使用和管理 vRealize Automation Code Stream

2022 年 12 月 14 日

vRealize Automation 8.7

vmware®

您可以在 VMware 網站上找到最新的技術文件，網址如下：

https://docs.vmware.com/tw/

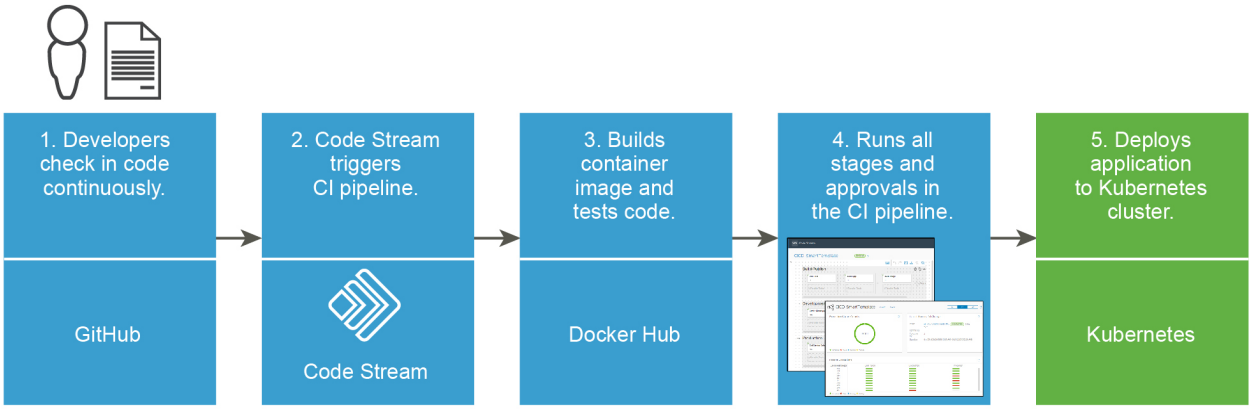
VMware, Inc.
3401 Hillview Ave.
Palo Alto, CA 94304
www.vmware.com

目錄

Code Stream 是什麼以及如何運作

1

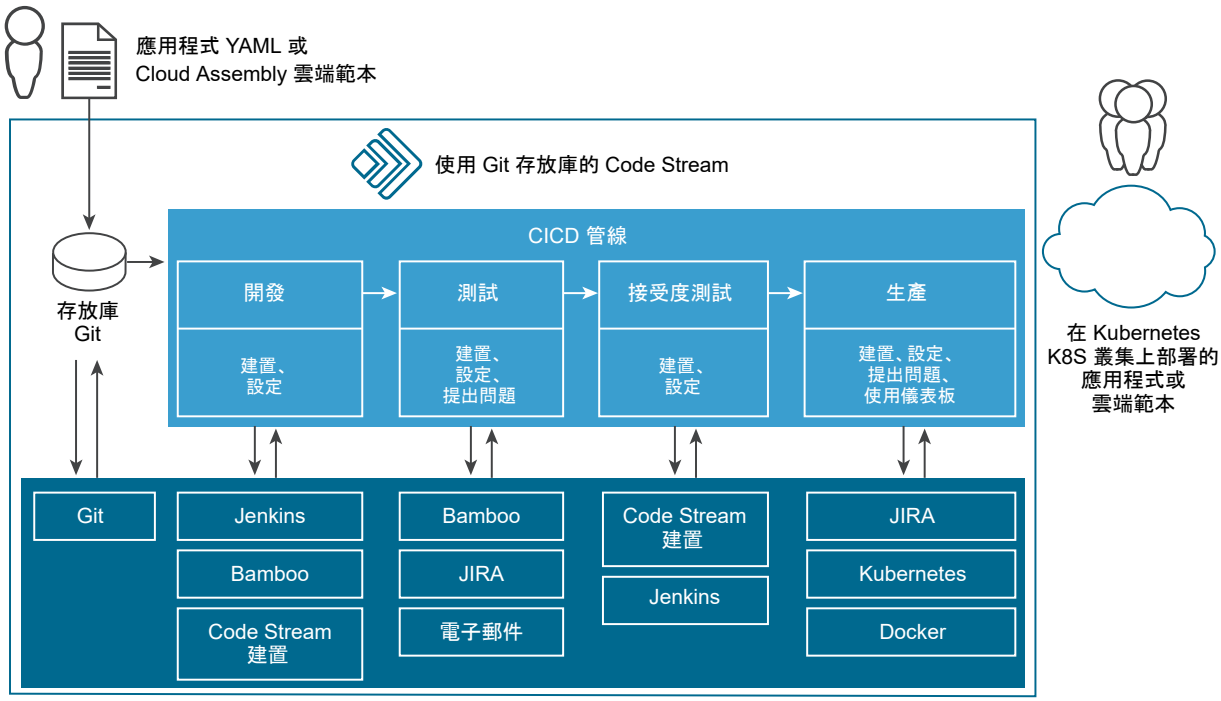
vRealize Automation Code Stream™ 是持續整合和持續交付 (CICD) 工具。透過建立對 DevOps 生命週期中的軟體發行程序進行建模的管線，可以建置代碼基礎結構，以快速且持續地交付軟體。



當您使用 Code Stream 交付軟體時，您可以整合 DevOps 生命週期的兩個最重要的部分：發佈程序和開發人員工具。在初始設定 (將 Code Stream 與現有的開發工具整合) 後，管線會自動執行整個 DevOps 生命週期。

從 vRealize Automation 8.2 開始，藍圖稱為 VMware Cloud Templates。

您可以建立用於建置、測試和發行軟體的管線。 Code Stream 使用該管線從原始程式碼存放庫、透過測試，以及在生產上進行軟體處理。會使用該管線對軟體進行推進，從原始程式碼存放庫到測試，然後再到生產環境。



您可以在第 4 章 在 Code Stream 中規劃原生建置、整合及交付程式碼 中瞭解規劃持續整合和持續交付管線的詳細資訊。

Code Stream 管理員如何使用 Code Stream

身為管理員，您可以建立端點，並確保開發人員可使用工作執行個體。您可以建立、觸發和管理管線等。您擁有 `Administrator` 角色，如如何在 Code Stream 中管理使用者存取和核准中所述。

表 1-1. Code Stream 管理員如何支援開發人員

支援開發人員...	以下是可執行的操作...
提供和管理環境。	為開發人員建立用於測試和部署其程式碼的環境。 ■ 追蹤狀態，並傳送電子郵件通知。 ■ 透過確保開發人員的環境不間斷工作，讓開發人員保持生產力。 若要深入瞭解，請參閱供 Code Stream 管理員和開發人員使用的更多資源。 另請參閱第 5 章 使用 Code Stream 的教學課程。
提供端點。	確保開發人員具有可以連線到其管線的端點的工作執行個體。
提供與其他服務的整合。	確保與其他服務的整合正常運作。 若要深入瞭解，請參閱 VMware Cloud Services 說明文件。
建立管線。	建立用於建立發佈程序模型的管線。 若要深入瞭解，請參閱第 3 章 在 Code Stream 中建立和使用管線。

表 1-1. Code Stream 管理員如何支援開發人員 (續)

支援開發人員...	以下是可執行的操作...
觸發管線。	確保管線在發生事件時執行。 ■ 若要在每當建立或更新建置構件時觸發獨立的持續交付 (CD) 管線，請使用 Docker 觸發器。 ■ 若要在開發人員認可其程式碼變更時觸發管線，請使用 Git 觸發器。 ■ 若要在開發人員檢閱程式碼、執行合併等作業時觸發管線，請使用 Gerrit 觸發器。 ■ 若要在每當建立或更新建置構件時執行獨立的持續交付 (CD) 管線，請使用 Docker 觸發器。 若要深入瞭解，請參閱第 7 章 在 Code Stream 中觸發管線。
管理管線和核准。	在管線上保持最新狀態。 ■ 檢視管線狀態，並查看執行管線的使用者。 ■ 檢視對管線執行的核准，並管理作用中與非作用中管線執行的核准。 若要深入瞭解，請參閱什麼是 Code Stream 中的使用者操作和核准。 另請參閱如何在 Code Stream 中使用自訂儀表板追蹤管線的關鍵效能指標。
監控開發人員環境。	建立自訂儀表板，以監控管線狀態、趨勢、度量和關鍵指標。使用自訂儀表板監控在開發人員環境中通過或失敗的管線。也可以識別和報告未得到充分利用的資源，並釋放資源。 還可以查看： ■ 管線執行多久才能成功。 ■ 管線等待核准需要多長時間，並通知必須予以核准的使用者。 ■ 最常失敗的階段和工作。 ■ 需要最長時間才能執行的階段和工作。 ■ 開發團隊正在進行中的發佈。 ■ 成功部署和發佈的應用程式。 若要深入瞭解，請參閱第 8 章 在 Code Stream 中監控管線。
疑難排解問題。	疑難排解和解決開發人員環境中的管線故障。 ■ 識別和解決持續整合和持續交付環境 (CICD) 中的問題。 ■ 使用管線儀表板，並建立自訂儀表板以查看更多。請參閱第 8 章 在 Code Stream 中監控管線。 另請參閱第 2 章 設定 Code Stream 以建立發佈程序模型。

Code Stream 是 VMware Cloud Services 的一部分。

- 使用 Cloud Assembly 部署雲端範本。
- 使用 Service Broker 從目錄取得雲端範本。

若要瞭解您可以執行的其他動作，請參閱 VMware vRealize Automation 說明文件。

開發人員如何使用 Code Stream

做為開發人員，您可以使用 Code Stream 在儀表板上建置和執行管線，以及監控管線活動。您擁有 `User` 角色，如如何在 Code Stream 中管理使用者存取和核准中所述。

執行管線後，您想要知道：

- 您的代碼是否成功通過管線的所有階段。若要瞭解相關資訊，請觀察管線執行中的結果。

- 如果管線失敗該怎麼辦，以及是什麼導致失敗。若要瞭解相關資訊，請觀察管線儀表板中的常見錯誤。

表 1-2. 使用 Code Stream 的開發人員

整合並發行代碼	以下是您要執行的操作
建置管線。	測試和部署程式碼。 管線發生故障時，請更新您的程式碼。
將管線連線至端點。	將管線中的工作連線至端點，例如 GitHub 存放庫。
執行管線。	新增使用者操作核准工作，以便其他使用者可在特定點核准管線。
檢視儀表板。	在管線儀表板上檢視結果。您可以查看趨勢、歷程記錄、故障等。

如需有關入門的詳細資訊，請參閱《VMware Code Stream 入門》。

在產品內部支援面板中找到更多說明文件

如果您在此處找不到所需的資訊，可以在產品中取得更多說明。 

- 按一下並讀取使用者介面中的路標和工具提示，以便隨時隨地按需取得所需的內容特定資訊。
- 開啟產品內部支援面板，並閱讀針對作用中使用者介面頁面顯示的主題。您也可以在面板中搜尋，以取得問題的答案。

關於 Webhook 的詳細資訊

透過使用相同的 Git 端點並在 Webhook 組態頁面中為分支名稱提供不同的值，可以為不同的分支建立多個 Webhook。若要為相同 Git 存放庫中的其他分支建立其他 Webhook，無需針對多個分支複製 Git 端點多次。而是在 Webhook 中提供分支名稱，以便重複使用 Git 端點。如果 Git Webhook 中的分支與端點中的分支相同，則無需在 Git Webhook 頁面中提供分支名稱。

設定 Code Stream 以建立發佈程序模型

2

若要建立發行程序的模型，您可以建立一個表示您通常用來發行軟體的階段、工作和核准的管線。然後，Code Stream 會自動執行建置、測試、核准及部署程式碼的程序。

現在，您已做好為軟體發行程序建立模型的準備，此處提供了您在 Code Stream 中執行此操作的方式。

必要條件

- 確認是否有任何端點可供使用。在 Code Stream 中，按一下**端點**。
- 瞭解可建置和部署代碼的原生方法。請參閱第 4 章 在 Code Stream 中規劃原生建置、整合及交付程式碼 。
- 判定將在管線中使用的部分資源是否必須標記為受限制。請參閱如何在 Code Stream 中管理使用者存取和核准。
- 如果您有使用者角色或檢視者角色而非管理員角色，請決定誰是 Code Stream 執行個體的管理員。

程序

1 檢查 Code Stream 中的可用專案，然後選取一個適合您的專案。

   - 如果未顯示任何專案，請詢問可以建立專案的 Code Stream 管理員，並將您設為專案成員。請參閱如何在 Code Stream 中新增專案。
   - 如果您不是任何所列專案的成員，請詢問可以將您新增為專案成員的 Code Stream 管理員。

2 新增管線所需的任何新端點。

   例如，您可能需要 Git、Jenkins、Code Stream Build、Kubernetes 以及 Jira。

3 建立變數，以便您可以在管線工作中重複使用值。

若要限制管線中使用的資源，例如主機電腦，請使用受限制的變數。您可以限制管線繼續執行，直到其他使用者明確核准為止。

管理員可以建立密碼變數和受限制的變數。使用者可以建立密碼變數。

您可以跨多個管線多次重複使用此變數。例如，用於定義主機電腦的變數可能為 `HostIPAddress`。若要在管線工作中使用此變數，請輸入 `${var.HostIPAddress}`。

4 如果您是管理員，請將對您的業務至關重要的任何端點和變數標記為受限制的資源。

如果非管理員使用者嘗試執行包含受限制資源的管線，則管線會在使用受限制資源的工作中停止。然後，管理員必須恢復管線。

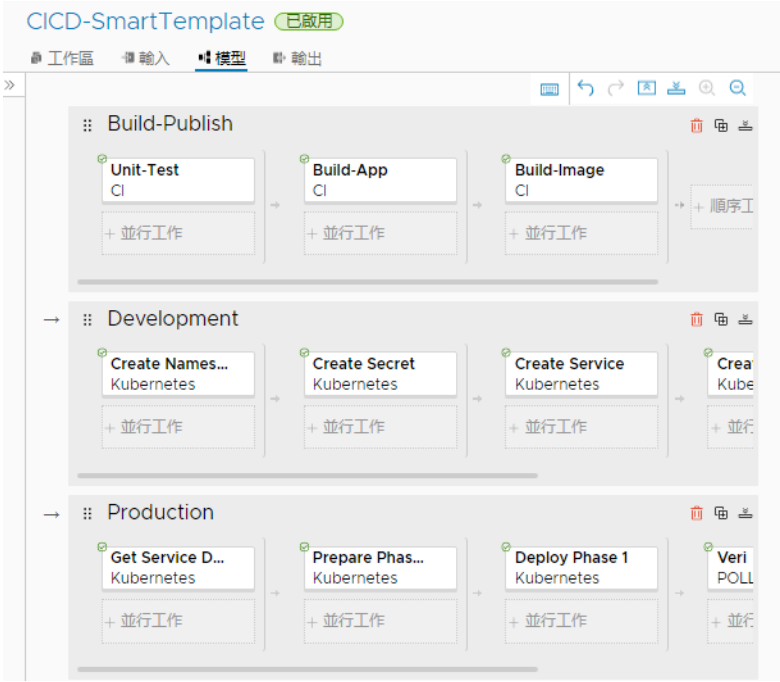
5 規劃原生 CICD、CI 或 CD 管線的建置策略。

建立持續整合 (CI) 和持續部署 (CD) 程式碼的管線之前，請規劃建置策略。建置規劃可協助您判斷 Code Stream 所需的內容，使其能夠在本機上建置、整合、測試和部署程式碼。

如何建立 Code Stream 原生建置	此建置策略中的結果
使用其中一個智慧管線範本。	■ 為您建置所有階段和工作。 ■ 複製來源存放庫。 ■ 建置並測試程式碼。 ■ 將程式碼容器化以進行部署。 ■ 根據您的選取項目填入管線工作步驟。
手動新增階段和工作。	新增階段，新增工作，並輸入資訊以填入其中。

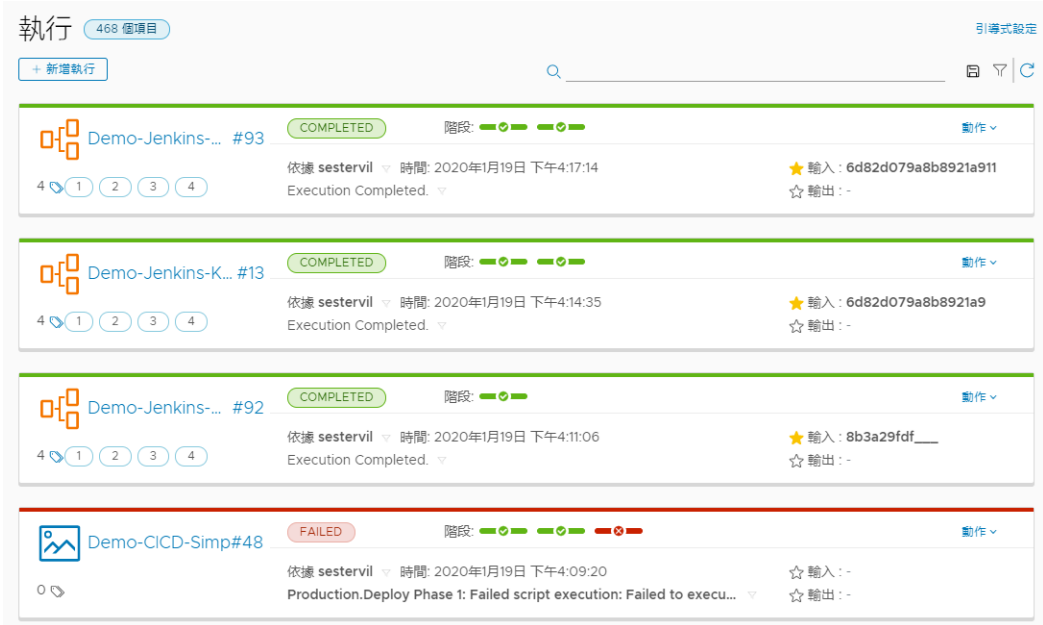
6 使用智慧管線範本建立管線，或手動將階段和工作新增至管線。

然後，將任何資源標記為受限制。視需要新增核准。套用任何一般、受限制或秘密變數。在工作之間新增任何繫結。



7 驗證、啟用並執行管線。

8 驗證管線執行。



9 若要追蹤狀態和關鍵效能指標 (KPI)，請使用管線儀表板，並建立任何自訂儀表板。

結果

您已建立管線，可以在所選專案中使用此管線。

您也可以匯出管線 YAML，然後在其他專案中將其匯入並重複使用。

後續步驟

瞭解可能要在您的環境中套用的使用案例。請參閱第 5 章 使用 Code Stream 的教學課程。

如何在 Code Stream 中新增專案

您可以建立專案，並在此專案中新增管理員和成員。專案成員可以使用建立管線和新增端點等功能。若要建立、刪除或更新開發團隊的專案，您必須是 Code Stream 管理員。

專案必須存在，您才能建立管線。建立管線時，您可以選取將所有管線資訊分組在一起的專案。端點和變數的定義也取決於現有專案。

必要條件

- 確認您具有 Code Stream 管理員角色。請參閱 Code Stream 中有哪些角色。

  如果您沒有 Code Stream 管理員角色，但您是 Cloud Assembly 中的管理員，則可以使用 Cloud Assembly UI 建立、更新或刪除專案。請參閱〈如何為我的 Cloud Assembly 開發團隊新增專案〉

- 如果您要新增 Active Directory 群組至專案，請確認已為您的組織設定了 Active Directory 群組。請參閱〈如何在 vRealize Automation 中為專案啟用 Active Directory 群組〉。如果這些群組未同步，則當您嘗試將其新增至專案時將無法使用。

