CHAPTER

1

Azure DevOps 實戰

這一章，我要來帶你體驗 Azure DevOps 的威力。廢話不多說，我們將會直接帶你建立一個自動化的建置與部署流程，讓你體驗近代軟體開發之所以能夠輕易實現一週交付數次、甚至是一天交付數次，卻又能夠同時維持高品質、且兼顧安全性的關鍵。

在體驗所謂的頻繁整合、持續交付的同時，你將完整的看到從需求出現，到功能交付到用戶手上的這整個自動化過程，並且讓你的團隊也可以建立出這個流程，且享有其帶來的好處與價值。

歡迎你立即加入這個旅程。

1-1 Azure DevOps 一日實戰

1-1-1 今晚，你想要來點…

開始前，我們先問自己一個問題。

今天，你的團隊，從需求（或 bugs）出現，一直到將其生產出來並且毫無瑕疵地交付到用戶手上，需要多久時間？

在 2021 年的時候，估計大家都有在網路上預約疫苗或是申請五倍券（或是其他什麼券）的經驗吧？告訴我，倘若你上網登記時發現網站故障了，你希望它多久修復好？倘若你覺得它很難使用，而設計單位也承諾要著手改善，你希望網站多久能夠完成更新？每當我問學員這個問題，答案都是同一個，就是「越快越好」，最好能夠立刻、馬上、隨時。

如果，我們對其他人的網站是這麼要求的，那我們的客戶呢？我們的客戶對我們的期待是不是也是如此？

過去（大概五到十五年前），軟體不管碰到 bugs 或是新功能的交付，往往必須等候數個月甚至數年，還記得以前 Windows 作業系統或是 Office 軟體的更新頻率嗎？然而，如今我們對系統更新頻率的期待已經變更為數週、數天、甚至數小時…。

而持續縮短這個等候時間，同時還能夠一併提升（而非犧牲）軟體的品質與安全性，這才是 DevOps 想追求的核心價值。

如何讓你的團隊實現快速、高品質的軟體更新與功能交付，就是我們這章想和你談論的話題。

1-2 你的預備動作

待會我們會立刻帶你體驗一下 Azure DevOps 的自動化快速建置與部署流程，但在此之前，建議你先做好一些基本準備…

1-2-1 你需要的各種帳號與軟體

在進入本書介紹的 Azure DevOps 之前，我們貼心建議你，先申請好底下幾種帳號與服務，別擔心，大多都是可以免費申請的：

1. Microsoft Account（一切的基礎）
   https://account.microsoft.com/
2. Github 帳號（optional）
   https://github.com/
3. Azure DevOps 站台（這是我們的主角，你非申請不可）
   https://dev.azure.com/
4. Azure Portal Free Trial
   https://azure.microsoft.com/zh-tw/free/

我們會用到的軟體工具包含：

1. Visual Studio Code（可跨平台）
   https://code.visualstudio.com/
2. Visual Studio 2019 +（optional）
   https://visualstudio.microsoft.com/zh-hant/vs/
3. Postman（可跨平台）
   https://www.postman.com/
4. PowerShell（可跨平台）
   https://docs.microsoft.com/zh-tw/powershell/scripting/install/installing-windows-powershell?view=powershell-7
5. Azure CLI（可跨平台）
   https://docs.microsoft.com/zh-tw/cli/azure/install-azure-cli?view=azure-cli-latest

先知道有上面這些，我們接著會介紹如何申請。

建議你依照底下短網址連結到的動畫中的操作步驟，來建立一組新的帳號：https://wwjd.tw/673k914

當你建立好一組可用的 Microsoft Account 之後，就可以使用該帳號來申請免費的 Azure DevOps 與 Azure 雲端服務了。

1-3-1　申請免費的 Azure DevOps 服務

Azure DevOps 其實就是以前的 TFS，微軟現在依舊有地端 On-Premises 版本的相同服務，名稱就叫做 Azure DevOps Server，詳細資訊你可以參考底下網址：

```
https://azure.microsoft.com/zh-tw/services/devops/server/
```

