tr>
</tbody>
</table>

Exporting test artifacts and test results from the test hub

TFS test artifacts comprise of test plans, test suites, and test cases, and of course, test results. It is common to have to export the test artifacts for purposes of sharing and reporting. Back in the days of TFS 2013 Test Scribe delivered as a Visual Studio Extension, it was the only way to export these artifacts from TFS. Test hub now boasts the e-mail or print test artifacts functionality that allows you to easily share test artifacts with stakeholders. The feature is simple to use and can be triggered from several places within the test hub in Team Web Portal. In this recipe, you’ll learn how to export the test artifacts from the test hub.
How to do it...

1. Browse to the FabrikamTFVC Team web access and navigate into Test hub. To export at the test plan level, select the test plan, and then click on the Email or print test artifacts button from the toolbar:

![Image 1](image1.png)

2. To export at the test suite level, select the test suite open the context menu and chose the Export option:

![Image 2](image2.png)
3. Whether you choose to export from test plan or test suite, in both cases, you will get a new form to select "what" and "how". The "what" in this case being the artifacts. The "how" in this case being e-mail or print. A few items are worth highlighting in the following screenshot. The **Latest test outcome** option has been added in Update 1, selecting this option also exports the test results. Choosing **Selected suite + children** recursively exports all children of the selected suite:

![Screenshot of export options](image)

How it works...

Clicking on print or e-mail starts the process of generating the extract; this may take up to a few seconds to complete depending on the quantity and size of the artifacts being exported. Once the export has been completed, a form will pop up to show you the preview of the export. You can also edit and format the values from the preview form. Since we have chosen the e-mail option, the form has a field that allows us to choose the e-mail address of the person we would like the export to be sent out to:

![Preview form](image)
As illustrated in the following screenshot, the export also includes the test steps:

<table>
<thead>
<tr>
<th>#</th>
<th>Action</th>
<th>Expected value</th>
<th>Attachments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Login to the system</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Click on Create New Ticket</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Enter Title, Description and Severity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Click on Select customer</td>
<td>This action should bring up another dialog showing existing customers</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Search by customer last name, first name and select appropriate customer</td>
<td>This should filter customers with given last name and first name</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Click on “Create”</td>
<td>This action should create the ticket and show ticket details back to the customer</td>
<td></td>
</tr>
</tbody>
</table>

### There is more...

It is possible to customize the format of the export by modifying the underlying template used by TFS during the export/print process. The following points are to be kept in mind before customizing the template:

You should create a backup of the original template, for example, copy and rename it to TestSuite-Original.xsl. If not, when you upgrade TFS, the changes you made in the TestSuite.xsl file may get overwritten. The export does not support customization per project; the style changes will affect all projects in your TFS instance.

Follow the steps listed here to add your company logo to the export:

1. Log on to the TFS application tier and navigate to the following path. Add your company logo (companylogo.png) in the C:\Program Files\Microsoft Team Foundation Server 14.0\Application Tier\Web Services\_static\tfs\12\_content folder path.

2. Modify the TestSuite.xsl file in the C:\Program Files\Microsoft Team Foundation Server 14.0\Application Tier\Web Services\_tfs_resources\TestManagement\v1.0\Transforms\1033 folder.
3. Open the TestSuite.xsl file in the notepad and add the following lines of code to include your company logo into the export template:

   <div style="align:center;">
   <img src="../../_static/tfs/12/_content/companylogo.png" />
   </div>

4. The results of the customization can be tested by generating an export through the test hub in Team Web Portal.

---

**Charting testing status on Dashboards in Team Portal**

The charting tools in Team Web Portal provide a great way to analyze and visualize test case execution. The charts created through the charting tools can be pinned to custom Dashboards. Both charts and Dashboards are fantastic information radiators to share the test execution results with Team members and stakeholders. In this recipe, you'll learn how to pin the test execution results on a custom Dashboard in Team Portal.

**Getting Started**

Follow the steps in the Configuring Dashboards in Team Project recipe in Chapter 1, Team Project Setup, to create a custom Dashboard for testing.

**How to do it...**

1. Navigate to the Test hub in FabrikamTFVC Team Web Portal. The Test Plan page gives you a list of test suites and a list of test cases for the selected suite. The Charts tab gives you a great way to visualize this information. Click on the + icon and select New test result charts. Select a bar chart and group by Outcome; this renders the test case outcome in the bar chart. Click on OK to save the chart. Right-click on the newly created chart and pin the chart to the testing Dashboard:
2. Now, click on the + icon and select the **New test case** chart. Test case chart types support trend charts and the supported trend period is from 7 days to up to 12 months. Select the stacked area chart type and chose to stack by **State**. This will allow you to visualize the state of the test cases over time. Click on OK to save the chart, right-click on the chart and pin it to the Dashboard:

![Chart Image](image)

**How it works...**

The charts are calculated using the Work Item data in the transactional database. When Work Items are updated, the charts reflect the updates immediately. To learn more about the charting functionality in Team Web Portal refer to the walkthrough at [http://bit.ly/1PGP8CU](http://bit.ly/1PGP8CU).
Where to buy this book

You can buy Microsoft Team Foundation Server 2015 Cookbook from the Packt Publishing website.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

Click here for ordering and shipping details.