程序

1 選取**專案**，然後按一下**新增專案**。

2 輸入專案名稱。

3 按一下**建立**。

4 選取新建立的專案的卡，然後按一下**開啟**。

5 按一下**使用者**索引標籤，然後新增使用者並指派角色。

   - 專案管理員可以新增成員。
   - 具有服務角色的專案成員可使用服務。
   - 專案檢視者可查看專案，但無法建立、更新或刪除專案。

   如需有關專案角色的詳細資訊，請參閱如何在 Code Stream 中管理使用者存取和核准。

6 按一下**儲存**。

後續步驟

新增使用該專案的端點和管線。請參閱第 6 章 將 Code Stream 連線至端點和第 3 章 在 Code Stream 中建立和使用管線。

建立管線後，將所有管線資訊分組在一起的專案的名稱會顯示在管線卡和管線執行卡上。

如何在 Code Stream 中管理使用者存取和核准

Code Stream 提供多種方式來確保使用者擁有適當的授權，並同意使用發佈軟體應用程式的管線。

團隊中的每個成員都具有一個指派的角色，該角色提供了有關管線、端點和儀表板的特定權限，以及將資源標記為受限制的功能。

透過使用者作業和核准，您可以控制管線何時執行以及何時必須停止以取得核准。您的角色會決定是否可以恢復管線，並執行包含限制端點或變數的管線。

使用密碼變數隱藏和加密敏感資訊。針對必須隱藏和加密以及限制在執行中使用的字串、密碼和 URL 使用受限制的變數。例如，針對密碼或 URL 使用密碼變數。您可以在管線中任何類型的工作中使用密碼和受限制的變數。

Code Stream 中有哪些角色

根據您在 Code Stream 中的角色，可以執行特定動作並存取特定區域。例如，您的角色可能允許您建立、更新和執行管線。或者，您可能僅具有檢視管線的權限。

除受限制以外的所有動作表示此角色有權對實體執行建立、讀取、更新和刪除動作，但受限制的變數和端點除外。

表 2-1. Code Stream 中的服務和專案層級存取權限

	Code Stream 角色				
存取層級	**Code Stream 管理員**	**Code Stream 開發人員**	**Code Stream 執行者**	**Code Stream 檢視者**	**Code Stream 使用者**
Code Stream 服務層級存取權	所有動作	除受限制以外的所有動作	執行動作	唯讀	無
專案層級存取權：專案管理員	所有動作	所有動作	所有動作	所有動作	所有動作
專案層級存取權：專案成員	所有動作	除受限制以外的所有動作	除受限制以外的所有動作	除受限制以外的所有動作	除受限制以外的所有動作
專案層級存取權：專案檢視者	所有動作	除受限制以外的所有動作	執行動作	唯讀	唯讀

具有專案管理員角色的使用者可以在其作為專案管理員的專案上執行所有動作。

專案管理員可以建立、讀取、更新和刪除管線、變數、端點、儀表板、觸發器，以及啟動包括受限制端點或變數在內的管線，前提是這些資源位於使用者作為專案管理員的專案中。

具有服務檢視者角色的使用者可以查看可供管理員使用的所有資訊。這些使用者無法執行任何動作，除非管理員將其設為專案管理員或專案成員。如果使用者與某個專案相關聯，便擁有與角色相關的權限。專案檢視者不會按照管理員或成員角色的方式來延伸其權限。此角色在所有專案之間都是唯讀的。

如果您在專案中具有讀取權限，則仍可查看受限制的資源。

- 若要查看在端點卡上顯示鎖定圖示的受限制端點，請按一下**設定 > 端點**。
- 若要查看受限制變數和密碼變數 (在**類型**資料行中顯示「受限制」或「密碼」)，請按一下**設定 > 變數**。

表 2-2. Code Stream 服務角色功能

UI 內容	功能	Code Stream 管理員角色	Code Stream 開發人員角色	Code Stream 執行者角色	Code Stream 檢視者角色	Code Stream 使用者角色
管線						
	檢視管線	是	是	是	是	
	建立管線	是	是			
	執行管線	是	是	是		
	執行包含受限制端點或變數的管線	是				
	更新管線	是	是			
	刪除管線	是	是			
管線執行						
	檢視管線執行	是	是	是	是	

表 2-2. Code Stream 服務角色功能 (續)

UI 內容	功能	Code Stream 管理員角色	Code Stream 開發人員角色	Code Stream 執行者角色	Code Stream 檢視者角色	Code Stream 使用者角色
	繼續、暫停和取消管線執行	是	是	是		
	恢復為獲得對受限制資源的核准而停止的管線	是				
自訂整合						
	建立自訂整合	是	是			
	讀取自訂整合	是	是	是	是	
	更新自訂整合	是	是			
端點						
	檢視執行	是	是	是	是	
	建立執行	是	是			
	更新執行	是	是			
	刪除執行	是	是			
將資源標記為受限制						
	將端點或變數標記為受限制	是				
儀表板						
	檢視儀表板	是	是	是	是	
	建立儀表板	是	是			
	更新儀表板	是	是			
	刪除儀表板	是	是			

Code Stream 中的自訂角色和權限

您可以在 Cloud Assembly 中建立自訂角色，將權限延伸至使用管線的使用者。為 Code Stream 管線建立自訂角色時，選取一或多個**管線**權限。

選取將獲指派此自訂角色之使用者所需的最小**管線**權限數目。

將使用者指派給某個專案並在該專案中指定角色時，如果為該使用者指派包含一或多個**管線**權限的自訂角色，則他們可以執行權限允許的所有動作。例如，使用者可以建立受限制的變數、管理受限制的管線、建立和管理自訂整合等。

表 2-3. 可指派給自訂角色的管線權限

管線權限	Code Stream 管理員	Code Stream 開發人員	Code Stream 執行者	Code Stream 檢視者	Code Stream 使用者	專案管理員	專案成員	專案檢視者
管理管線	是	是				是	是	
管理受限制的管線	是					是		
管理自訂整合	是	是						
執行管線	是	是	是			是	是	
執行受限制的管線	是					是		
管理執行	是					是		
讀取。此權限不可見。	是	是	是	是		是	是	是

表 2-4. 如何將管線權限用於自訂角色

權限	可執行的操作
管理管線	■ 建立、更新、刪除、複製管線。 ■ 向 VMware Service Broker 發行和取消發行管線。 ■ 建立、更新和刪除端點。 ■ 建立、更新和刪除一般變數和密碼變數。 ■ 建立、複製、更新和刪除 Gerrit 接聽程式。 ■ 連線和中斷連線 Gerrit 接聽程式。 ■ 建立、複製、更新、刪除 Gerrit 觸發器。 ■ 建立、更新和刪除 Git webhook。 ■ 建立、更新和刪除 Docker webhook。 ■ 使用智慧管線範本來建立管線。 ■ 從 YAML 匯入管線以及將其匯出至 YAML。 ■ 建立、更新和刪除自訂儀表板。 ■ 讀取所有自訂整合。 ■ 讀取所有受限制的端點和變數，但無法檢視其值。
管理受限制的管線	■ 建立、更新和刪除端點。 ■ 將端點標記為受限制、更新受限制的端點，以及刪除這些端點。 ■ 建立、更新和刪除一般變數和密碼變數。 ■ 建立、更新和刪除受限制的變數。 ■ 可以使用「管理管線」執行的所有權限。
管理自訂整合	■ 建立和更新自訂整合。 ■ 版本設定和發行自訂整合。 ■ 刪除和取代自訂整合版本。 ■ 刪除自訂整合。

表 2-4. 如何將管線權限用於自訂角色 (續)

權限	可執行的操作
執行管線	■ 執行管線。 ■ 暫停、繼續和取消管線執行。 ■ 重新執行管線執行。 ■ 繼續、重新執行和手動觸發 Gerrit 觸發器事件。 ■ 核准使用者作業，並且可以對使用者作業執行批次核准。
執行受限制的管線	■ 執行管線。 ■ 暫停、繼續、取消和刪除管線執行。 ■ 重新執行管線執行。 ■ 同步正在執行的管線執行。 ■ 強制刪除正在執行的管線執行。 ■ 繼續、重新執行、刪除和手動觸發 Gerrit 觸發器事件。 ■ 解決受限制的項目和繼續管線執行。 ■ 切換使用者內容並在核准使用者作業工作後繼續管線執行。 ■ 可以使用「執行管線」執行的所有權限。
管理執行	■ 執行管線。 ■ 暫停、繼續、取消和刪除管線執行。 ■ 重新執行管線執行。 ■ 繼續、重新執行、刪除和手動觸發 Gerrit 觸發器事件。 ■ 可以使用「執行管線」執行的所有權限。

自訂角色可包含權限組合。這些權限會組織整理成允許使用者管理或執行管線的功能群組 (受限制和不受限制的資源)。這些權限表示每個角色可在 Code Stream 中執行的所有功能。

例如，如果建立自訂角色並包含稱為**管理受限制的管線**的權限，則具有 Code Stream 開發人員角色的使用者可以：

- 建立、更新和刪除端點。
- 將端點標記為受限制、更新受限制的端點，以及刪除這些端點。
- 建立、更新和刪除一般變數和密碼變數。
- 建立、更新和刪除受限制的變數。

表 2-5. 自訂角色中管線權限的組合範例

指派給自訂角色的權限數目	合併權限的範例	如何使用此組合
單一權限	**執行管線**	
兩項權限	**管理管線**和**執行管線**	
三項權限	**管理管線**、**執行管線**和**執行受限制的管線**	
	管理管線、**管理自訂整合**和**執行受限制的管線**	這種組合可能適用於 Code Stream 開發人員角色，但僅限於使用者所屬的專案。

表 2-5. 自訂角色中管線權限的組合範例 (續)

指派給自訂角色的權限數目	合併權限的範例	如何使用此組合
	管理管線、**管理自訂整合**和**管理執行**	這種組合可能適用於 Code Stream 管理員，但僅限於使用者所屬的專案。
	管理管線、**管理受限制的管線**和**管理自訂整合**	透過此組合，使用者擁有完整權限，並且可以在 Code Stream 中建立和刪除任何內容。

如果您具有管理員角色

身為管理員，您可以建立自訂整合、端點、變數、觸發器、管線和儀表板。

專案允許管線存取基礎結構資源。管理員會建立專案，以便使用者可以將管線、端點和儀表板分組在一起。然後，使用者會在其管線中選取專案。每個專案都包括一個管理員和已指派角色的使用者。

如果您具有管理員角色，您可以將端點與變數標記為受限制資源，並且可以執行使用受限制資源的管線。如果非管理使用者執行包含受限制端點或變數的管線，則管線將在使用受限制變數的工作處停止，且管理員必須繼續執行管線。

作為管理員，您還可以請求在 vRealize Automation Service Broker 中發佈管線。

如果您具有開發人員角色

除了無法使用受限制的端點或變數以外，您可以像管理員一樣使用管線。

如果您執行使用受限制端點或變數的管線，則管線僅會執行使用受限制資源的工作。然後，管線停止，並且 Code Stream 管理員或專案管理員必須繼續執行管線。

如果您具有使用者角色

您可以存取 Code Stream，但不具有其他角色提供的任何權限。

如果您具有檢視者角色

您可以查看管理員所查看的相同資源，例如管線、端點、管線執行、儀表板、自訂整合和觸發器，但無法建立、更新或刪除這些資源。若要執行動作，還必須為檢視者角色指定專案管理員或專案成員角色。

具有檢視者角色的使用者可以查看專案。他們也可以查看受限制的端點和受限制的變數，但無法查看其相關的詳細資訊。

如果您具有執行者角色

您可以執行管線，並對使用者作業工作採取動作。您也可以繼續、暫停和取消管線執行。但是，您無法修改管線。

如何指派和更新角色

您必須是管理員，才能為其他使用者指派和更新角色。

1 若要查看作用中的使用者及其角色，請在 vRealize Automation 中按一下右上方的九個點。

2 按一下**身分識別與存取管理**。

3 若要顯示使用者名稱和角色，請按一下**作用中使用者**。

4 若要為使用者新增角色或變更其角色，請按一下使用者名稱旁邊的核取方塊，然後按一下**編輯角色**。

5 當您新增或變更使用者角色時，還可以新增對服務的存取權。

6 若要儲存變更，請按一下**儲存**。

什麼是 Code Stream 中的使用者操作和核准

使用者操作區域會顯示需要核准的管線執行。所需的核准者可以核准或拒絕管線執行。

當您建立管線時，您可能需要在下列情況中將核准新增至管線：

- 群組成員需要檢閱您的程式碼。
- 另一位使用者需要確認建置構件。
- 您必須確保所有測試已完成。
- 工作會使用管理員標記為受限制的資源，並且該工作需要核准。
- 管線將軟體發行至生產環境。

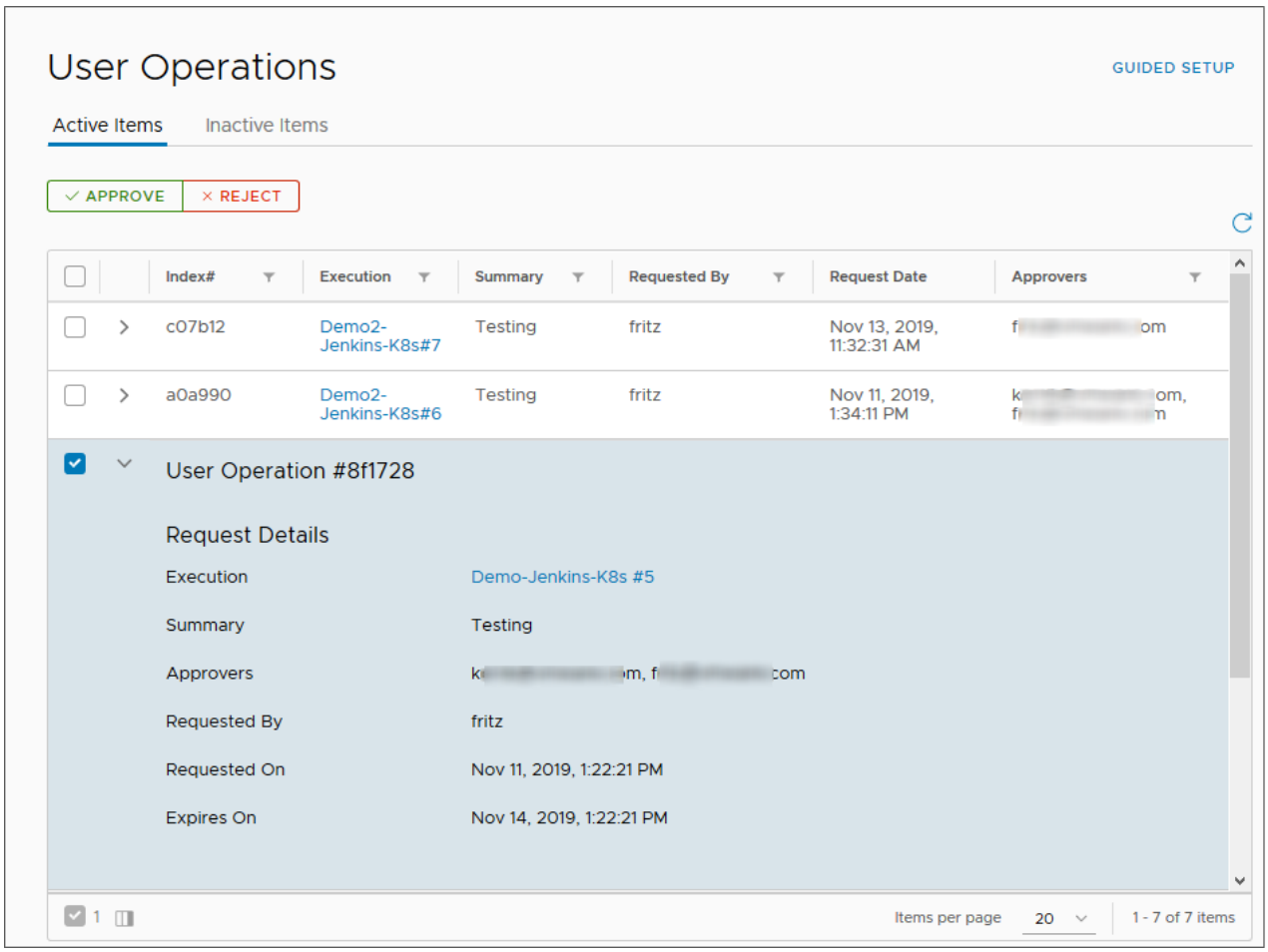
若要確定是否核准管線工作，所需的核准者必須具有權限和專業知識。

新增使用者作業工作時，您可以設定到期逾時 (以天、小時或分鐘為單位)。例如，可能需要所需使用者在 30 分鐘內核准管線。如果未在 30 分鐘內核准，則管線會按預期失敗。

如果啟用了傳送電子郵件通知，使用者作業工作只會將通知傳送至具有完整電子郵件地址的核准者，而不是傳送至未採用電子郵件格式的核准者名稱。

在所需的使用者核准工作後：

- 可繼續執行擱置中的管線。
- 當管線繼續時，將取消任何先前用於核准該相同使用者作業工作的擱置要求。



在 [使用者作業] 區域中，要核准或拒絕的項目會顯示為作用中或非作用中項目。每個項目都會對應至管線中的使用者作業工作。

- **作用中項目**等待必須檢閱工作並核准或拒絕該工作的核准者。如果您是核准者清單中的使用者，則可以展開使用者作業列，然後按一下**接受**或**拒絕**。
- **非作用中項目**已核准或拒絕。如果使用者已拒絕使用者作業，或工作的核准已逾時，則無法再進行核准。

索引編號是唯一的六個英數字元字串，您可以將其用作搜尋特定核准的篩選器。

管線核准也顯示在**執行**區域中。

- 正在等待核准的管線指示其狀態為正在等待。
- 其他狀態包括已排入佇列、已完成和已失敗。
- 如果您的管線處於等待狀態，所需的核准者必須核准管線工作。

在 Code Stream 中建立和使用管線

3

您可以使用 vRealize Automation Code Stream 為建置、測試和部署程序建模。使用 vRealize Automation Code Stream，您可以設定支援發行週期的基礎結構，並建立用於為軟體發行活動建模的管線。vRealize Automation Code Stream 交付從開發程式碼直至測試的軟體，然後將其部署至生產執行個體。

每個管線都包含階段和工作。階段表示您的開發階段，而工作會執行在各個階段交付軟體應用程式所需的動作。

vRealize Automation Code Stream 中的管線是什麼

管線是軟體發行程序的持續整合和持續交付模型。其涵蓋從原始程式碼到測試再到生產環境的完整軟體發行程序。它包括一系列階段，這些階段包括表示軟體發行週期中的活動的工作。軟體應用程式會在管線中從一個階段流向下一個階段。

您可以新增端點，以便管線中的工作可以連線到資料來源、存放庫或通知系統。

建立管線

您可以從空白畫布開始，透過使用智慧管線範本或匯入 YAML 代碼來建立管線。

- 使用空白畫布。如需範例，請參閱手動新增工作前在 Code Stream 中規劃 CICD 原生建置。
- 使用智慧管線範本。如需範例，請參閱第 4 章 在 Code Stream 中規劃原生建置、整合及交付程式碼 。
- 匯入 YAML 程式碼。按一下**管線 > 匯入**。在**匯入**對話方塊中，選取 YAML 檔案或輸入 YAML 代碼，然後按一下**匯入**。

使用空白畫布建立管線時，可以新增階段、工作和核准。管線會自動執行建置、測試、部署及發行應用程式的程序。每個階段中的工作將執行在每個階段建置、測試和發行代碼的動作。

表 3-1. 管線階段和使用範例

階段範例	可執行的操作範例
開發	在開發階段中，您可以佈建機器、擷取構件、新增建置工作以建立要用於持續整合代碼的 Docker 主機等。 例如： ■ 若要規劃和建立持續整合 (CI) 建置，以使用 vRealize Automation Code Stream 中的原生建置功能來傳送程式碼，請參閱使用智慧管線範本前在 Code Stream 中規劃持續整合原生建置。
測試	在測試階段中，您可以新增 Jenkins 工作以測試軟體應用程式，並包含 JUnit 和 JaCoCo 等後續處理測試工具等。 例如： ■ 將 vRealize Automation Code Stream 與 Jenkins 整合，並在管線中執行 Jenkins 工作以建置和測試原始程式碼。請參閱如何整合 Code Stream 與 Jenkins。 ■ 建立用於延伸 vRealize Automation Code Stream 功能的自訂指令碼，以便與您自己的建置、測試和部署工具進行整合。請參閱如何將自己的建置工具、測試工具和部署工具與 Code Stream 整合。 ■ 追蹤持續整合 (CI) 管線的後續處理趨勢。請參閱如何在 Code Stream 中使用自訂儀表板追蹤管線的關鍵效能指標。
生產	在生產階段中，您可以整合 Cloud Assembly 中的雲端範本以佈建基礎結構、將軟體部署到 Kubernetes 叢集等。 例如： ■ 若要查看開發和生產的階段範例 (可在您自己的藍綠部署模型中部署軟體應用程式)，請參閱如何在 Code Stream 中將應用程式部署至藍綠部署。 ■ 若要將雲端範本整合到管線中，請參閱如何在 Code Stream 中自動發行從 YAML 雲端範本部署的應用程式。您也可以新增部署工作，執行指令碼以部署應用程式。 ■ 若要自動將軟體應用程式部署至 Kubernetes 叢集，請參閱如何在 Code Stream 中自動將應用程式發行至 Kubernetes 叢集。 ■ 若要將程式碼整合至管線並部署您的建置映像，請參閱如何在 Code Stream 中將 GitHub 或 GitLab 存放庫中的程式碼持續整合到管線。