而我們要申請的，則是雲端版本，完整的名稱是 Azure DevOps Services，進入申請網址「https://dev.azure.com/」後，你會看到「類似[2]」底下的畫面：

點選「Start Free」之後，就可以開啟申請，過程中如果需要你登入，請用你剛才申請好的 Microsoft Account 登入即可。

[2] 為什麼說「類似」？因為 Azure DevOps 站台似乎有做近代網站常見的 A/B Test，不同的帳號登入可能會有不同的 UI/UX。再加上，現在雲端網站的改版異常迅速，可能每 1 2 個月就會有些異動。因此，我們也只能說「類似」了。如果有相當大的改版，我們會在本書的網站上補充。

就這樣，你已經把我們在 Github 上的 ASP.NET Core 專案範例程式碼匯入進來囉。

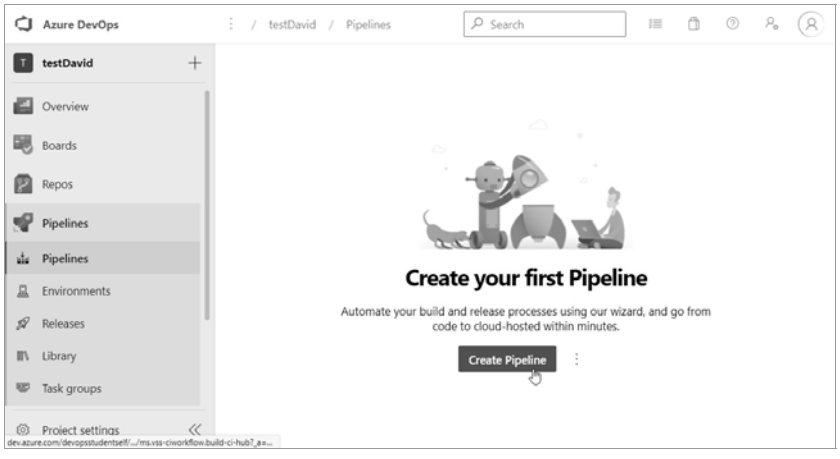
匯入這個範例專案程式碼，是為了讓你搶先體驗一下 Azure DevOps 在 Git Repos 的使用與 Pipeline 設計上的便利性，並且，立刻帶你實際上著手建立一個自動化 CI/CD Pipeline。

未來，你應該會和你的開發團隊一起在 Repos 中重新建立專案的程式碼，細節的部分我們會在後面的章節中介紹。

1-3-6 建立自動化 CI Build

在成功的匯入程式碼之後，你就可以嘗試建立一個雲端的自動化建置流程 -- 「CI Build Pipeline」。

請在左方的主選單中，選擇 Pipelines 分類，並且點選其中的 Pipelines 項目，視窗正中間應當會出現如同底下的畫面：



單元測試，是透過撰寫驗證程式碼，來驗證我們程式中的運算邏輯。順利通過（pass）單元測試，表示我們的程式碼有著一定程度的品質檢核。我們應該要盡可能的，讓單元測試幫助我們驗證核心邏輯的正確性，並且減少重複犯錯的機會，這部分細節我們後面介紹 CI 的章節會再詳細一點討論。

現在，我們要先將建置好的成品，直接部署到特定網站。

1-5 實現自動部署

嚴格來說，自動化部署，我們應當放到 CD（Continuous Delivery）Pipeline 當中來進行。這部分在 Azure DevOps 當中，是屬於 Release Pipeline 的範疇（後面會有專屬的章節介紹）。不過，我們現在可以先在 CI Pipeline 中讓大家體驗一下。