您可以將管線匯出為 YAML 檔案。按一下**管線**，接著按一下管線卡，然後按一下**動作 > 匯出**。

核准管線

您可以在管線中的特定點取得其他群組成員的核准。

- 如果需要透過在管線中包含使用者操作工作以核准管線，請參閱如何執行管線和查看結果。此工作會將電子郵件通知傳送給必須進行檢閱的使用者。檢閱者必須核准或拒絕核准，管線才能繼續執行。如果使用者作業工作設定了到期逾時 (以天、小時或分鐘為單位)，則所需使用者必須在工作到期前核准管線。否則，管線會按預期失敗。
- 在管線的任何階段中，如果工作或階段失敗，您可以讓 vRealize Automation Code Stream 建立 Jira 票證。請參閱如何在管線工作失敗時在 Code Stream 中建立 Jira 票證。

觸發管線

當開發人員將其代碼簽入存放庫或檢閱代碼時，或在識別出新增或更新的建置構件時，可能會觸發管線。

- 若要將 vRealize Automation Code Stream 與 Git 生命週期整合並在開發人員更新其程式碼時觸發管線，請使用 Git 觸發器。請參閱如何在 Code Stream 中使用 Git 觸發器執行管線。
- 若要將 vRealize Automation Code Stream 與 Gerrit 程式碼檢閱生命週期整合並在程式碼檢閱時觸發管線，請使用 Gerrit 觸發器。請參閱如何在 Code Stream 中使用 Gerrit 觸發器執行管線。
- 若要在建立或更新 Docker 建置構件時觸發管線，請使用 Docker 觸發器。請參閱如何在 Code Stream 中使用 Docker 觸發器執行持續交付管線。

如需有關 vRealize Automation Code Stream 所支援之觸發器的詳細資訊，請參閱第 7 章 在 Code Stream 中觸發管線。

本章節討論下列主題：

- 如何執行管線和查看結果
- 在 Code Stream 中提供哪些類型的工作
- 如何在 Code Stream 管線中使用變數繫結
- 如何在 Code Stream 中使用 [條件] 工作中的變數繫結來執行或停止管線
- 在 Code Stream 中繫結管線工作時可以使用哪些變數和運算式
- 如何在 Code Stream 中傳送有關我的管線的通知
- 如何在管線工作失敗時在 Code Stream 中建立 Jira 票證
- 如何在 Code Stream 中復原我的部署

如何執行管線和查看結果

您可以從管線卡、在管線編輯模式下，以及在管線執行中執行管線。您也可以在發生特定事件時使用可用的觸發器讓 Code Stream 執行管線。

當管線中的所有階段與工作均有效時，管線即做好發佈、執行或觸發的準備。

若要使用 Code Stream 來執行或觸發管線，則可以從管線卡啟用和執行管線，也可以在管線中啟用和執行管線。然後，您可以檢視管線執行，以確認管線已建置、測試和部署您的代碼。

當管線執行正在進行時，如果您是管理員或非管理員使用者，則可以刪除執行。

- 管理員：若要在執行時刪除管線，請按一下**執行**。在要刪除的執行上，按一下**動作 > 刪除**。
- 非管理員使用者：若要刪除正在執行的管線執行，請按一下**執行**，然後按一下 **Alt Shift d**。

當管線執行正在進行中且似乎停滯時，管理員可以從 [執行] 頁面或 [執行詳細資料] 頁面重新整理執行。

- 執行頁面：按一下**執行**。在要重新整理的執行上，按一下**動作 > 同步**。
- 執行詳細資料頁面：按一下**執行**，按一下執行詳細資料的連結，然後按一下**動作 > 同步**。

若要在發生特定事件時執行管線，請使用觸發器。

- Git 觸發器可以在開發人員更新代碼時觸發管線。
- Git 觸發器可以在進行代碼檢閱時執行管線。
- Docker 觸發器可以在 Docker 登錄中建立構件時執行管線。
- 在 Jenkins 建置完成後，`curl` 命令或 `wget` 命令會使 Jenkins 執行管線。

如需有關使用觸發器的詳細資訊，請參閱第 7 章 在 Code Stream 中觸發管線。

下列程序顯示如何從管線卡執行管線、檢視執行、查看執行詳細資料，以及使用動作。此外，還會顯示如何釋放管線，以便將其新增至 vRealize Automation Service Broker。

必要條件

- 確認已建立一或多個管線。請參閱第 5 章 使用 Code Stream 的教學課程中的範例。

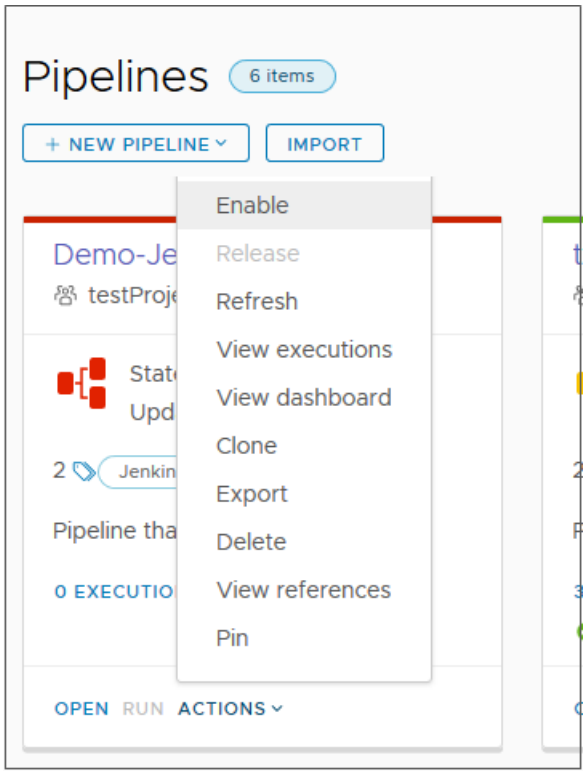
程序

1 啟用管線。

若要執行或發佈管線，您必須先將其啟用。

a 按一下**管線**。

b 在管線卡上，按一下**動作 > 啟用**。



您也可以在管線中啟用管線。如果您的管線已啟用，**執行**會處於作用中狀態，並且**動作**功能表會顯示**停用**。

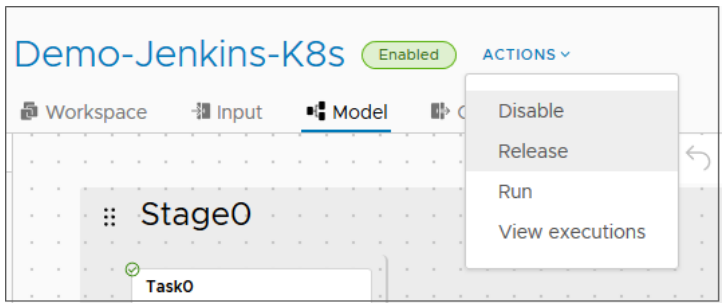
2 (選擇性) 發佈管線。

如果您想要讓管線在 vRealize Automation Service Broker 中做為目錄項目使用，則必須在 Code Stream 中將其發佈。

a 按一下**管線**。

b 在管線卡上，按一下**動作 > 發佈**。

您也可以在管線中發佈管線。



發佈管線後，您可以開啟 Service Broker，以將管線新增為目錄項目並加以執行。請參閱〈將 Code Stream 管線新增至 Service Broker 目錄〉。

備註 如果管線需要 120 分鐘以上的時間來執行，請提供大約的執行時間作為請求逾時值。若要設定或檢閱專案的請求逾時，請以管理員身分開啟 Service Broker，然後選取**基礎結構 > 專案**。按一下專案名稱，然後按一下**佈建**。

如果未設定請求逾時值，則需要 120 分鐘以上的時間才能執行的執行會顯示為失敗，並顯示回撥逾時請求錯誤。但是，管線執行不受影響。

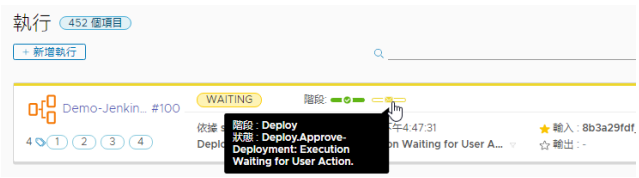
3 在管線卡上按一下**執行**。

4 若要檢視執行的管線，按一下**執行**。

管線會依序執行每個階段，並且管線執行會顯示各個階段的狀態圖示。如果管線包含使用者操作工作，則使用者必須核准此工作才能繼續執行管線。使用使用者操作工作時，管線會停止執行，並等待所需使用者核准工作。

例如，您可以使用使用者操作工作，核准將程式碼部署到生產環境。

如果使用者作業工作設定了到期逾時 (以天、小時或分鐘為單位)，則所需使用者必須在工作到期前核准管線。否則，管線會按預期失敗。



5 若要查看正在等待使用者核准的管線階段，請按一下此階段的狀態圖示。

6 若要查看工作的詳細資料，請按一下工作。

所需使用者核准工作後，具有相應角色的使用者必須繼續執行管線。如需必要角色的相關資訊，請參閱如何在 Code Stream 中管理使用者存取和核准。

如果執行失敗，您必須進行分類並修正失敗的原因。然後，移至執行，按一下**動作 > 重新執行**。

可以恢復主要管線執行和巢狀執行。

7 從管線執行中，您可以按一下**動作**來檢視管線，然後選取**暫停**、**取消**等動作。當管線執行正在進行時，如果您是管理員，則可以刪除或同步管線執行。如果您是非管理員使用者，則可以刪除正在執行的管線。

8 若要輕鬆地在執行之間導覽並查看工作的詳細資料，請按一下**執行**，再按一下管線執行。然後，按一下頂端的索引標籤並選取管線執行。

結果

恭喜您！您已執行管線、檢查管線執行，並檢視需要核准才能繼續執行管線的使用者操作工作。您還使用了管線執行中的**動作**功能表以返回管線模型，以便能夠進行任何必要的變更。

後續步驟

若要瞭解有關使用 Code Stream 自動執行軟體發佈週期的詳細資訊，請參閱第 5 章 使用 Code Stream 的教學課程。

在 Code Stream 中提供哪些類型的工作

設定管線時，您可以新增管線針對所需動作執行的特定工作類型。每個工作類型均與另一個應用程式整合，並且在建置、測試和交付應用程式時啟用管線。

若要執行管線，無論您必須從存放庫提取構件以進行部署、執行遠端指令碼，還是需要團隊成員對使用者作業進行核准， Code Stream 都具有適當的工作類型供您選取！

Code Stream 支援取消各種類型工作上的管線執行。在管線執行上按一下**取消**時，工作、階段或整個管線將進入取消狀態並取消管線執行。

在 Code Stream 中，可以在使用以下工作時取消工作、階段或整個管線上的管線執行：

- Jenkins
- SSH
- PowerShell
- 使用者作業
- 管線
- 雲端範本
- vRO
- 輪詢

Code Stream 不會將取消行為散佈到這些工作的第三方系統：CI、自訂整合或 Kubernetes。 Code Stream 將工作標記為已取消並立即停止擷取狀態，而不會等待工作完成。按一下**取消**時，工作可能會在第三方系統上完成或失敗，但會在 Code Stream 中立即停止執行。

在管線中使用工作之前，請確認對應的端點可用。

表 3-2. 取得核准或設定決定點

工作類型	執行的作業	範例和詳細資料
使用者作業	使用者作業工作會啟用所需的核准，以控制何時執行管線以及何時必須等待核准。	請參閱如何執行管線和查看結果。和如何在 Code Stream 中管理使用者存取和核准。
條件	新增決定點，以根據條件運算式確定繼續執行或停止管線。條件為 true 時，管線執行連續工作。條件為 false 時，管線將停止。	請參閱如何在 Code Stream 中使用 [條件] 工作中的變數繫結來執行或停止管線。

表 3-3. 自動執行持續整合和部署

工作類型	執行的作業	範例和詳細資料
雲端範本	從 GitHub 部署自動執行雲端範本，並佈建應用程式，針對您的部署自動執行該雲端範本的持續整合和持續交付 (CICD)。	請參閱如何在 Code Stream 中自動發行從 YAML 雲端範本部署的應用程式。 依序選取**建立**或**更新**、**雲端範本**和**版本**後，雲端範本參數隨即出現。您可以將可容納變數繫結的這些元素新增至雲端範本工作中的輸入文字區域： ■ 整數 ■ 列舉字串 ■ 布林值 ■ 陣列變數 當您在輸入中使用變數繫結時，請注意這些例外狀況。對於列舉，您必須從固定集中選取一個列舉值。對於布林值，您必須在輸入文字區域中輸入值。 當 Cloud Assembly 中的雲端範本包含輸入變數時，雲端範本參數會出現在雲端範本工作中。例如，如果雲端範本具有輸入類型 `Integer`，您可以直接輸入整數或使用變數繫結來輸入整數以作為變數。
CI	透過從登錄端點提取 Docker 建置映像並將其部署至 Kubernetes 叢集，CI 工作能夠將代碼持續整合到管線中。 CI 工作將記錄的 100 行顯示為輸出，在下載記錄時顯示 500 行。 CI 工作需要暫時連接埠 32768 到 61000。	請參閱使用智慧管線範本前在 Code Stream 中規劃 CICD 原生建置。
自訂	自訂工作將 Code Stream 與您自己的建置工具、測試工具和部署工具整合。	請參閱如何將自己的建置工具、測試工具和部署工具與 Code Stream 整合。
Kubernetes	自動將軟體應用程式部署到 AWS 上的 Kubernetes 叢集。	請參閱如何在 Code Stream 中自動將應用程式發行至 Kubernetes 叢集。
管線	將管線嵌套於主要管線中。嵌套管線後，此管線將作為主要管線中的工作。 在主要管線的 [工作] 索引標籤上，您可以透過按一下巢狀管線的連結輕鬆導覽至該管線。巢狀管線將在新的瀏覽器索引標籤中開啟。	若要在**執行**中尋找巢狀管線，請在搜尋區域中輸入 **`nested`**。

表 3-4. 整合開發、測試和部署應用程式

工作類型...	它執行的作業...	範例和詳細資料...
Bamboo	與 Bamboo 持續整合 (CI) 伺服器互動，該伺服器持續建置、測試和整合軟體，為部署準備就緒，並在開發人員認可變更時觸發程式碼建置。它會公開 Bamboo 建置產生的構件位置，讓工作可輸出用於建置和部署的其他工作的參數。	連線至 Bamboo 伺服器端點，並從管線啟動 Bamboo 建置計劃。
Jenkins	觸發建置和測試原始程式碼的 Jenkins 工作，執行測試案例，並且可以使用自訂指令碼。	請參閱如何整合 Code Stream 與 Jenkins。

表 3-4. 整合開發、測試和部署應用程式 (續)

工作類型...	它執行的作業...	範例和詳細資料...
TFS	允許您將管線連線至 Team Foundation Server，以便管理和叫用建置專案，包括可建置和測試程式碼的已設定工作。	如需 Code Stream 支援的 Team Foundation Server 版本，請參閱 什麼是 Code Stream 中的端點 。
vRO	透過在 vRealize Orchestrator 中執行預先定義的工作流程或自訂工作流程，來延伸 Code Stream 的功能。 Code Stream 支援 vRealize Orchestrator 的基本驗證和以 Token 為基礎的驗證。 Code Stream 使用 API Token 對 vRealize Orchestrator 叢集進行驗證。透過以 Token 為基礎的驗證， Code Stream 支援使用雲端擴充性 Proxy 的 vRealize Orchestrator 端點。因此，在 Code Stream 中，您可以透過使用雲端擴充性 Proxy 的 vRealize Orchestrator 端點觸發工作流程。	請參閱如何將 Code Stream 與 vRealize Orchestrator 整合。

表 3-5. 透過 API 整合其他應用程式

工作類型...	它執行的作業...	範例和詳細資料...
REST	將 Code Stream 與其他使用 REST API 的應用程式整合，可讓您持續開發和交付彼此互動的軟體應用程式。	請參閱如何使用 REST API 將 Code Stream 與其他應用程式整合。
輪詢	叫用 REST API，並對其進行輪詢，直到管線工作滿足結束準則並完成為止。 Code Stream 管理員最多可以將輪詢計數設定為 10000。輪詢時間間隔必須大於或等於 60 秒。 選取**失敗時繼續**核取方塊時，如果計數或時間間隔超過這些值，輪詢工作將繼續執行。 `POLL Iteration Count`：顯示在管線執行中，並顯示 POLL 工作從 URL 請求回應的次數。例如，如果 POLL 輸入為 65，而 POLL 請求執行的實際次數為 4，則管線執行輸出中的反覆運算計數將顯示 4 (共 65 次)。	請參閱如何使用 REST API 將 Code Stream 與其他應用程式整合。

表 3-6. 執行遠端指令碼和使用者定義的指令碼

工作類型	執行的作業	範例和詳細資料
PowerShell	透過 PowerShell 工作， Code Stream 可以在遠端主機上執行指令碼命令。例如，指令碼可以自動執行測試工作，並執行管理類型的命令。 指令碼可以是遠端指令碼，也可以是使用者定義的指令碼。它可以透過 HTTP 或 HTTPS 進行連線，並且可以使用 TLS。 Windows 主機必須已設定 `winrm` 服務，並且 `winrm` 必須已設定 `MaxShellsPerUser` 和 `MaxMemoryPerShellMB`。 若要執行 PowerShell 工作，您必須具有與遠端 Windows 主機的作用中工作階段。 **PowerShell 命令列長度** 如果輸入 base64 PowerShell 命令，請務必計算整體命令長度。 Code Stream 管線會對 base64 PowerShell 命令進行編碼並包裝在另一個命令中，這會增加命令的整體長度。 PowerShell `winrm` 命令允許的長度上限為 8192 個位元組。對 PowerShell 工作進行編碼和包裝後，其命令長度限制會降低。因此，您必須先計算命令長度，然後才能輸入 PowerShell 命令。 Code Stream PowerShell 工作的命令長度限制取決於原始命令的 base64 編碼長度。命令長度的計算方式如下。 `3 * (length of original command / 4)) - (numberOfPaddingCharacters) + 77 (Length of Write-output command)` Code Stream 的命令長度必須小於 8192 的上限。	設定 `MaxShellsPerUser` 和 `MaxMemoryPerShellMB` 時： ■ `MaxShellsPerUser` 可接受的值為 `500` (50 個並行管線)，每個管線 5 個 PowerShell 工作。若要設定值，請執行：`winrm set winrm/config/winrs '@{MaxShellsPerUser="500"}'` ■ `MaxMemoryPerShellMB` 可接受的記憶體值為 `2048`。若要設定值，請執行：`winrm set winrm/config/winrs '@{MaxMemoryPerShellMB="2048"}'` 指令碼會將輸出寫入至其他管線可以耗用的回應檔案中。
SSH	SSH 工作允許 Bash Shell 指令碼工作在遠端主機上執行指令碼命令。例如，指令碼可以自動執行測試工作，並執行管理類型的命令。 指令碼可以是遠端指令碼，也可以是使用者定義的指令碼。它可以透過 HTTP 或 HTTPS 進行連線，且需要私密金鑰或密碼。 必須在 Linux 主機上設定 SSH 服務，而且 `MaxSessions` 的 SSHD 組態必須設定為 `50`。 如果同時執行多個 SSH 工作，請增加 SSH 主機上的 `MaxSessions` 和 `MaxOpenSessions`。如果需要修改 MaxSessions 和 MaxOpenSessions 組態設定，請勿使用 vRealize Automation 執行個體作為 SSH 主機。	指令碼可以是遠端指令碼，也可以是使用者定義的指令碼。例如，指令碼可能類似於： `message="Hello World" echo $message` 指令碼會將輸出寫入至其他管線可以耗用的回應檔案中。

如何在 Code Stream 管線中使用變數繫結

繫結管線工作表示在管線執行時建立了工作的相依性。您可以透過多種方式建立管線工作的繫結。您可以將某個工作繫結到其他工作，將其繫結到變數和運算式，或將其繫結到一個條件。

如何在雲端範本工作中將貨幣繫結套用至雲端範本變數

您可以在 Code Stream 管線雲端範本工作中將貨幣繫結套用至雲端範本變數。在 Code Stream 中修改變數的方式，取決於雲端範本中變數內容的編碼。

如果必須在雲端範本工作中使用貨幣繫結，但在雲端範本工作中使用的目前雲端範本版本不允許使用，請在 Cloud Assembly 中修改雲端範本並部署新版本。然後，在雲端範本工作中使用新雲端範本版本，並在需要時新增貨幣繫結。

若要在 Cloud Assembly 雲端範本提供的內容類型上套用貨幣繫結，您必須擁有正確的權限。

- 您必須與在 Cloud Assembly 中建立雲端範本部署的人員具有相同的角色。
- 管線建模人員和管線執行人員可能是兩個不同的使用者，並且可能具有不同的角色。
- 如果開發人員具有 Code Stream 執行者角色並對管線進行建模，則開發人員還必須具有雲端範本部署人員的相同 Cloud Assembly 角色。例如，必要的角色可能為 Cloud Assembly 管理員。
- 只有管線建模人員才能建立管線和建立部署，因為他們具有權限。

在雲端範本工作中使用 API Token：

- 管線建模人員可以將 API Token 提供給具有 Code Stream 執行者角色的其他使用者。然後，當執行者執行管線時，將會使用 API Token 和 API Token 建立的認證。
- 當使用者在雲端範本工作中輸入 API Token 時，將會建立管線所需的認證。
- 若要加密 API Token 值，請按一下**建立變數**。
- 如果您沒有為 API Token 建立變數，並在雲端範本工作中使用，則 API Token 值會以純文字形式顯示。

若要在雲端範本工作中將貨幣繫結套用至雲端範本變數，請執行下列步驟。

從已定義輸入變數內容的雲端範本開始，例如 `integerVar`、`stringVar`、`flavorVar`、`BooleanVar`、`objectVar` 和 `arrayVar`。您可以找到在 `resources` 區段中定義的映像內容。雲端範本代碼中的內容可能類似於：

```
formatVersion: 1
inputs:
  integerVar:
    type: integer
    encrypted: false
    default: 1
  stringVar:
    type: string
    encrypted: false
    default: bkix
  flavorVar:
    type: string
    encrypted: false
    default: medium
  BooleanVar:
    type: boolean
    encrypted: false
    default: true
```

```
  objectVar:
    type: object
    encrypted: false
    default:
      bkix2: bkix2
  arrayVar:
    type: array
    encrypted: false
    default:
      - '1'
      - '2'
resources:
  Cloud_Machine_1:
    type: Cloud.Machine
    properties:
      image: ubuntu
      flavor: micro
      count: '${input.integerVar}'
```

您可以將貨幣符號變數 ($) 用於 `image` 和 `flavor`。例如：

```
resources:
  Cloud_Machine_1:
    type: Cloud.Machine
    properties:
      input: '${input.image}'
      flavor: '${input.flavor}'
```

若要在 Code Stream 管線中使用雲端範本並向其新增貨幣繫結，請依照下列步驟進行操作。

1 在 Code Stream 中，按一下**管線 > 空白畫布**。

2 將**雲端範本**工作新增至管線。

3 在雲端範本工作中，針對**雲端範本來源**選取 **Cloud Assembly 雲端範本**，輸入雲端範本名稱，然後選取雲端範本版本。

4 請注意，您可以輸入為管線提供認證的 API Token。若要建立在雲端範本工作中加密 API Token 的變數，請按一下**建立變數**。

5 在顯示的**參數和值**資料表中，請注意參數值。`flavor` 的預設值為 `small`，`image` 的預設值為 `ubuntu`。

6 假設您必須在 Cloud Assembly 中變更雲端範本。例如：

   a 對 `flavor` 進行設定，使其使用 `array` 類型的內容。當類型為 **`array`** 時，Cloud Assembly 允許對 `Flavor` 使用逗號分隔值。

   b 按一下**部署**。

   c 在 [部署類型] 頁面上，輸入部署名稱，然後選取雲端範本的版本。

   d 在 [部署輸入] 頁面上，您可以為 `Flavor` 定義一或多個值。

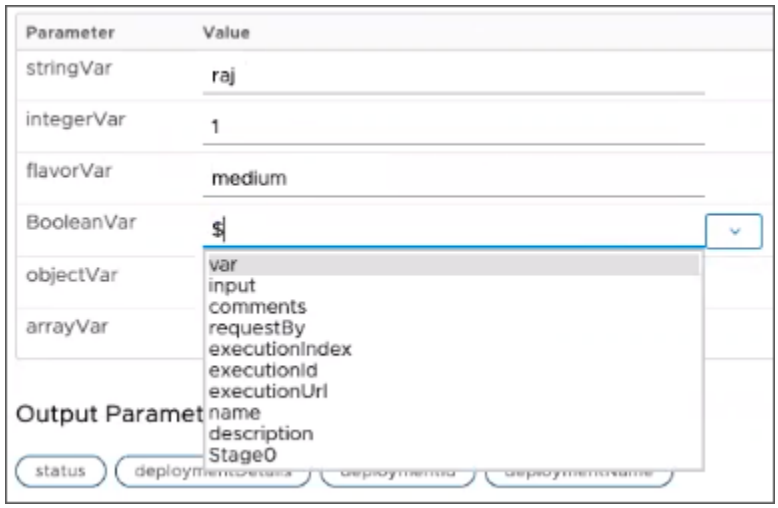
e 請注意，部署輸入包括在雲端範本代碼中定義的所有變數，並且會在雲端範本代碼中顯示為已定義。例如：`Integer Var`、`String Var`、`Flavor Var`、`Boolean Var`、`Object Var` 和 `Array Var`。`String Var` 和 `Flavor Var` 為字串值，而 `Boolean Var` 為核取方塊。

f 按一下**部署**。

7 在 Code Stream 中，選取新版本的雲端範本，然後在**參數和值**資料表中輸入值。雲端範本支援下列類型的參數，這些參數可使用貨幣符號變數啟用 Code Stream 繫結。 Code Stream 雲端範本工作的使用者介面與 Cloud Assembly 雲端範本的使用者介面之間稍有差異。根據 Cloud Assembly 中雲端範本的編碼，可能不允許在 Code Stream 的雲端範本工作中輸入值。

a 對於 **flavorVar**，如果雲端範本已將類型定義為字串或陣列，請輸入字串或以逗號分隔的值陣列。範例陣列類似於 **`test, test`**。

b 對於 **BooleanVar**，請在下拉式功能表中選取 **true** 或 **false**。或者，若要使用變數繫結，請輸入 **`$`**，然後從清單中選取變數繫結。



c 對於 **objectVar**，輸入以大括號和引號括住的值，格式如下：**`{"bkix":"bkix":}`**。

d **objectVar** 將傳遞至雲端範本，並且可以多種方式加以使用，具體取決於雲端範本。允許對 JSON 物件使用字串格式，您可以在索引鍵-值資料表中以逗號分隔值形式新增索引鍵-值配對。可以為 JSON 物件輸入純文字，也可以輸入索引鍵-值配對作為 JSON 的一般字串化格式。

e 對於 **arrayVar**，輸入以逗號分隔的輸入值作為陣列，格式如下：**`["1","2"]`**。

8 在管線中，您可以將輸入參數繫結至陣列。

a 按一下**輸入**索引標籤。

b 輸入輸入的名稱。例如，**`arrayInput`**。

c 在**參數和值**資料表中，按一下 **arrayVar**，然後輸入 **`${input.arrayInput}`**。

d 儲存管線並將其啟用後，必須在管線執行時提供陣列輸入值。例如，輸入 **`["1","2"]`**，然後按一下**執行**。

現在，您已瞭解如何在 Code Stream 管線雲端範本工作的雲端範本中使用貨幣符號 ($) 變數繫結。

如何在管線執行時將參數傳遞到管線

您可以將輸入參數新增至管線，讓 Code Stream 將其傳遞到管線。然後，當管線執行時，使用者必須輸入輸入參數的值。將輸出參數新增至管線時，管線工作可以使用工作的輸出值。 Code Stream 支援以多種方式使用參數，以支援您自己的管線需求。

例如，若要在包含 REST 工作的管線執行時提示使用者輸入其 Git 伺服器的 URL，您可以將 REST 工作繫結至 Git 伺服器 URL。

若要建立變數繫結，請將 URL 繫結變數新增至 REST 工作。當管線執行並到達 REST 工作時，使用者必須輸入其 Git 伺服器的 URL。以下是建立繫結的方式：

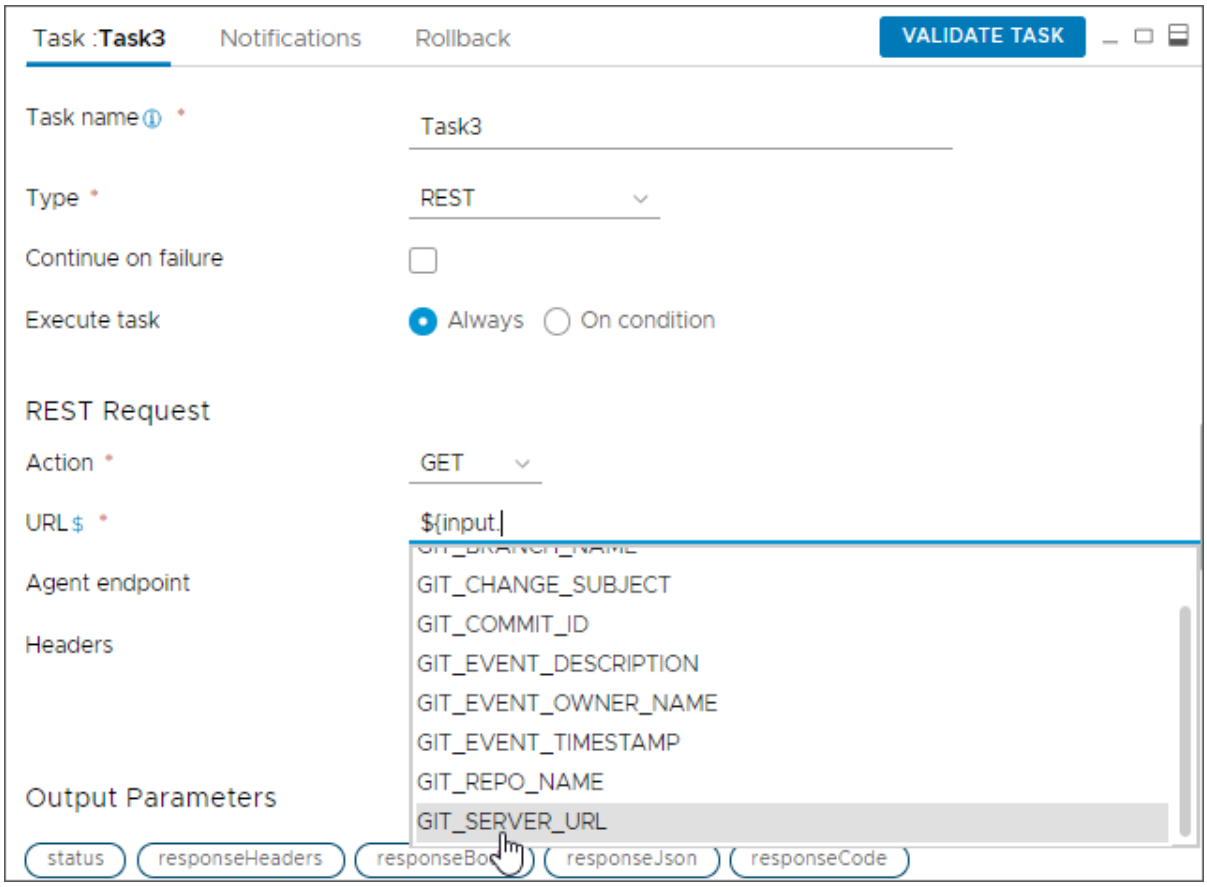
1 在您的管線中，按一下**輸入**索引標籤。

2 若要設定參數，請針對**自動插入參數**按一下 **Git**。

此時會顯示 Git 參數的清單，並包含 **GIT_SERVER_URL**。如果必須使用 Git 伺服器 URL 的預設值，請編輯此參數。

3 按一下**模型**，然後按一下 REST 工作。

4 在**工作**索引標籤的 **URL** 區域中，輸入 **$**，然後選取**輸入**和 **GIT_SERVER_URL**。



此項目類似於：**${input.GIT_SERVER_URL}**

5 若要確認工作的變數繫結完整性，請按一下**驗證工作**。

Code Stream 指示工作已成功驗證。

6 當管線執行 REST 工作時，使用者必須輸入 Git 伺服器的 URL。否則，此工作無法完成執行。

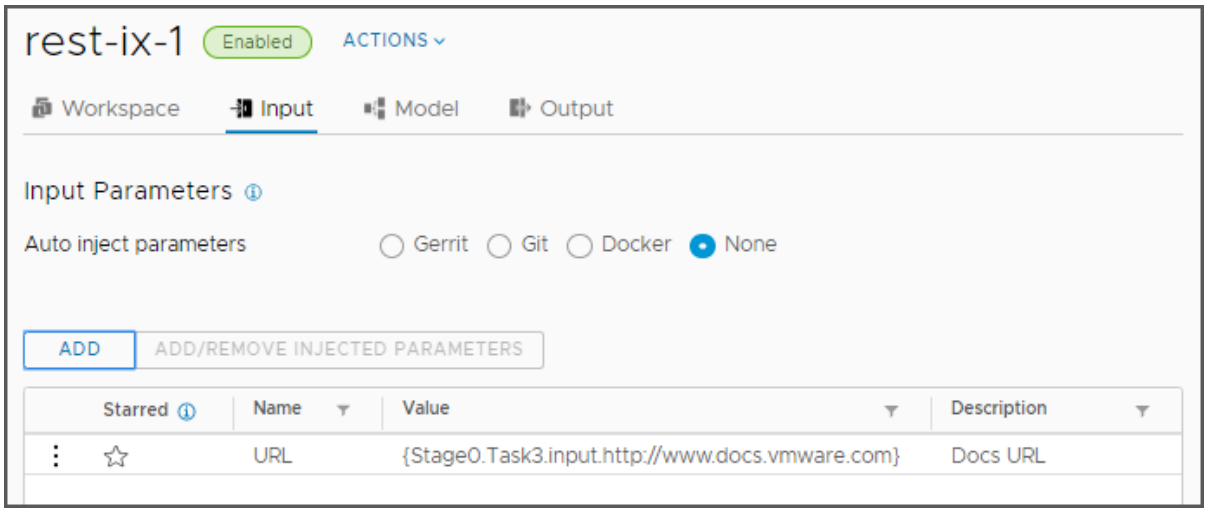
如何建立輸入和輸出參數以繫結兩個管線工作

將工作繫結在一起時，請將繫結變數新增至接收工作的輸入組態中。然後，當管線執行時，使用者將繫結變數取代為所需的輸入。

若要將管線工作繫結在一起，請在輸入參數和輸出參數中使用貨幣符號變數 ($)。此範例顯示了操作方法。

假設您需要管線來呼叫 REST 工作中的 URL，並輸出回應。若要呼叫 URL 並輸出回應，則需要在 REST 工作中同時包含輸入和輸出參數。同時，還需要可以核准工作的使用者，並為可以在管線執行時進行核准的其他使用者包括使用者作業工作。此範例顯示如何在輸入和輸出參數中使用運算式，並讓管線等待核准工作。

1 在您的管線中，按一下**輸入**索引標籤。



2 保留**自動插入參數**為**無**。

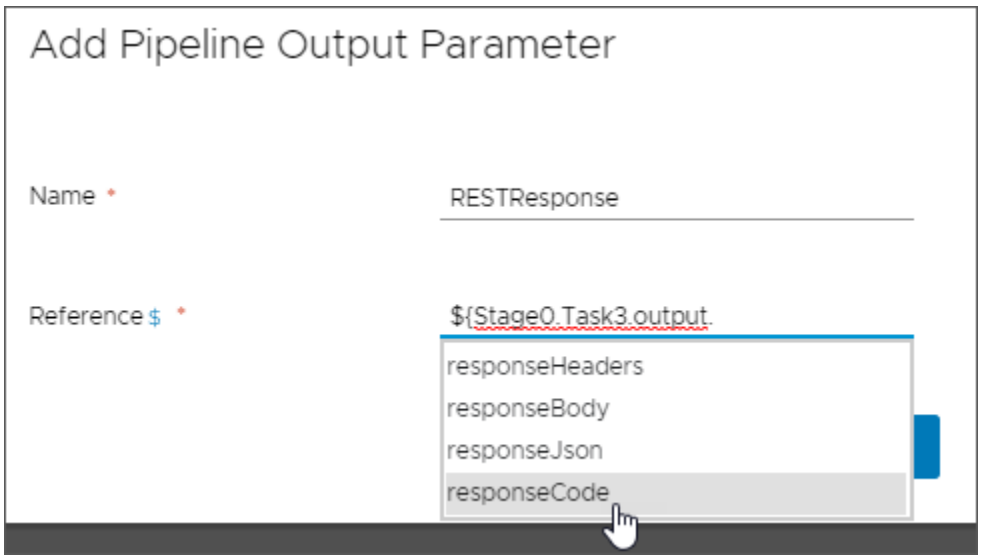
3 按一下**新增**，輸入參數名稱、值和說明，然後按一下**確定**。例如：

a 輸入 URL 名稱。

b 輸入值：`{Stage0.Task3.input.http://www.docs.vmware.com}`

c 輸入說明。

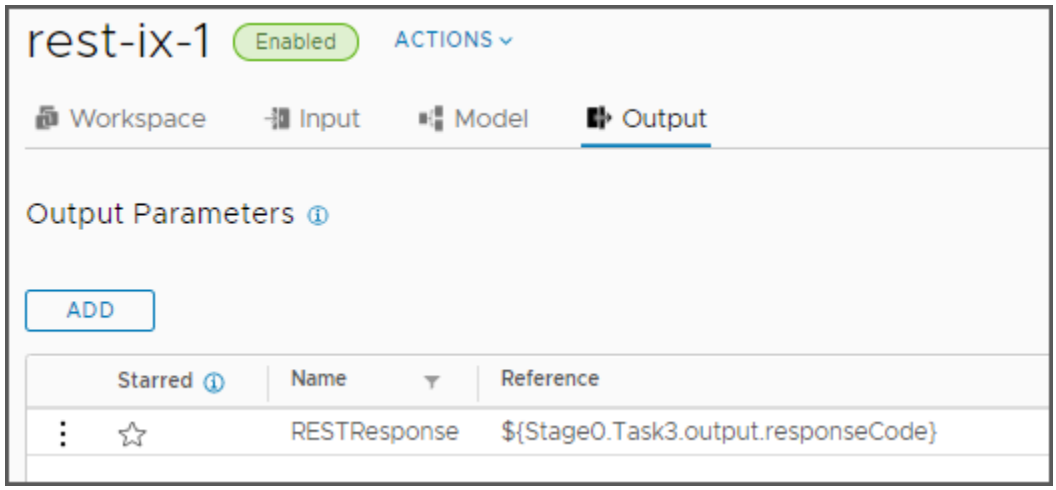
4 按一下**輸出**索引標籤，按一下**新增**，然後輸入輸出參數名稱和對應。



a 輸入唯一的輸出參數名稱。

b 按一下**參考**區域，然後輸入 $。

c 在彈出時選取選項，以輸入工作輸出對應。依序選取 **Stage0**、**Task3**、**輸出**和 **responseCode**。然後，按一下**確定**。