要將剛才在 CI Pipeline 當中建置好的成品（Artifacts），部署到特定網站，變成真正可以運行的系統，那我們當然得先建立出一個網站。底下，我們會帶大家在 Azure 雲端上建立一個 Web App 網站，並且將系統透過自動化 Pipeline 部署上去。

1-5-1 申請 Azure Portal

建立網站之前，你必須有 Azure 雲端服務訂閱（Subscription），我們採用 Azure 雲端服務，你可以在底下網址免費申請（但若你並非學生，則需要註冊信用卡，不過也是免費的）：

```
https://azure.microsoft.com/zh-tw/free/
```

除此之外，你也可以選擇加入 Visual Studio Dev Essentials 方案，這是微軟提供給開發人員的免費方案，一口氣提供開發人員許多好康，其中也有我們需要的 Azure 服務與 Azure DevOps：

CHAPTER

2

持續整合的基礎 — 版控

我們非得在介紹 CI（持續整合）/CD（持續交付）之前，先來談談 Azure Repos 與 Git 版控這些個議題。因為，所有自動化流程的觸發，其實是由 PR 開始的。

版控（特別是 Git）與分支（Branch）策略的選擇，很大幅度的影響後續自動化流程的進行，以及團隊合作的順暢與否。而 PR（Pull Request）則是後續 CI/CD 被觸發的源頭，這些都是 Git Repos 之所以如此重要的原因。

在這一章中，我們不會介紹基礎的 Git 觀念（請讀者自行選擇其他教材補足相關概念），而是直接介紹 Azure Repos 中如何使用 Git 版控機制，以及如何透過 PR 來進行所有的工作。不論你過去是否熟悉 Git，都別錯過本章的介紹與說明。

2-1 一切都是為了頻繁交付

2-1-1 沒有持續整合，就沒有頻繁交付

前面我們提過，想要實現頻繁交付，且交付給用戶的成果不能充斥著品質上的瑕疵，要能夠符合一定的水準，程式碼持續整合是必須的。

當你有夠多的開發經驗後，應該會發現，寫程式並不難。有時候，一個人寫程式也不難。困難的地方是**團隊合作**。而且，在合作過程中，團隊等於是在共同編輯一份彼此有高度相依性的程式碼。也就是說，團隊共享著一份位於相同 repository 的程式碼，在同樣的基礎上持續的累積成果。

過去，大型團隊常常透過將程式碼切出分支（branch）來進行協作，也就是說，開發人員（團隊）可以依照自己的需要，從主幹（master）上將程式碼複製（其實是分支）一份出去，然後開始修改程式碼，待完成之後，再合併回主幹。

這樣做當然有好處，因為開發的過程當中，我們可以隨時保有一份可運行的原始程式碼（在 Master 主幹上），待分支上修改的程式碼都完成的差不多了，最後再合併（Merge）回主幹上。一方面不怕改壞程式碼，另一方面可以因應不時之需（例如用戶的正式機壞掉了，需要重新部署一份）。

但真實世界並非總是那麼美好，由於 Git 版控進行分支（Branch）太容易了，儲存成本也低，導致初學的開發人員常常隨意的就 branch 開了出去。也有另一種狀況，是開發人員會為了不同運行環境（例如正式機、測試機、開發機…）的需要，從主線上又開了分支出去。

本來沒什麼問題的分支，隨著分支的數量愈來愈多、分支的生命週期愈來愈長，未來造成的問題將愈來愈大，什麼問題？就是「合併」。

不管你最初因為什麼理由切出的分支，終有一天，分支上的程式碼異動必須被合併回主幹。而合併常常是開發人員最痛苦的時候，有時候碰到衝突不打緊，因為切出很多分支，等於是有很多版本，這些不同的版本由不同的開發人員施工，時間一長，開發人員很容易忘了一週前寫的這段 code 到底是跟哪些其他程式碼有關？每個分支又有不同的修改，到底要保留哪一個分支上的 code 才對？還是該怎麼整合幾個分支之間的差異？有時在合併時，發現不同分支上的程式碼似乎根本彼此互斥，完全無法合併成一個版本，得花時間從頭看過程式碼才行。