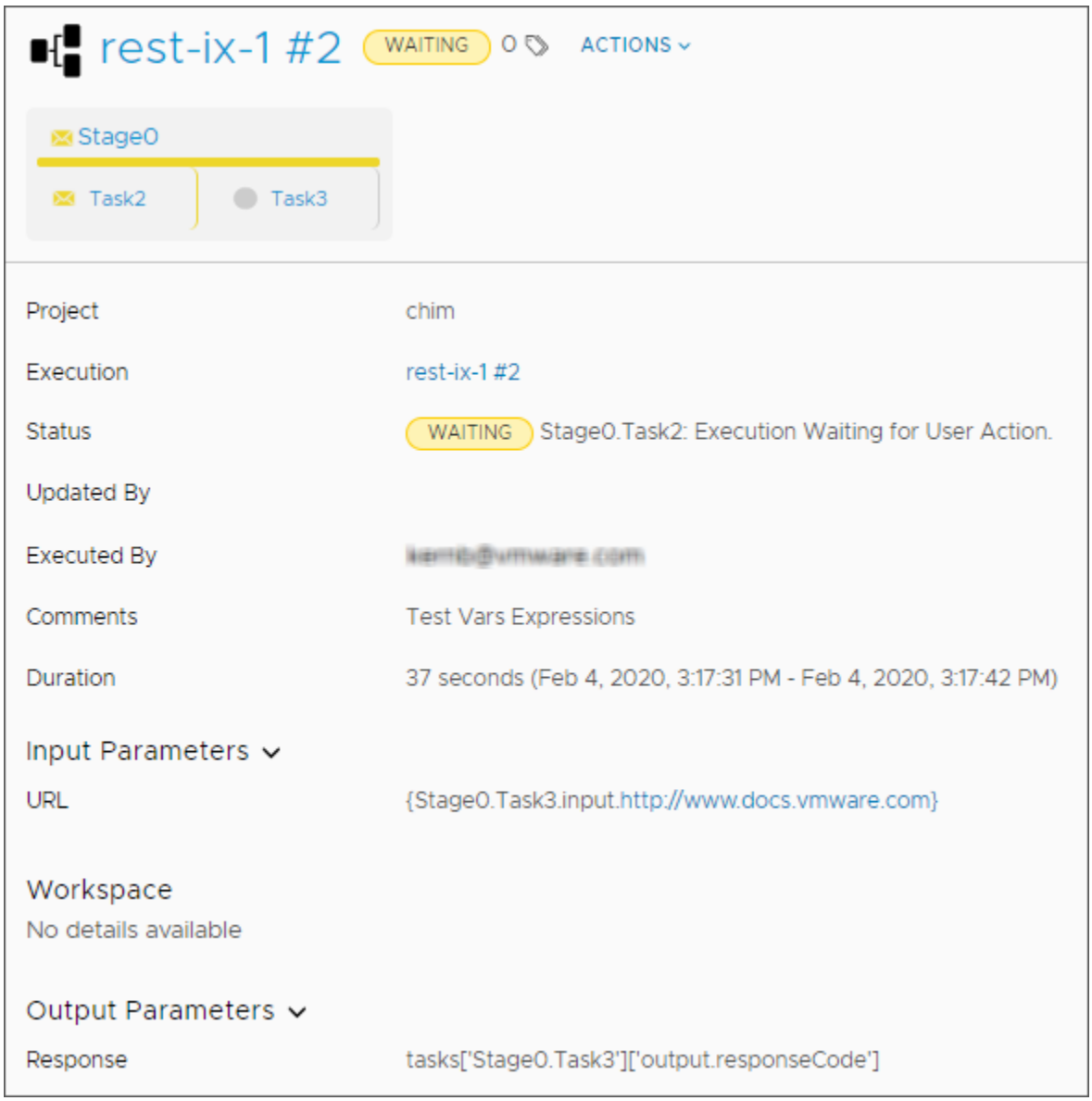


5 儲存管線。

6 從**動作**功能表中，按一下**執行**。

7 按一下**動作 > 檢視執行**。

8 按一下管線執行，然後檢查已定義的輸入參數和輸出參數。

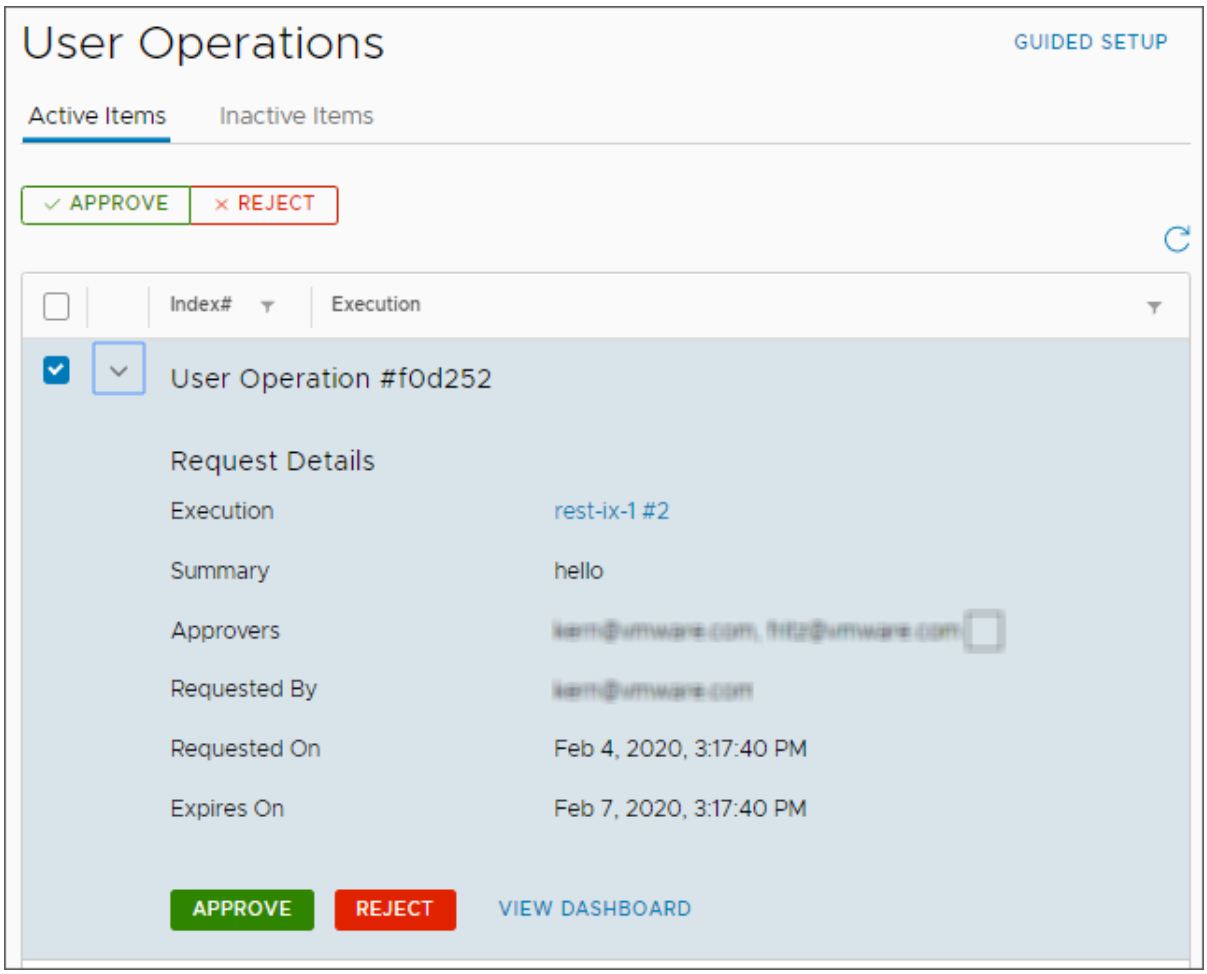


9 若要核准管線，請按一下**使用者作業**，然後在**作用中項目**索引標籤上檢視核准清單。或者，保留在 [執行] 中，按一下工作，然後按一下**核准**。

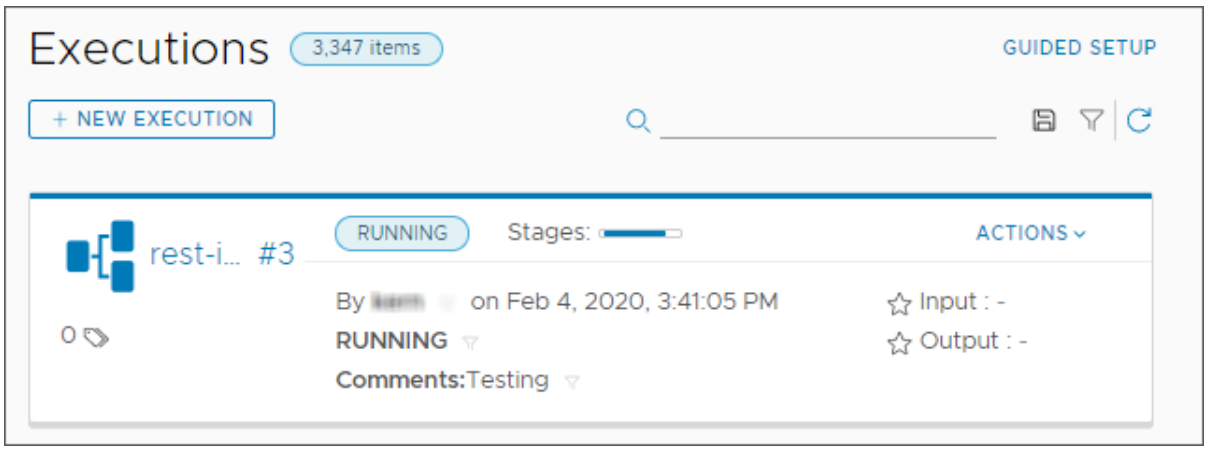
10 若要啟用**核准**和**拒絕**按鈕，請按一下執行旁邊的核取方塊。

11 若要查看詳細資料，請展開下拉式箭頭。

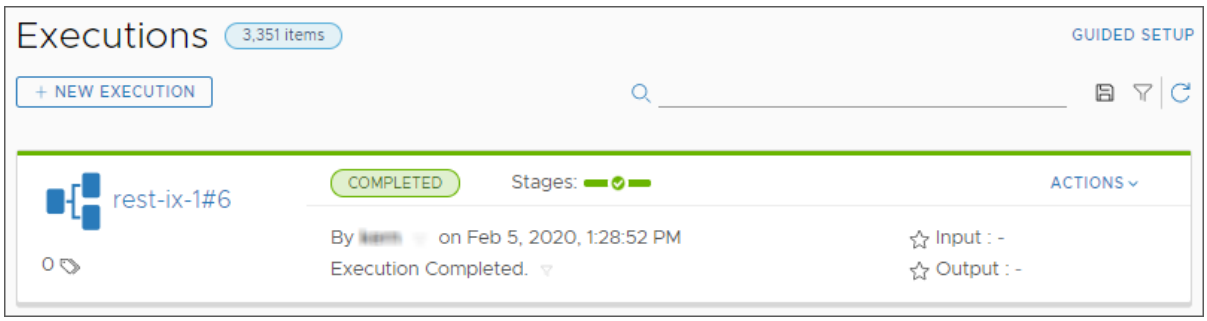
12 若要核准工作，請按一下**核准**，輸入原因，然後按一下**確定**。



13 按一下**執行**並監視管線繼續執行。



14 如果管線失敗，請更正任何錯誤，然後儲存管線並再次執行。



如何進一步瞭解變數和運算式

若要查看有關在繫結管線工作時使用變數和運算式的詳細資料，請參閱在 Code Stream 中繫結管線工作時可以使用哪些變數和運算式。

若要瞭解如何將管線工作輸出與條件變數繫結搭配使用，請參閱如何在 Code Stream 中使用 [條件] 工作中的變數繫結來執行或停止管線。

如何在 Code Stream 中使用 [條件] 工作中的變數繫結來執行或停止管線

您可以讓管線中的工作輸出根據所提供的條件判定執行或停止管線。若要根據工作輸出讓管線通過或失敗，請使用 [條件] 工作。

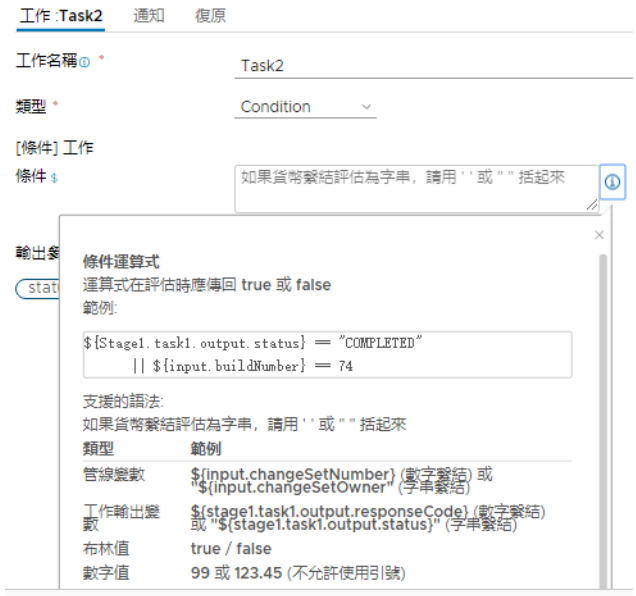
您可以使用**條件**工作作為管線中的決定點。透過將 [條件] 工作與提供的條件運算式搭配使用，您可以評估管線、階段和工作中的任何內容。

[條件] 工作的結果會決定管線中的下一個工作是否執行。

- 條件為 true，則允許管線執行繼續。
- 條件為 false，則停止管線。

如需有關如何透過將工作與 [條件] 工作繫結在一起，來將一個工作的輸出值用作下一個工作的輸入的範例，請參閱如何在 Code Stream 管線中使用變數繫結。

表 3-7. [條件] 工作及其條件運算式與管線的關聯性

[條件] 工作	影響對象	執行的作業
[條件] 工作	管線	**條件**工作根據工作輸出為 true 或 false 來判定管線在當時是執行或停止。
條件運算式	[條件] 工作輸出	管線執行時，您在**條件**工作中包含的條件運算式會產生 true 或 false 輸出狀態。例如，條件運算式可能需要 [條件] 工作輸出狀態為已完成，或使用組建編號 74。 條件運算式將顯示在 [條件] 工作中的 [工作] 索引標籤上。 

條件工作在功能和行為方面不同於其他工作類型中的**依條件**設定。

在其他工作類型中，**依條件**根據目前工作先決條件運算式的評估結果為 true 或 false，來判定目前工作是否執行，而不是連續執行工作。當管線執行時，**依條件**設定的條件運算式將針對目前工作產生 true 或 false 輸出狀態。**依條件**設定與其自己的條件運算式一起顯示在 [工作] 索引標籤上。

此範例使用 [條件] 工作。

必要條件

- 確認管線存在，且包含階段和工作。

程序

1 在您的管線中，確定 [條件] 工作必須出現的決策點。

2 在相依於其通過或失敗狀態的工作之前新增 [條件] 工作。

3 向 [條件] 工作新增條件運算式。

例如：

```
"${Stage1.task1.output.status}" == "COMPLETED" || ${input.buildNumber} == 74
```

4 驗證工作。

5 儲存管線，然後啟用並執行此管線。

結果

觀察管線執行，並注意管線是繼續執行，還是在 [條件] 工作處停止。

後續步驟

如果要復原管線部署，也可以使用 [條件] 工作。例如，在復原管線中，[條件] 工作可協助 Code Stream 根據條件運算式標記管線失敗，並可以針對各種失敗類型觸發單一復原流程。

若要復原部署，請參閱如何在 Code Stream 中復原我的部署。

在 Code Stream 中繫結管線工作時可以使用哪些變數和運算式

透過變數和運算式，您可以在管線工作中使用輸入參數與輸出參數。您輸入的參數會將管線工作繫結到一或多個變數、運算式或條件，並確定執行時的管線行為。

管線可以執行簡單或複雜的軟體交付解決方案

將管線工作繫結在一起時，可以包括預設運算式和複雜運算式。因此，您的管線可以執行簡單或複雜的軟體交付解決方案。

若要在管線中建立參數，請按一下**輸入**或**輸出**索引標籤，然後透過輸入貨幣符號 **$** 和運算式來新增變數。例如，此參數會用作呼叫 URL 的工作輸入：`${Stage0.Task3.input.URL}`。

變數繫結的格式使用稱為範圍和金鑰的語法元件。`SCOPE` 會將內容定義為輸入或輸出，而 `KEY` 會定義詳細資料。在參數範例 `${Stage0.Task3.input.URL}` 中，`input` 是 `SCOPE`，而 URL 是 `KEY`。

任何工作的輸出內容都可以解析為任何數目的變數繫結嵌套層級。

若要瞭解有關在管線中使用變數繫結的詳細資訊，請參閱如何在 Code Stream 管線中使用變數繫結。

將美元運算式與 Scope 和 Key 搭配使用，以繫結管線工作

您可以透過在貨幣符號變數中使用運算式，將管線工作繫結在一起。您可以輸入運算式為 `${SCOPE.KEY.<PATH>}`。

若要確定管線工作的行為，則在每個運算式中，`SCOPE` 是 Code Stream 所使用的內容。範圍將尋找 `KEY`，以定義工作所採取動作的詳細資料。當 `KEY` 的值為巢狀物件時，您可以提供選用的 `PATH`。

這些範例說明了 `SCOPE` 和 `KEY`，並向您顯示如何在管線中使用它們。

表 3-8. 使用 SCOPE 和 KEY

SCOPE	運算式的用途和範例	KEY	如何在管線中使用範圍和金鑰

input	管線的輸入內容：
	`${input.input1}`
		輸入內容的名稱	若要在工作中參考管線的輸入內容，請使用此格式：
			`tasks:`
			`  mytask:`
			`  type: REST`
			`  input:`
			`  url: ${input.url}`
			`  action: get`
			
			`input:`
			`  url: https://www.vmware.com`

output	管線的輸出內容：
	`${output.output1}`
		輸出內容的名稱	若要參考用於傳送通知的輸出內容，請使用此格式：
			`notifications:`
			`  email:`
			`  - endpoint: MyEmailEndpoint`
			`  subject: "Deployment Successful"`
			`  event: COMPLETED`
			`  to:`
			`  - user@example.org`
			`  body: |`
			`  Pipeline deployed the service successfully. Refer ${output.serviceURL}`

表 3-8. 使用 SCOPE 和 KEY (續)

SCOPE	運算式的用途和範例	KEY	如何在管線中使用範圍和金鑰

task input	工作的輸入：
	`${MY_STAGE.MY_TASK.input.SOMETHING}`
		在通知中指示工作輸入	當 Jenkins 工作啟動時，它可以參考從工作輸入觸發的工作的名稱。在此情況下，請使用下列格式傳送通知：
			`notifications:`
			`  email:`
			`  - endpoint: MyEmailEndpoint`
			`  stage: MY_STAGE`
			`  task: MY_TASK`
			`  subject: "Build Started"`
			`  event: STARTED`
			`  to:`
			`  - user@example.org`
			`  body: |`
			`  Jenkins job ${MY_STAGE.MY_TASK.input.job} started for commit id ${input.COMMITID}.`

task output	工作的輸出：
	`${MY_STAGE.MY_TASK.output.SOMETHING}`
		在後續工作中指示工作輸出	若要在工作 2 中參考管線工作 1 的輸出，請使用此格式：
			`taskOrder:`
			`  - task1`
			`  - task2`
			`tasks:`
			`  task1:`
			`  type: REST`
			`  input:`
			`  action: get`
			`  url: https://www.example.org/api/status`
			`  task2:`
			`  type: REST`
			`  input:`
			`  action: post`
			`  url: https://status.internal.example.org/api/activity`
			`  payload: ${MY_STAGE.task1.output.responseBody}`

表 3-8. 使用 SCOPE 和 KEY (續)

SCOPE	運算式的用途和範例	KEY	如何在管線中使用範圍和金鑰

var	變數：
	`${var.myVariable}`
		參考端點中的變數	若要針對密碼參考端點中的密碼變數，請使用此格式：
			`---`
			`project: MyProject`
			`kind: ENDPOINT`
			`name: MyJenkinsServer`
			`type: jenkins`
			`properties:`
			`url: https://jenkins.example.com`
			`username: jenkinsUser`
			`password: ${var.jenkinsPassword}`

var	變數：
	`${var.myVariable}`
		參考管線中的變數	若要參考管線 URL 中的變數，請使用此格式：
			`tasks:`
			`task1:`
			`type: REST`
			`input:`
			`action: get`
			`url: ${var.MY_SERVER_URL}`

task status	工作的狀態：
	`${MY_STAGE.MY_TASK.status}`
	`${MY_STAGE.MY_TASK.statusMessage}`		

stage status	階段的狀態：
	`${MY_STAGE.status}`
	`${MY_STAGE.statusMessage}`		

預設運算式

您可以在管線中搭配使用變數與運算式。此摘要包含您可以使用的預設運算式。

運算式	說明
`${comments}`	在管線執行請求時提供的註解。
`${duration}`	管線執行的持續時間。
`${endTime}`	管線執行的結束時間 (如果結束) (UTC)。