只要有過這種經驗，你應該會發現，Git 的分支確實好用，但分支切的愈多、分支的生命週期愈長，合併時就可能會產生愈大的災難。合併時所額外花費的大把時間，可能把團隊合作所產生的綜效吃的一乾二淨。

早期很多大型團隊，甚至有所謂的合併日，在每月合併日的當天，把所有程式設計師通通找來會議室，大家盯著螢幕，由其中某位手腳比較快的程式設計師來整合程式碼，其他人盯著看並適時的解釋或提供意見…，花一個下午甚至一整天才能把所有分支上的程式碼給合併起來。

可以想像嗎？倘若團隊合作時，每一次的合併，都需要花上大把的時間，那我們怎麼可能實現一天交付數次或一週交付數次的目標呢？

2-1-2 團隊合作模式，決定了整合將多頻繁

因此，我們先說結論，若要實現愈高強度的 CI/CD，分支的數量得要愈少愈好，分支的生命週期則是愈短愈好。簡單的說，程式碼有多頻繁的整合，系統才有可能多頻繁的交付，沒有頻繁整合，頻繁交付顯然會淪為空談。而團隊採用的版控機制與工作流程，則會成為限制團隊能夠多頻繁整合的主要因素。

舉例來說，如果你的團隊採用 Gitflow（你可以很容易 google 到這種分支策略的圖形），做為團隊的版控與合作方式，對於需要進行高強度（一天數次或一週數次）的 CI/CD 來說很可能是不利的，Gitflow 中的 develop 分支，造成了過

長的生命週期以及未來 Merge 的需求。而相較之下，Github Flow 分支策略則相對較適合 CI/CD。不過，如同我們前面說過的，若真的要進行一週數次甚至一天數次的高強度交付，那程式碼整合勢必也是一天數次。在一天數次整合的狀況下，你還能走 Github Flow 做 Feature Branch 嗎？很難。

會不會，其實沒有多餘分支的 trunk-based development，比較起來更為適合呢？

上述這些，都是團隊合作時可選擇的分支策略，團隊想要實現多大強度的頻繁交付，勢必也得要選擇搭配的分支策略，我們在後面的章節中，將會繼續討論這個部分。

而上面提到的這一切，都是基於 Git 版控。因此，現在我們先回頭來看看 Azure Repos 的使用基礎。

2-2 認識 Azure Repos

如果你是微軟技術的愛好者，你大概之前就知道，Microsoft 早期自己有一套版控機制，是謂 TFVC（Team Foundation Version Control），過去我也是採用這個集中式的版控技術，特別是如果你用 TFS 或早期的 VSTS，那當時確實也都是以 TFVC 為主。

但時間來到了 2015，差不多就是在 2015 年的前後，這幾年微軟發生了很大的改變，從這個時間軸開始，微軟彷彿走入了另一個平行世界。微軟面對社群與 Open Source 的改變愈來愈顯著。微軟在面對外部市場與用戶有所改變的同時，其內部也發生許多質變，這一段歷史我們就不談了，但差不多從這個時間點開始，微軟對版控技術的看法似乎也作了調整。