運算式	說明
${executedOn}	管線執行的開始時間 (UTC)，與開始時間相同。
${executionId}	管線執行的識別碼。
${executionUrl}	在使用者介面中導覽至管線執行的 URL。
${name}	管線的名稱。
${requestBy}	請求執行之使用者的名稱。
${stageName}	目前階段在階段範圍中使用時的名稱。
${startTime}	管線執行的開始時間 (UTC)。
${status}	執行狀態：
${statusMessage}	管線執行的狀態訊息。
${taskName}	目前工作在工作輸入或通知中使用時的名稱。

在管線工作中使用 SCOPE 和 KEY

您可以將運算式與任何支援的管線工作搭配使用。這些範例顯示如何定義 SCOPE 和 KEY，並確認語法。代碼範例使用 MY_STAGE 和 MY_TASK 作為管線階段和工作名稱。

若要瞭解有關可用工作的詳細資訊，請參閱在 Code Stream 中提供哪些類型的工作。

表 3-9. 管制工作

工作	Scope	Key	如何在工作中使用 SCOPE 和 KEY
使用者作業			
	Input	summary：使用者作業請求的摘要 description：使用者作業請求的說明 approvers：核准者電子郵件地址清單，其中每個項目可以是包含逗號的變數，也可以針對不同的電子郵件使用分號 approverGroups：平台和身分識別的核准者群組地址清單 sendemail：在設定為 true 時，選擇性地按照請求或回應傳送電子郵件通知 expirationInDays：表示請求到期時間的天數	${MY_STAGE.MY_TASK.input.summary} ${MY_STAGE.MY_TASK.input.description} ${MY_STAGE.MY_TASK.input.approvers} ${MY_STAGE.MY_TASK.input.approverGroups} ${MY_STAGE.MY_TASK.input.sendemail} ${MY_STAGE.MY_TASK.input.expirationInDays}

表 3-9. 管制工作 (續)

工作	Scope	Key	如何在工作中使用 SCOPE 和 KEY
	Output	index：表示請求的 6 位數十六進位字串 respondedBy：核准/拒絕使用者作業之人員的帳戶名稱 respondedByEmail：回應者的電子郵件地址 comments：回應期間提供的註解	${MY_STAGE.MY_TASK.output.index} ${MY_STAGE.MY_TASK.output.respondedBy} ${MY_STAGE.MY_TASK.output.respondedByEmail} ${MY_STAGE.MY_TASK.output.comments}
條件			
	Input	condition：要評估的條件。當條件評估為 true 時，它會將工作標記為完成，而其他回應會使工作失敗	${MY_STAGE.MY_TASK.input.condition}
	Output	result：評估後的結果	${MY_STAGE.MY_TASK.output.response}

表 3-10. 管線工作

工作	Scope	Key	如何在工作中使用 SCOPE 和 KEY
管線			
	Input	name：要執行的管線的名稱 inputProperties：要傳遞到巢狀執行的輸入內容	${MY_STAGE.MY_TASK.input.name} ${MY_STAGE.MY_TASK.input.inputProperties} # 參考所有內容 ${MY_STAGE.MY_TASK.input.inputProperties.input1} # 參考 input1 的值
	Output	executionStatus：管線執行的狀態 executionIndex：管線執行的索引 outputProperties：管線執行的輸出內容	${MY_STAGE.MY_TASK.output.executionStatus} ${MY_STAGE.MY_TASK.output.executionIndex} ${MY_STAGE.MY_TASK.output.outputProperties} # 參考所有內容 ${MY_STAGE.MY_TASK.output.outputProperties.output1} # 參考 output1 的值

表 3-11. 自動化連續整合工作

工作	Scope	Key	如何在工作中使用 SCOPE 和 KEY
CI			
	Input	steps：一組字串，表示要執行的命令 export：執行步驟後要保留的環境變數 artifacts：要在共用路徑中保留的構件路徑 process：用於 JUnit、JaCoCo、Checkstyle、FindBugs 處理的組態元素集	${MY_STAGE.MY_TASK.input.steps} ${MY_STAGE.MY_TASK.input.export} ${MY_STAGE.MY_TASK.input.artifacts} ${MY_STAGE.MY_TASK.input.process} ${MY_STAGE.MY_TASK.input.process[0].path} # 參考第一個組態的路徑

表 3-11. 自動化連續整合工作 (續)

工作	Scope	Key	如何在工作中使用 SCOPE 和 KEY

	Output	`exports`：索引鍵-值配對，表示從輸入 `export` 中匯出的環境變數
		`artifacts`：已成功保留的構件的路徑
		`processResponse`：輸入 `process` 的已處理結果集
			`${MY_STAGE.MY_TASK.output.exports}` # 參考所有匯出
			`${MY_STAGE.MY_TASK.output.exports.myvar}` # 參考 **`myvar`** 的值
			`${MY_STAGE.MY_TASK.output.artifacts}`
			`${MY_STAGE.MY_TASK.output.processResponse}`
			`${MY_STAGE.MY_TASK.output.processResponse[0].result}` # 第一個程序組態的結果

自訂			

	Input	`name`：自訂整合的名稱
		`version`：自訂整合的版本 (已發佈或已過時)
		`properties`：要傳送至自訂整合的內容
			`${MY_STAGE.MY_TASK.input.name}`
			`${MY_STAGE.MY_TASK.input.version}`
			`${MY_STAGE.MY_TASK.input.properties}` # 參考所有內容
			`${MY_STAGE.MY_TASK.input.properties.property1}` # 參考 property1 的值

	Output	`properties`：來自自訂整合回應的輸出內容	`${MY_STAGE.MY_TASK.output.properties}` # 參考所有內容 `${MY_STAGE.MY_TASK.output.properties.property1}` # 參考 property1 的值

表 3-12. 自動化連續部署工作：雲端範本

工作	Scope	Key	如何在工作中使用 SCOPE 和 KEY
雲端範本			
	Input	`action`：**createDeployment**、**updateDeployment**、**deleteDeployment**、**rollbackDeployment** 中的其中一個 `blueprintInputParams`：用於**建立部署**和**更新部署**動作 `allowDestroy`：可以在更新部署程序中銷毀機器。 **CREATE_DEPLOYMENT** ■ `blueprintName`：雲端範本的名稱 ■ `blueprintVersion`：雲端範本的版本 或 ■ `fileUrl`：選取 GIT 伺服器後，遠端雲端範本 YAML 的 URL。 **UPDATE_DEPLOYMENT** 這些組合中的任何一種： ■ `blueprintName`：雲端範本的名稱 ■ `blueprintVersion`：雲端範本的版本 或 ■ `fileUrl`：選取 GIT 伺服器後，遠端雲端範本 YAML 的 URL。 `------` ■ `deploymentId`：部署的識別碼 或 ■ `deploymentName`：部署的名稱 `------` **DELETE_DEPLOYMENT** ■ `deploymentId`：部署的識別碼 或 ■ `deploymentName`：部署的名稱 **ROLLBACK_DEPLOYMENT** 這些組合中的任何一種： ■ `deploymentId`：部署的識別碼	

表 3-12. 自動化連續部署工作：雲端範本 (續)

工作	Scope	Key	如何在工作中使用 SCOPE 和 KEY
		或 ■ deploymentName：部署的名稱 ------ ■ blueprintName：雲端範本的名稱 ■ rollbackVersion：要復原到的版本	
	Output		可繫結至其他工作或管線輸出的參數： ■ 部署名稱可以 **${Stage0.Task0.output.deploymentName}** 形式進行存取 ■ 部署識別碼可以 **${Stage0.Task0.output.deploymentId}** 形式進行存取 ■ 部署詳細資料是一個複雜物件，可透過使用 JSON 結果存取內部詳細資料。 若要存取任何內容，請使用點運算子遵循 JSON 階層。例如，若要存取資源 Cloud_Machine_1[0] 的位址，**$** 繫結為： **${Stage0.Task0.output.deploymentDetails.resources['Cloud_Machine_1[0]'].address}** 同樣地，對於類型模板，**$** 繫結為： **${Stage0.Task0.output.deploymentDetails.resources['Cloud_Machine_1[0]'].flavor}** 在 Code Stream 使用者介面中，您可以取得任何內容的 **$** 繫結。 1 在工作輸出內容區域中，按一下**檢視輸出 JSON**。 2 若要尋找 **$** 繫結，請輸入任何內容。 3 按一下搜尋圖示，其會顯示對應的 **$** 繫結。

JSON 輸出範例：

範例部署詳細資料物件：


```
{
    "id": "6a031f92-d0fa-42c8-bc9e-3b260ee2f65b",
    "name": "deployment_6a031f92-d0fa-42c8-bc9e-3b260ee2f65b",
    "description": "Pipeline Service triggered operation",
    "orgId": "434f6917-4e34-4537-b6c0-3bf3638a71bc",
    "blueprintId": "8d1dd801-3a32-4f3b-adde-27f8163dfe6f",
    "blueprintVersion": "1",
    "createdAt": "2020-08-27T13:50:24.546215Z",
    "createdBy": "user@vmware.com",
    "lastUpdatedAt": "2020-08-27T13:52:50.674957Z",
    "lastUpdatedBy": "user@vmware.com",
    "inputs": {},
    "simulated": false,
    "projectId": "267f8448-d26f-4b65-b310-9212adb3c455",
    "resources": {
        "Cloud_Machine_1[0]": {
            "id": "/resources/compute/1606fbcd-40e5-4edc-ab85-7b559aa986ad",
            "name": "Cloud_Machine_1[0]",
            "powerState": "ON",
            "address": "10.108.79.33",
            "resourceLink": "/resources/compute/1606fbcd-40e5-4edc-ab85-7b559aa986ad",
            "componentTypeId": "Cloud.vSphere.Machine",
            "endpointType": "vsphere",
            "resourceName": "Cloud_Machine_1-mcm110615-146929827053",
            "resourceId": "1606fbcd-40e5-4edc-ab85-7b559aa986ad",
            "resourceDescLink": "/resources/compute-descriptions/1952d1d3-15f0-4574-
ae42-4fbf8a87d4cc",
            "zone": "Automation / Vms",
            "countIndex": "0",
            "image": "ubuntu",
            "count": "1",
            "flavor": "small",
            "region": "MYBU",
            "_clusterAllocationSize": "1",
            "osType": "LINUX",
            "componentType": "Cloud.vSphere.Machine",
            "account": "bha"
        }
```

```
    },
    "status": "CREATE_SUCCESSFUL",
    "deploymentURI": "https://api.yourenv.com/automation-ui/#/deployment-ui;ash=/deployment/
6a031f92-d0fa-42c8-bc9e-3b260ee2f65b"
}
```

表 3-13. 自動化連續部署工作：Kubernetes

工作	Scope	Key	如何在工作中使用 SCOPE 和 KEY
Kubernetes			

	Input	`action`：**GET**、**CREATE**、**APPLY**、**DELETE**、**ROLLBACK** 中的其中一個
		■ `timeout`：任何動作的整體逾時
		■ `filterByLabel`：要使用 K8S labelSelector 針對動作 **GET** 篩選的其他標籤
		GET、**CREATE**、**DELETE**、**APPLY**
		■ `yaml`：要處理並傳送至 Kubernetes 的內嵌 YAML
		■ `parameters`：KEY、VALUE 配對 - 在內嵌 YAML 輸入區域中，以 **VALUE** 取代 **$$KEY**
		■ `filePath`：SCM Git 端點 (如果提供) 中要從中擷取 YAML 的相對路徑
		■ `scmConstants`：KEY、VALUE 配對 - 在透過 SCM 擷取的 YAML 中，以 **VALUE** 取代 **$${KEY}**
		■ `continueOnConflict`：設定為 true 時，如果資源已存在，則工作會繼續。
		ROLLBACK
		■ `resourceType`：要復原的資源類型
		■ `resourceName`：要復原的資源名稱
		■ `namespace`：必須執行復原的命名空間
		■ `revision`：要復原到的修訂版本
			`${MY_STAGE.MY_TASK.input.action}` # 決定要執行的動作。
			`${MY_STAGE.MY_TASK.input.timeout}`
			`${MY_STAGE.MY_TASK.input.filterByLabel}`
			`${MY_STAGE.MY_TASK.input.yaml}`
			`${MY_STAGE.MY_TASK.input.parameters}`
			`${MY_STAGE.MY_TASK.input.filePath}`
			`${MY_STAGE.MY_TASK.input.scmConstants}`
			`${MY_STAGE.MY_TASK.input.continueOnConflict}`
			`${MY_STAGE.MY_TASK.input.resourceType}`
			`${MY_STAGE.MY_TASK.input.resourceName}`
			`${MY_STAGE.MY_TASK.input.namespace}`
			`${MY_STAGE.MY_TASK.input.revision}`

	Output	`response`：擷取完整回應 `response.<RESOURCE>`：資源對應於 configMaps、部署、端點、入口、工作、命名空間、網繭、replicaSets、Replicationcontroller、密碼、服務、statefulSets、節點、負載平衡器。 `response.<RESOURCE>.<KEY>`：金鑰對應於 apiVersion、種類、中繼資料、規格中的其中一個	`${MY_STAGE.MY_TASK.output.response}` `${MY_STAGE.MY_TASK.output.response.}`

表 3-14. 整合開發、測試和部署應用程式

工作	Scope	Key	如何在工作中使用 SCOPE 和 KEY
Bamboo			

	Input	`plan`：計劃名稱
		`planKey`：計劃金鑰
		`variables`：要傳遞至計劃的變數
		`parameters`：要傳遞至計劃的參數
			`${MY_STAGE.MY_TASK.input.plan}`
			`${MY_STAGE.MY_TASK.input.planKey}`
			`${MY_STAGE.MY_TASK.input.variables}`
			`${MY_STAGE.MY_TASK.input.parameters}` # 參考所有參數
			`${MY_STAGE.MY_TASK.input.parameters.param1}` # 參考 param1 的值

	Output	`resultUrl`：所產生建置的 URL
		`buildResultKey`：所產生建置的金鑰
		`buildNumber`：組建編號
		`buildTestSummary`：所執行測試的摘要
		`successfulTestCount`：測試結果已通過
		`failedTestCount`：測試結果失敗
		`skippedTestCount`：測試結果已略過
		`artifacts`：來自建置的構件
			`${MY_STAGE.MY_TASK.output.resultUrl}`
			`${MY_STAGE.MY_TASK.output.buildResultKey}`
			`${MY_STAGE.MY_TASK.output.buildNumber}`
			`${MY_STAGE.MY_TASK.output.buildTestSummary}` # 參考所有結果
			`${MY_STAGE.MY_TASK.output.successfulTestCount}` # 參考特定測試計數
			`${MY_STAGE.MY_TASK.output.buildNumber}`

Jenkins			
	Input	`job`：Jenkins 工作的名稱 `parameters`：要傳遞至工作的參數	`${MY_STAGE.MY_TASK.input.job}` `${MY_STAGE.MY_TASK.input.parameters}` # 參考所有參數 `${MY_STAGE.MY_TASK.input.parameters.param1}` # 參考參數的值

	Output	`job`：Jenkins 工作的名稱
		`jobId`：所產生工作的識別碼，例如 1234
		`jobStatus`：Jenkins 中的狀態
		`jobResults`：測試/程式碼涵蓋範圍結果的集合
		`jobUrl`：所產生工作執行的 URL
			`${MY_STAGE.MY_TASK.output.job}`
			`${MY_STAGE.MY_TASK.output.jobId}`
			`${MY_STAGE.MY_TASK.output.jobStatus}`
			`${MY_STAGE.MY_TASK.output.jobResults}` # 參考所有結果
			`${MY_STAGE.MY_TASK.output.jobResults.junitResponse}` # 參考 JUnit 結果
			`${MY_STAGE.MY_TASK.output.jobResults.jacocoRespose}` # 參考 JaCoCo 結果
			`${MY_STAGE.MY_TASK.output.jobUrl}`

TFS			

	Input	`projectCollection`：TFS 中的專案集合
		`teamProject`：從可用集合中選取的專案
		`buildDefinitionId`：要執行的建置定義識別碼
			`${MY_STAGE.MY_TASK.input.projectCollection}`
			`${MY_STAGE.MY_TASK.input.teamProject}`
			`${MY_STAGE.MY_TASK.input.buildDefinitionId}`

表 3-14. 整合開發、測試和部署應用程式 (續)

工作	Scope	Key	如何在工作中使用 SCOPE 和 KEY

	Output	`buildId`：所產生建置的識別碼
		`buildUrl`：用於造訪建置摘要的 URL
		`logUrl`：用於造訪記錄的 URL
		`dropLocation`：構件的放置位置 (如果有)
			`${MY_STAGE.MY_TASK.output.buildId}`
			`${MY_STAGE.MY_TASK.output.buildUrl}`
			`${MY_STAGE.MY_TASK.output.logUrl}`
			`${MY_STAGE.MY_TASK.output.dropLocation}`

vRO			
	Input	`workflowId`：要執行的工作流程的識別碼 `parameters`：要傳遞至工作流程的參數	`${MY_STAGE.MY_TASK.input.workflowId}` `${MY_STAGE.MY_TASK.input.parameters}`
	Output	`workflowExecutionId`：工作流程執行的識別碼 `properties`：來自工作流程執行的輸出內容	`${MY_STAGE.MY_TASK.output.workflowExecutionId}` `${MY_STAGE.MY_TASK.output.properties}`

表 3-15. 透過 API 整合其他應用程式

工作	Scope	Key	如何在工作中使用 SCOPE 和 KEY
REST			

	Input	`url`：要呼叫的 URL
		`action`：要使用的 HTTP 方法
		`headers`：要傳遞的 HTTP 標頭
		`payload`：請求裝載
		`fingerprint`：URL 為 https 時要符合的指紋
		`allowAllCerts`：設定為 true 時，可以是 URL 為 https 的任何憑證
			`${MY_STAGE.MY_TASK.input.url}`
			`${MY_STAGE.MY_TASK.input.action}`
			`${MY_STAGE.MY_TASK.input.headers}`
			`${MY_STAGE.MY_TASK.input.payload}`
			`${MY_STAGE.MY_TASK.input.fingerprint}`
			`${MY_STAGE.MY_TASK.input.allowAllCerts}`

	Output	`responseCode`：HTTP 回應代碼
		`responseHeaders`：HTTP 回應標頭
		`responseBody`：收到的回應字串格式
		`responseJson`：周遊回應 (如果內容類型為 application/json)
			`${MY_STAGE.MY_TASK.output.responseCode}`
			`${MY_STAGE.MY_TASK.output.responseHeaders}`
			`${MY_STAGE.MY_TASK.output.responseHeaders.header1}` # 參考回應標頭「header1」
			`${MY_STAGE.MY_TASK.output.responseBody}`
			`${MY_STAGE.MY_TASK.output.responseJson}` # 參考 JSON 回應
			`${MY_STAGE.MY_TASK.output.responseJson.a.b.c}` # 參考回應中 a.b.c JSON 路徑之後的巢狀物件

輪詢			

表 3-15. 透過 API 整合其他應用程式 (續)

工作	Scope	Key	如何在工作中使用 SCOPE 和 KEY

	Input	`url`：要呼叫的 URL
		`headers`：要傳遞的 HTTP 標頭
		`exitCriteria`：工作成功或失敗需符合的準則。索引鍵-值配對：「成功」→運算式，「失敗」→運算式
		`pollCount`：要執行的反覆運算次數。Code Stream 管理員最多可以將輪詢計數設定為 10000。
		`pollIntervalSeconds`：每個反覆運算之間要等待的秒數。輪詢時間間隔必須大於或等於 60 秒。
		`ignoreFailure`：設定為 true 時，忽略中繼回應失敗
		`fingerprint`：URL 為 https 時要符合的指紋
		`allowAllCerts`：設定為 true 時，可以是 URL 為 https 的任何憑證
			`${MY_STAGE.MY_TASK.input.url}`
			`${MY_STAGE.MY_TASK.input.headers}`
			`${MY_STAGE.MY_TASK.input.exitCriteria}`
			`${MY_STAGE.MY_TASK.input.pollCount}`
			`${MY_STAGE.MY_TASK.input.pollIntervalSeconds}`
			`${MY_STAGE.MY_TASK.input.ignoreFailure}`
			`${MY_STAGE.MY_TASK.input.fingerprint}`
			`${MY_STAGE.MY_TASK.input.allowAllCerts}`