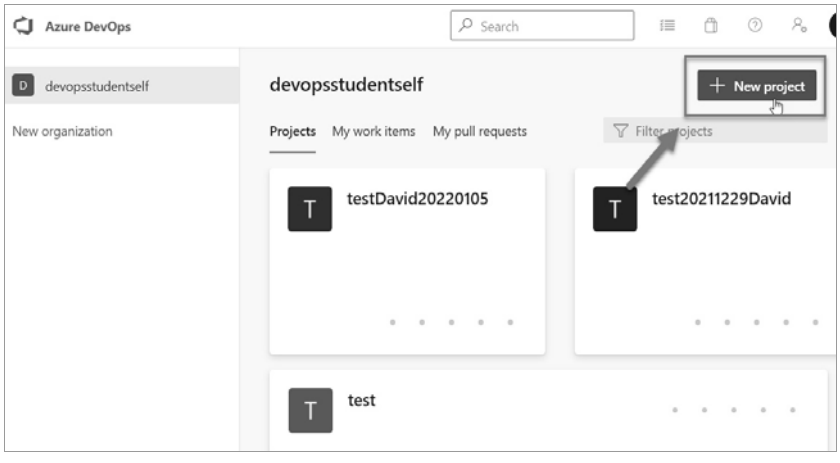
時至今日，微軟自己內部採用的版控技術也大多是使用 Git，而 Azure DevOps 的版控環境中，也陸續的在 Azure Repos 當中，加上了許多 Git 相關的服務和整合，同時在微軟的官方文件上，也不乏介紹如何從 TFVC 轉換到 Git 版控的

方式[1]。今時今日，我們大概幾乎底定是以 Git 版控技術作為未來程式碼版控技術的主軸了。

2-2-1 在 Team Project 中建立 Repos

很多人一開始使用 Azure DevOps 的時候並不知道，其實 Azure DevOps 有內建的 Git Repos，也就是說，你根本無需自行建立 Git 版控環境或伺服器，就可以享有相關的功能。不僅如此，使用 Azure DevOps 內建的 Git Repos 還有諸多的好處，像是可以在看板（Azure Board）的需求卡片（Backlogs）上，直接建立/關聯到版控的 Branch（分支），連帶著讓 PR（Pull Reuqest）串起整個開發流程。千萬別小看這件事。這件事是提升你的程式碼品質和控管的一大關鍵，如果沒有使用這個功能是很可惜的。[2]

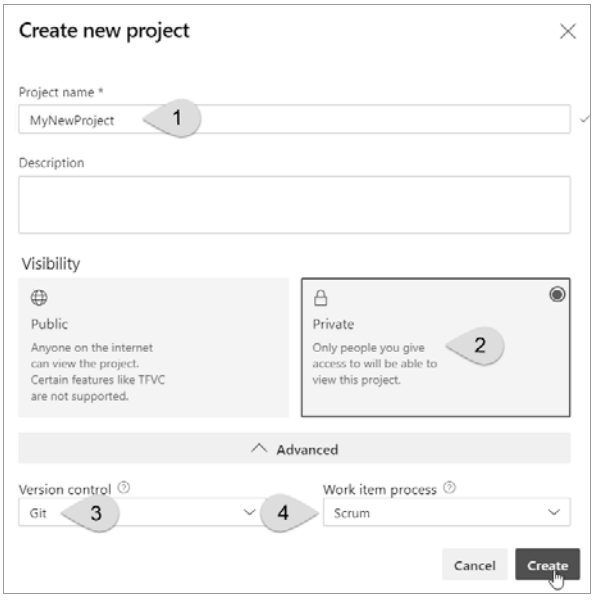
讓我們先從頭開始，你只需要從首頁的左上角，點選「New Project」即可：



1 Migrate from TFVC to Git
https://docs.microsoft.com/zh tw/azure/devops/learn/git/migrate from tfvc to git

2 我知道有一些企業，因為對程式碼的儲存有自己的 Policy，所以並不一定方便存放在雲端的 Git Repos 當中，不過，這個現象愈來愈少見。如今，台灣幾各指標型的大廠，都可以允許把程式碼放到 Azure DevOps 的 Git Repos 中了，我想未來會愈來愈普遍。

在出現的畫面上，請務必點選 Git 版控（下圖 3）：



如此一來，建立出的專案就會預設擁有一個 Git Repository。

我們接著仔細看，當你在 Azure DevOps 中建立好一個新的專案之後，可以點選左方主選單 Repos 中的 Files，你會看到底下畫面：