	Output	`responseCode`：HTTP 回應代碼
		`responseBody`：收到的回應字串格式
		`responseJson`：周遊回應 (如果內容類型為 **application/json**)
			`${MY_STAGE.MY_TASK.output.responseCode}`
			`${MY_STAGE.MY_TASK.output.responseBody}`
			`${MY_STAGE.MY_TASK.output.responseJson} # Refer to response as JSON`

表 3-16. 執行遠端指令碼和使用者定義的指令碼

工作	Scope	Key	如何在工作中使用 SCOPE 和 KEY
PowerShell 若要執行 PowerShell 工作，您必須： ■ 具有與遠端 Windows 主機的作用中工作階段。 ■ 如果打算輸入 base64 PowerShell 命令，請先計算整體命令長度。如需詳細資料，請參閱在 Code Stream 中提供哪些類型的工作。			

	Input	`host`：機器的 IP 位址或主機名稱
		`username`：用於連線的使用者名稱
		`password`：用於連線的密碼
		`useTLS`：嘗試 https 連線
		`trustCert`：設定為 true 時，信任自我簽署憑證
		`script`：要執行的指令碼
		`workingDirectory`：執行指令碼前要切換到的目錄路徑
		`environmentVariables`：要設定的環境變數的索引鍵-值配對
		`arguments`：要傳遞至指令碼的引數
			`${MY_STAGE.MY_TASK.input.host}`
			`${MY_STAGE.MY_TASK.input.username}`
			`${MY_STAGE.MY_TASK.input.password}`
			`${MY_STAGE.MY_TASK.input.useTLS}`
			`${MY_STAGE.MY_TASK.input.trustCert}`
			`${MY_STAGE.MY_TASK.input.script}`
			`${MY_STAGE.MY_TASK.input.workingDirectory}`
			`${MY_STAGE.MY_TASK.input.environmentVariables}`
			`${MY_STAGE.MY_TASK.input.arguments}`

表 3-16. 執行遠端指令碼和使用者定義的指令碼 (續)

工作	Scope	Key	如何在工作中使用 SCOPE 和 KEY
	Output	response：檔案 $SCRIPT_RESPONSE_FILE 的內容 responseFilePath：$SCRIPT_RESPONSE_FILE 的值 exitCode：程序結束代碼 logFilePath：包含 stdout 之檔案的路徑 errorFilePath：包含 stderr 之檔案的路徑	${MY_STAGE.MY_TASK.output.response} ${MY_STAGE.MY_TASK.output.responseFilePath} ${MY_STAGE.MY_TASK.output.exitCode} ${MY_STAGE.MY_TASK.output.logFilePath} ${MY_STAGE.MY_TASK.output.errorFilePath}
SSH			
	Input	host：機器的 IP 位址或主機名稱 username：用於連線的使用者名稱 password：用於連線的密碼 (可選擇性地使用 privateKey) privateKey：用於連線的 PrivateKey passphrase：用於解鎖 privateKey 的選用複雜密碼 script：要執行的指令碼 workingDirectory：執行指令碼前要切換到的目錄路徑 environmentVariables：要設定的環境變數的索引鍵-值配對	${MY_STAGE.MY_TASK.input.host} ${MY_STAGE.MY_TASK.input.username} ${MY_STAGE.MY_TASK.input.password} ${MY_STAGE.MY_TASK.input.privateKey} ${MY_STAGE.MY_TASK.input.passphrase} ${MY_STAGE.MY_TASK.input.script} ${MY_STAGE.MY_TASK.input.workingDirectory} ${MY_STAGE.MY_TASK.input.environmentVariables}
	Output	response：檔案 $SCRIPT_RESPONSE_FILE 的內容 responseFilePath：$SCRIPT_RESPONSE_FILE 的值 exitCode：程序結束代碼 logFilePath：包含 stdout 之檔案的路徑 errorFilePath：包含 stderr 之檔案的路徑	${MY_STAGE.MY_TASK.output.response} ${MY_STAGE.MY_TASK.output.responseFilePath} ${MY_STAGE.MY_TASK.output.exitCode} ${MY_STAGE.MY_TASK.output.logFilePath} ${MY_STAGE.MY_TASK.output.errorFilePath}

如何在工作之間使用變數繫結

此範例會顯示如何在管線工作中使用變數繫結。

表 3-17. 範例語法格式

範例	語法
為管線通知和管線輸出內容使用工作輸出值	`${<Stage Key>.<Task Key>.output.<Task output key>}`
參考上一個工作的輸出值做為目前工作的輸入	`${<Previous/Current Stage key>.<Previous task key not in current Task group>.output.<task output key>}`

進一步瞭解

若要進一步瞭解工作中的繫結變數，請參閱：

- 如何在 Code Stream 管線中使用變數繫結
- 如何在 Code Stream 中使用 [條件] 工作中的變數繫結來執行或停止管線
- 在 Code Stream 中提供哪些類型的工作

如何在 Code Stream 中傳送有關我的管線的通知

通知是與您的團隊進行通訊，並讓他們知道管線在 Code Stream 中的的狀態的方式。

若要在管線執行時傳送通知，您可以根據整個管線、階段或工作的狀態設定 Code Stream 通知。

- 電子郵件通知會在以下情況下傳送電子郵件：
  - 管線完成、正在等待、失敗、取消或啟動。
  - 階段完成、失敗或啟動。
  - 工作完成、正在等待、失敗或啟動。
- 票證通知會建立票證，並在以下情況下將其指派給團隊成員：
  - 管線失敗或完成。
  - 階段失敗。
  - 工作失敗。
- Webhook 通知會在以下情況下將請求傳送至其他應用程式：
  - 管線失敗、完成、正在等待、取消或啟動。
  - 階段失敗、完成或啟動。
  - 工作失敗、完成、正在等待或啟動。

例如，您可以在使用者作業工作中設定電子郵件通知，以在管線中的特定點取得核准。當管線執行時，此工作會將電子郵件傳送給必須核准工作的人員。如果使用者作業工作設定了到期逾時 (以天、小時或分鐘為單位)，則所需使用者必須在工作到期前核准管線。否則，管線會按預期失敗。

若要在管線工作失敗時建立 Jira 票證，您可以設定通知。或者，若要根據管線事件向 Slack 通道傳送有關管線狀態的請求，您可以設定 Webhook 通知。

您可以在所有類型的通知中使用變數。例如，您可以在 Webhook 通知的 URL 中使用 **`${var}`**。

必要條件

- 確認已建立一或多個管線。請參閱第 5 章 使用 Code Stream 的教學課程 中的使用案例。
- 若要傳送電子郵件通知，請確認您可以存取有效的電子郵件伺服器。如需說明，請咨詢您的管理員。
- 若要建立票證，例如 Jira 票證，請確認存在端點。請參閱什麼是 Code Stream 中的端點 。
- 若要根據整合傳送通知，請建立 webhook 通知。然後，確認 webhook 已新增並正常運作。您可以將通知用於諸如 Slack、GitHub 或 GitLab 之類的應用程式。

程序

1 開啟管線。

2 針對整體管線狀態或某個階段或工作的狀態建立通知：

針對以下狀態建立通知：	您執行的操作：
管線狀態	按一下管線畫布上的空白區域。
階段狀態	按一下管線階段中的空白區域。
工作狀態	按一下管線階段中的工作。

3 按一下**通知**索引標籤。

4 按一下**新增**，選取通知的類型，並設定通知詳細資料。

5 若要在管線成功時建立 Slack 通知，請建立 webhook 通知。

   a 選取 **Webhook**。

   b 若要設定 Slack 通知，請輸入資訊。

   c 按一下**儲存**。

   d 當管線執行時，Slack 通道會收到管線狀態的通知。例如，使用者可能會在 Slack 通道上看到下列內容：

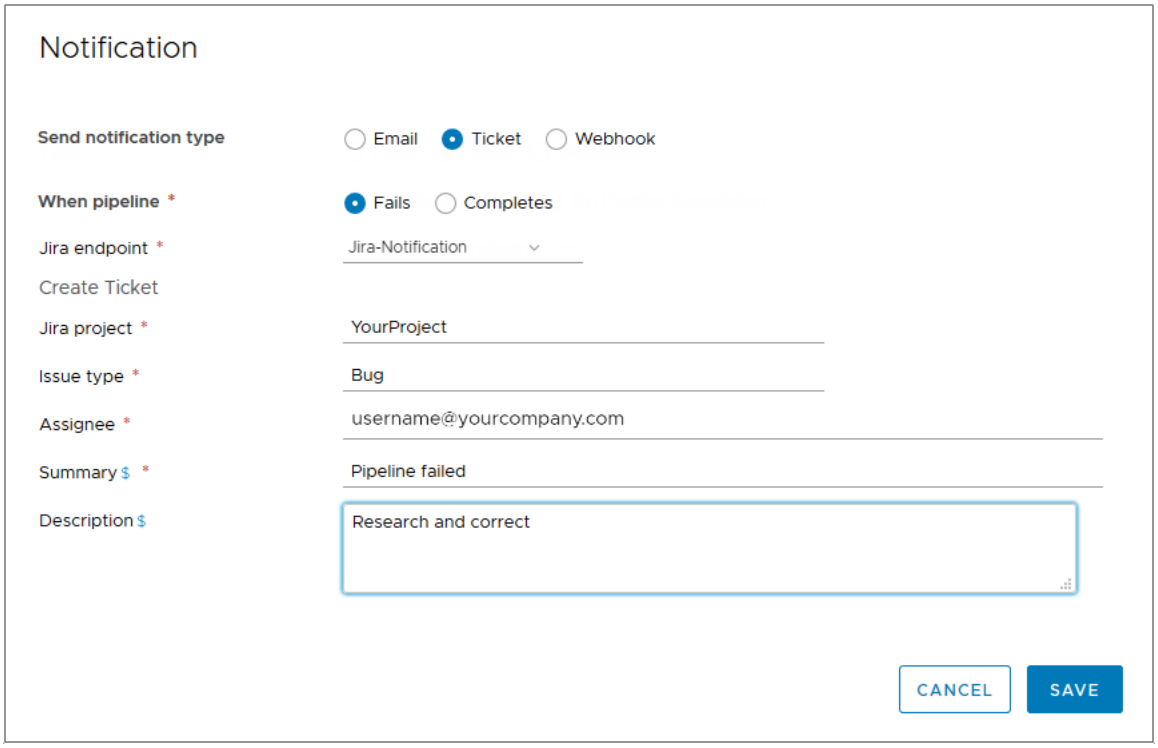
```
Codestream APP [12:01 AM]
Tested by User1 - Staging Pipeline 'User1-Pipeline', Pipeline ID
'e9b5884d809ce2755728177f70f8a' succeeded
```

6 若要建立 Jira 票證，請設定票證資訊。

a 選取**票證**。

b 若要設定 Jira 通知，請輸入資訊。

c 按一下**儲存**。



結果

恭喜您！您已瞭解可以在 Code Stream 的管線的數個區域中建立不同類型的通知。

後續步驟

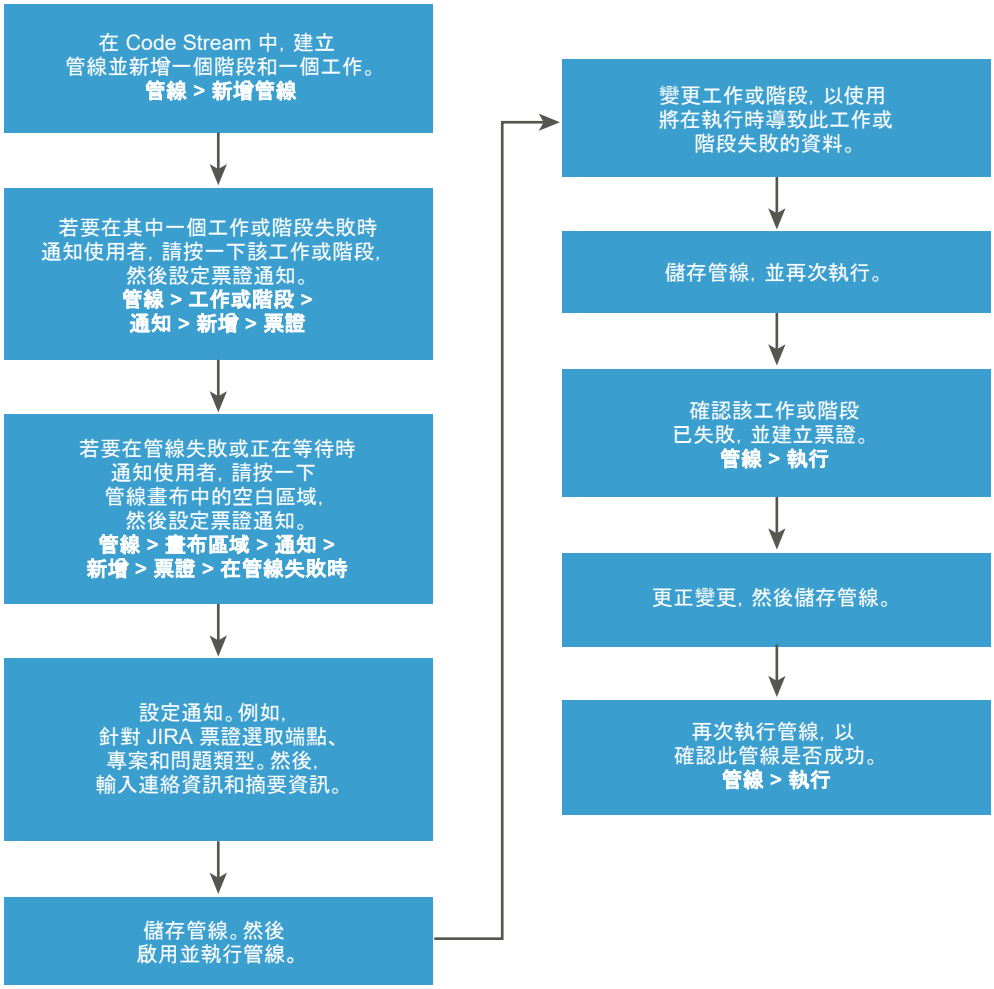
如需如何建立通知的詳細範例，請參閱如何在管線工作失敗時在 Code Stream 中建立 Jira 票證。

如何在管線工作失敗時在 Code Stream 中建立 Jira 票證

如果管線中的階段或工作失敗，您可以讓 Code Stream 建立 Jira 票證。您可以將票證指派給必須解決問題的人員。您也可以在管線等待時或管線成功後建立票證。

您可以在工作、階段或管線上新增和設定通知。 Code Stream 會根據您在其中新增通知的工作、階段或管線的狀態來建立票證。例如，如果某個端點不可用，可使用 Code Stream 為無法連線到端點而失敗的工作建立 Jira 票證。

當管線成功時，您也可以建立通知。例如，您可以通知 QA 團隊管線成功，以便他們可以確認建置並執行其他測試管線。或者，您可以通知效能團隊，以便他們能夠衡量管線的效能，並為暫存或生產的更新做好準備。



此範例會在管線工作失敗時建立 Jira 票證。

必要條件

- 確認您具有有效的 Jira 帳戶，並且可以登入 Jira 執行個體。
- 確認 Jira 端點已存在，並且正在運作。

程序

1 在您的管線中，按一下工作。

2 在 [工作組態] 區域中，按一下**通知**。

3 按一下**新增**，然後設定票證資訊。

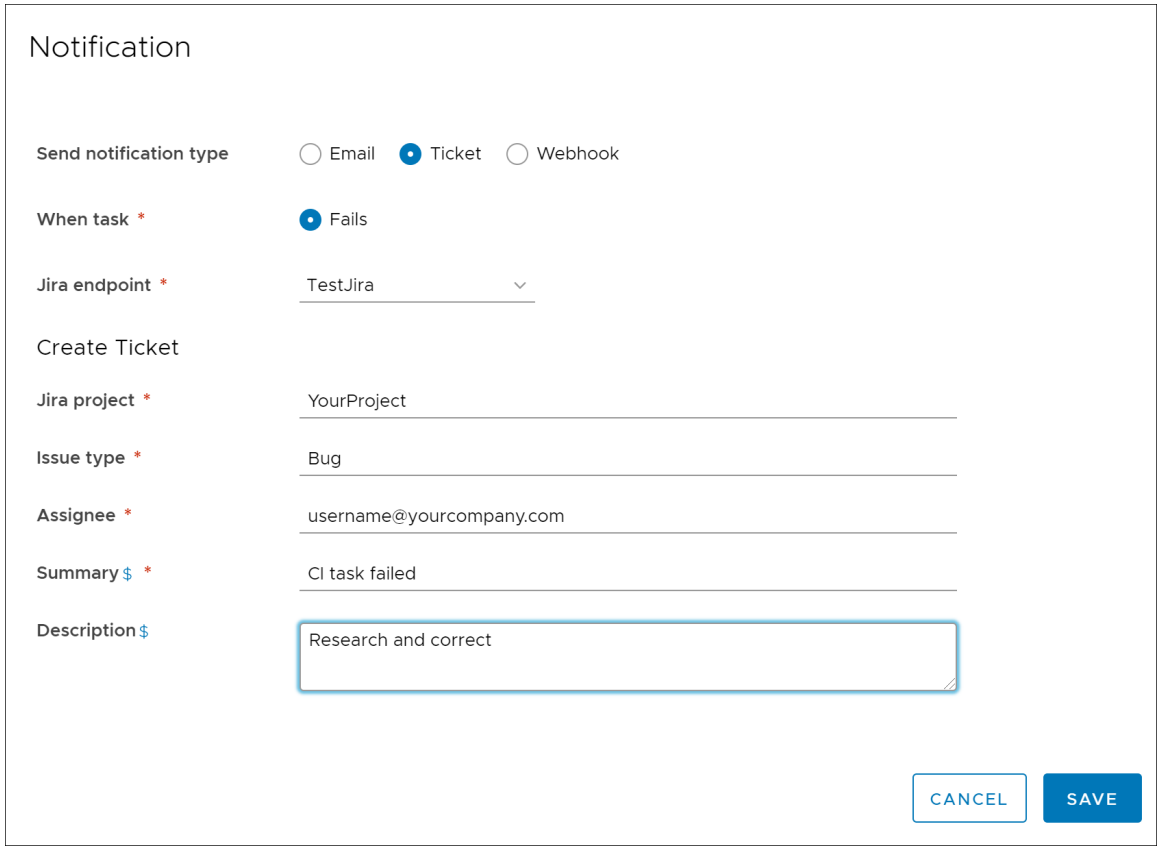
   a 按一下**票證**。

   b 選取 Jira 端點。

   c 輸入 Jira 專案和問題類型。

d 輸入將收到票證的人員的電子郵件地址。

e 輸入票證的摘要和說明，然後按一下**儲存**。



4 儲存管線，然後啟用並執行此管線。

5 測試票證。

a 變更工作資訊，以包含導致工作失敗的資料。

b 儲存管線，並再次執行。

c 按一下**執行**，並確認管線失敗。

d 在執行中，確認 Code Stream 已建立並傳送票證。

e 將工作資訊變更回原先狀態以進行更正，然後再次執行管線並確保其會成功。

結果

恭喜您！當管線工作失敗時，使 Code Stream 建立 Jira 票證並將其指派給需要解決該問題的人員。

後續步驟

繼續新增通知，以向您的團隊警示管線相關資訊。

如何在 Code Stream 中復原我的部署

您可以透過使部署還原到故障部署管線中緊隨故障的先前穩定狀態的工作，將復原設定為一個管線。若要在發生故障時進行復原，請將復原管線連結至工作或階段。

根據您的角色，復原的原因可能有所不同。

- 做為發行工程師，我想要 Code Stream 驗證發行是否成功，以便我瞭解是否應該繼續發行還是應該復原。可能的故障包含工作故障、UserOps 拒絕、超出度量臨界值。
- 做為環境擁有者，我想重新部署先前的發行，以便可以快速將環境還原為已知良好狀態。
- 做為環境擁有者，我想支援復原藍綠部署，以便最大程度地縮短因發行失敗導致的停機時間。

使用智慧管線範本建立 CD 管線並按一下復原選項時，會自動將復原新增至管線中的工作。在此使用案例中，您將使用智慧管線範本為使用輪流升級部署模型部署到 Kubernetes 叢集的應用程式定義復原。智慧管線範本會建立一個部署管線和一或多個復原管線。

- 在部署管線中，如果 [更新部署] 或 [確認部署] 工作失敗，則需要復原。
- 在復原管線中，會使用舊映像更新部署。

您也可以使用空白範本手動建立復原管線。建立復原管線之前，您需要計劃復原流程。如需有關復原的更多背景資訊，請參閱在 Code Stream 中規劃復原。

必要條件

- 確認您是 Code Stream 中的專案的成員。如果您不是管理員，請要求 Code Stream 管理員將您新增為專案成員。請參閱如何在 Code Stream 中新增專案。
- 設定 Kubernetes 叢集，以便管線在其中部署應用程式。設定一個開發叢集和一個生產叢集。
- 確認您具有 Docker 登錄設定。
- 識別專案以便對所有工作進行分組，包括您的管線、端點和儀表板。
- 請按使用智慧管線範本前在 Code Stream 中規劃 CICD 原生建置的 CD 部分所述自行熟悉 CD 智慧範本，例如：
  - 建立 Kubernetes 開發端點和生產端點，以將應用程式映像部署到 Kubernetes 叢集。
  - 準備建立命名空間、服務和部署的 Kubernetes YAML 檔案。如果需要從專有存放庫下載映像，YAML 檔案必須包含具有 Docker 組態密碼的區段。

程序

1 按一下**管線 > 新增管線 > 智慧範本 > 持續交付**。

2 在智慧管線範本中輸入資訊。

   a 選取專案。

   b 輸入管線名稱，例如 **`RollingUpgrade-Example`**。

   c 選取應用程式的環境。若要將復原新增至部署，您必須選取**生產**。

d 按一下**選取**，選擇 Kubernetes YAML 檔案，然後按一下**處理**。

智慧管線範本會顯示可用的服務和部署環境。

e 選取管線將用於部署的服務。

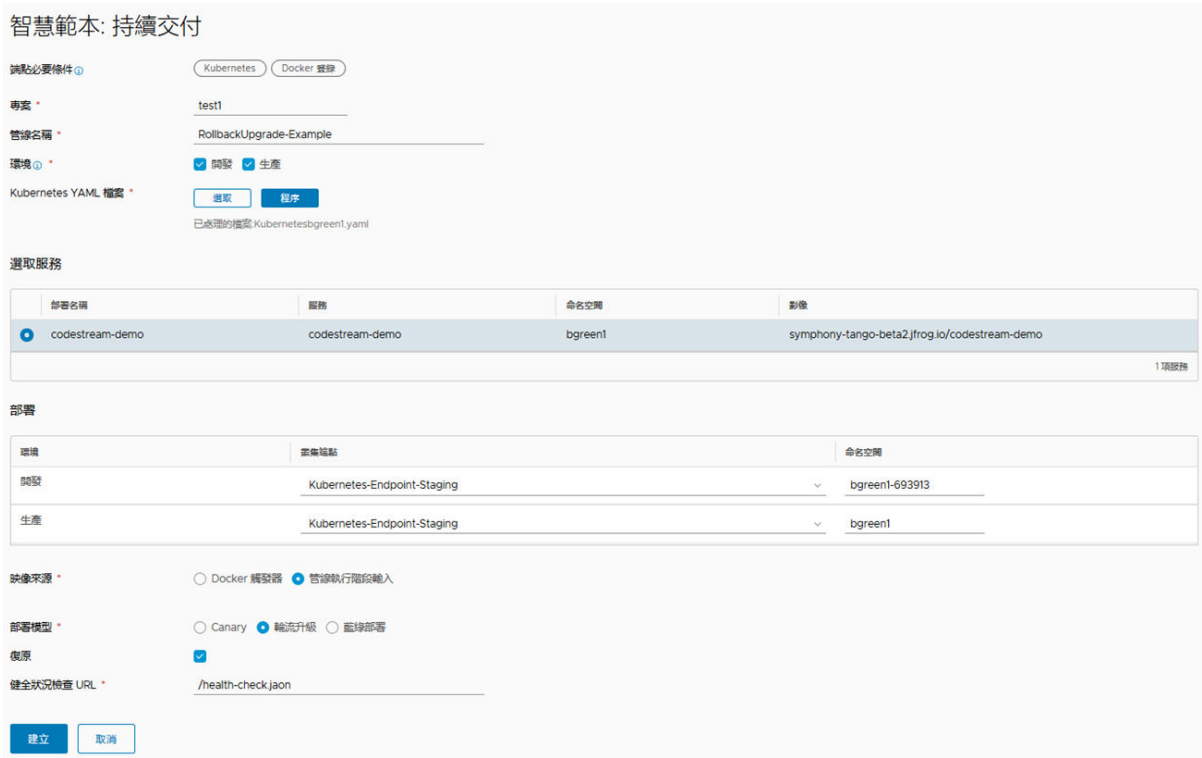
f 選取用於開發環境和生產環境的叢集端點。

g 針對映像來源，選取**管線執行期間輸入**。

h 針對部署模型，選取**輪流升級**。

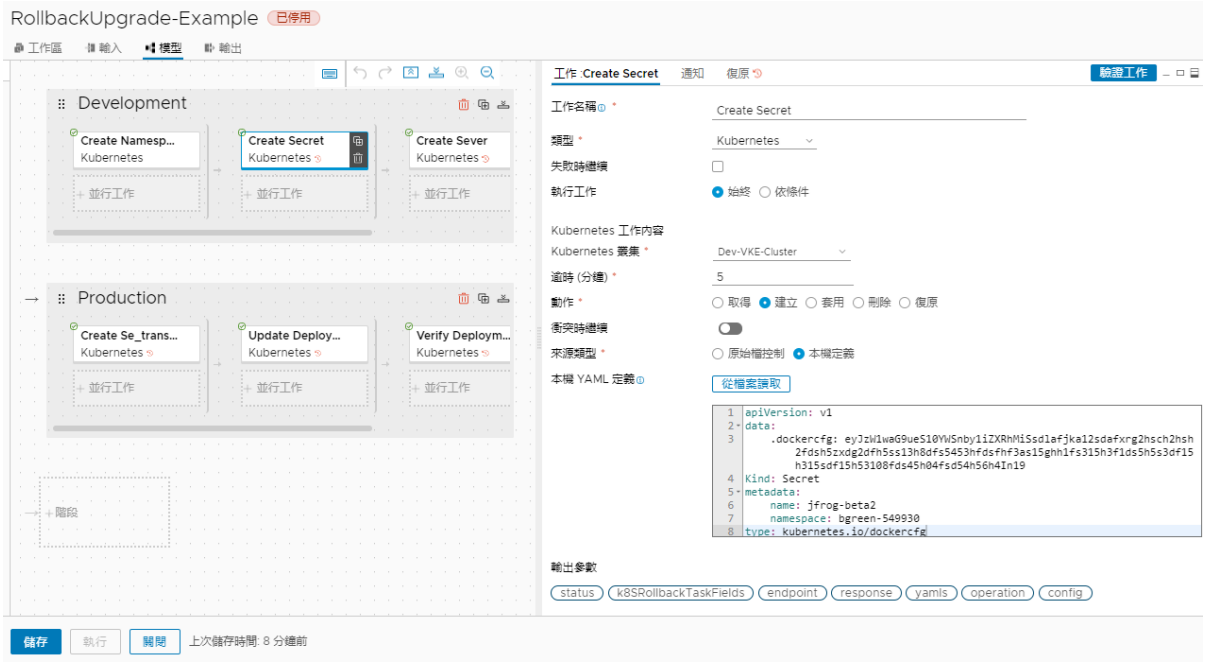
i 按一下**復原**。

j 提供**健全狀況檢查 URL**。



3 若要建立名為 RollbackUpgrade-Example 的管線，請按一下**建立**。

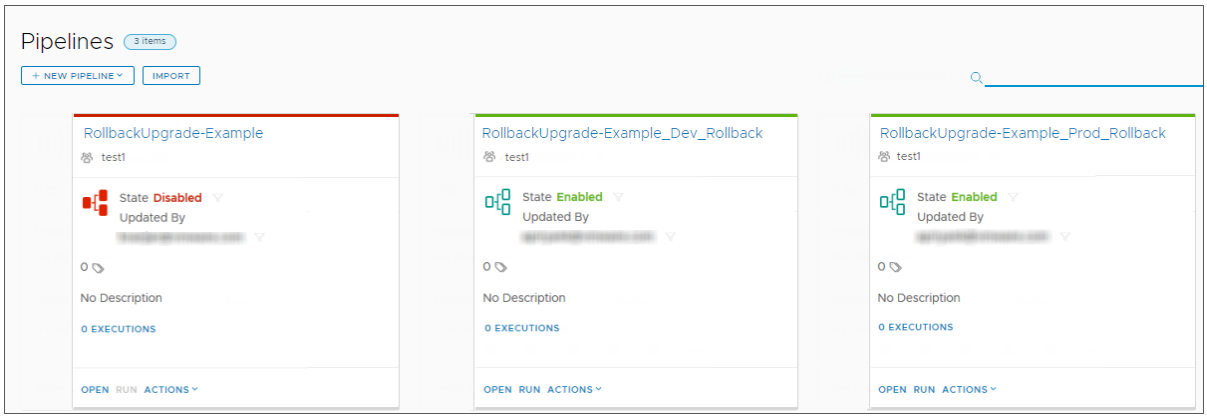
將顯示名為 RollbackUpgrade-Example 的管線，並且復原圖示會顯示在可於開發階段和生產階段中復原的工作上。



4 關閉管線。

在 [管線] 頁面中，會顯示已建立的管線以及管線中每個階段的新管線。

- RollingUpgrade-Example。 Code Stream 預設會停用已建立的管線，這可確保您在執行管線前先進行檢閱。
- RollingUpgrade-Example_Dev_Rollback。如果開發階段中的工作 (例如**建立服務**、、**建立密碼**、**建立部署**和**驗證部署**) 失敗，會叫用此復原開發管線。為確保復原開發工作， Code Stream 預設會啟用復原開發管線。
- RollingUpgrade-Example_Prod_Rollback。如果生產階段中的工作 (例如**部署階段 1**、**驗證階段 1**、**部署推出階段**、**完成推出階段**和**驗證推出階段**) 失敗，會叫用此復原生產管線。為確保復原生產工作， Code Stream 預設會啟用復原生產管線。

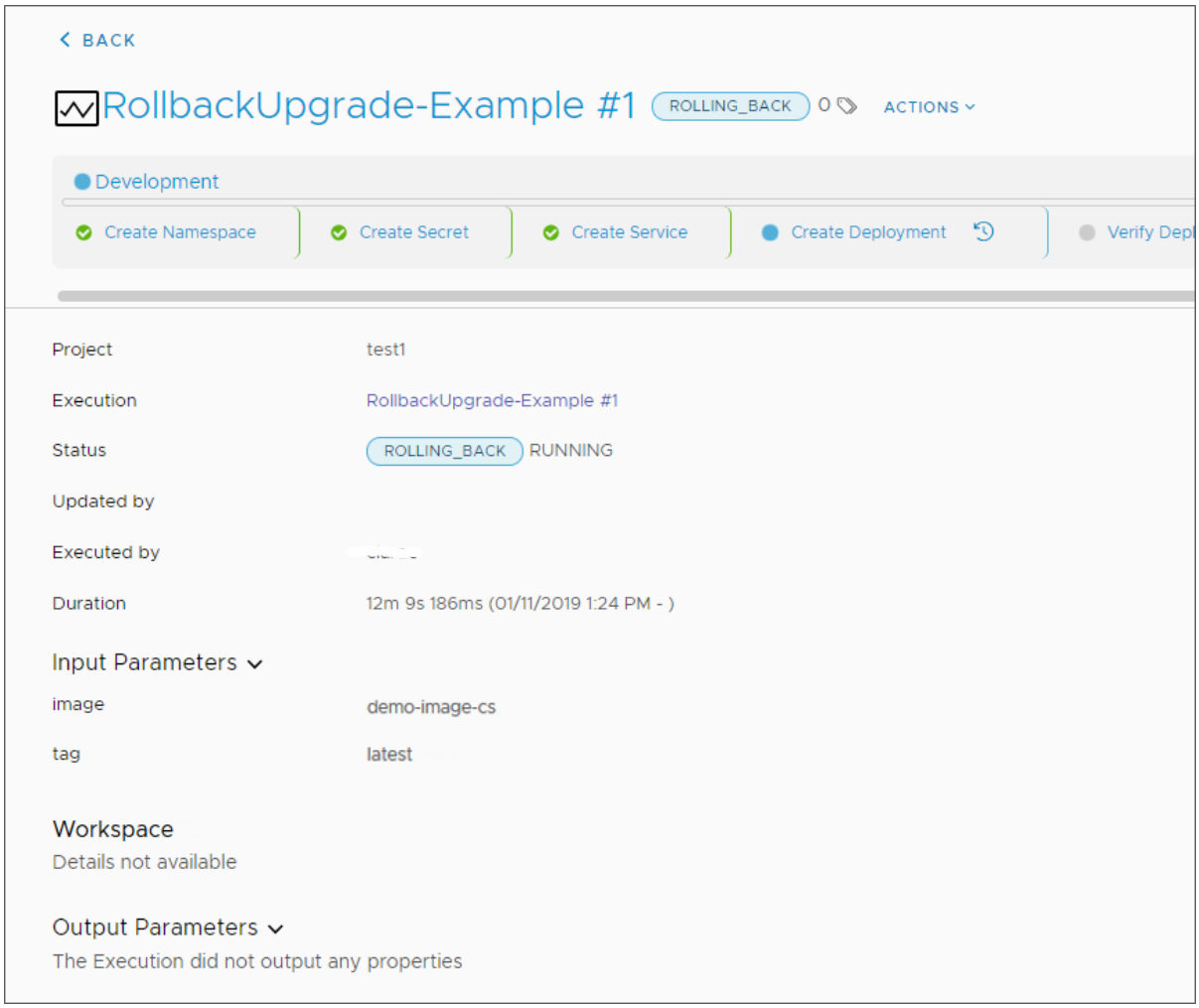


5 啟用並執行已建立的管線。

開始執行時， Code Stream 會提示您提供輸入參數。您需要為所使用的 Docker 存放庫中的端點提供映像和標籤。

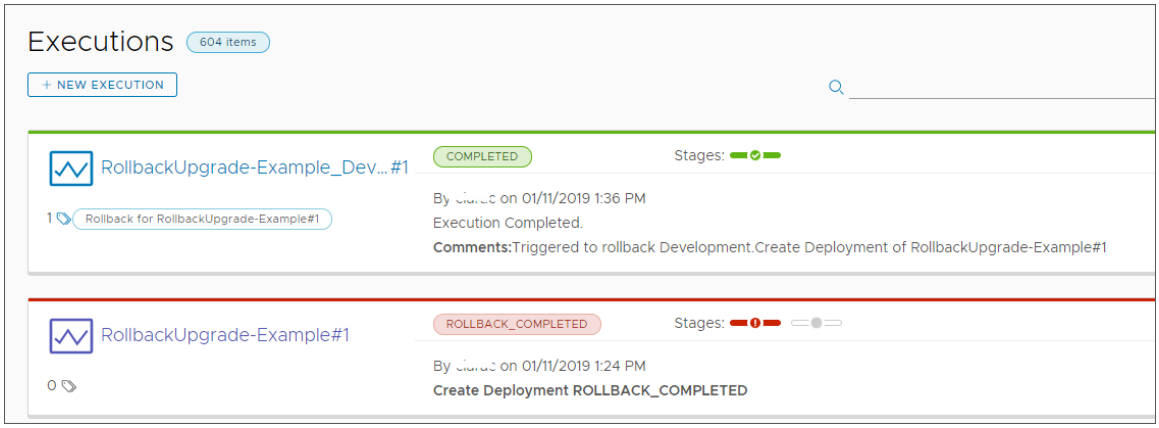
6 在 [執行] 頁面中，選取**動作 > 檢視執行**，然後監視管線執行。

管線將從 **RUNNING** 狀態開始，並經過開發階段中的多項工作。如果管線無法執行開發階段中的工作，則會觸發名為 RollingUpgrade-Example_Dev_Rollback 的管線並復原部署，且管線狀態會變更為 **ROLLING_BACK**。



復原後，[執行] 頁面會列出兩個 RollingUpgrade-Example 管線執行。

- 您建立的管線已復原，並顯示 **ROLLBACK_COMPLETED**。
- 觸發並執行復原的復原開發管線會顯示 **COMPLETED**。



結果

恭喜您！您已成功定義包含復原的管線並看到 Code Stream 會在發生故障時復原管線。

在 Code Stream 中規劃原生建置、整合及交付程式碼

4

透過使用為您建立 CICD、CI 或 CD 管線的原生功能使 Code Stream 建置、整合及交付程式碼之前，請規劃原生建置。然後，您可以透過使用其中一個智慧管線範本或手動新增階段和工作來建立管線。

為了規劃持續整合和持續交付建置，我們納入了幾個範例以向您顯示如何操作。這些計劃說明了可協助您在建置管線時有效地準備和使用原生建置功能的必要條件和概觀。

本章節討論下列主題：

設定管線工作區

若要執行持續整合工作和自訂工作，必須為 Code Stream 管線設定工作區。

在管線工作區中，選取**類型**為 Docker 或 Kubernetes，並提供相應的端點。Docker 和 Kubernetes 平台管理 Code Stream 為執行持續整合 (CI) 工作或自訂工作而部署的容器的整個生命週期。

- Docker 工作區需要 Docker 主機端點、產生器映像 URL、映像登錄、工作目錄、快取、環境變數、CPU 限制和記憶體限制。您還可以建立 Git 存放庫的複製品。
- Kubernetes 工作區需要 Kubernetes API 端點、產生器映像 URL、映像登錄、命名空間、NodePort、持續性磁碟區宣告 (PVC)、工作目錄、環境變數、CPU 限制和記憶體限制。您還可以建立 Git 存放庫的複製品。

管線工作區組態包含許多一般參數以及特定於工作區類型的其他參數，如下表所述。

表 4-1. 工作區區域、詳細資料和可用性

選取項目	說明	詳細資料和可用性
類型	工作區的類型。	可用於 Docker 或 Kubernetes。
主機端點	執行持續整合 (CI) 工作和自訂工作的主機端點。	選取 Docker 主機端點時，可用於 Docker 工作區。 選取 Kubernetes API 端點時，可用於 Kubernetes 工作區。
產生器映像 URL	產生器映像的名稱與位置。使用此映像在 Docker 主機和 Kubernetes 叢集上建立容器。持續整合 (CI) 工作和自訂工作在此容器內執行。	範例：**`fedora:latest`** 產生器映像必須具有 `curl` 或 `wget`。
映像登錄	如果產生器映像在登錄中可用，並且登錄需要認證，則必須建立映像登錄端點，然後在此處選取該端點，以便從登錄中提取映像。	可用於 Docker 和 Kubernetes 工作區。
工作目錄	工作目錄是容器內執行持續整合 (CI) 工作步驟的位置，也是 Git Webhook 觸發管線執行時複製代碼的位置。	可用於 Docker 或 Kubernetes。
命名空間	如果未輸入命名空間， Code Stream 將在您提供的 Kubernetes 叢集中建立唯一名稱。	特定於 Kubernetes 工作區。
Proxy	為了與 Kubernetes 叢集中的工作區網繭進行通訊，Code Stream 在命名空間 `codestream-proxy` 中為每個 Kubernetes 叢集部署一個 Proxy 執行個體。您可以根據叢集組態選取 **NodePort** 或 **LoadBalancer** 類型。 選取哪個選項取決於已部署 Kubernetes 叢集的特性。 ■ 通常，如果端點中指定的 Kubernetes API 伺服器 URL 透過其中一個主要節點公開，則選取 **NodePort**。 ■ 如果 Kubernetes API 伺服器 URL 透過負載平衡器公開，例如 Amazon EKS (Elastic Kubernetes Service)，則選取 **LoadBalancer**。	
NodePort	Code Stream 使用 NodePort 與 Kubernetes 叢集中執行的容器進行通訊。 如果未選取連接埠，則 Code Stream 將使用 Kubernetes 指派的暫時連接埠。您必須確保防火牆規則的組態允許輸入暫時連接埠範圍 (30000-32767)。 如果輸入連接埠，則必須確保叢集中的其他服務尚未使用該連接埠，並且防火牆規則允許該連接埠。	特定於 Kubernetes 工作區。
持續性磁碟區宣告	為 Kubernetes 工作區提供一種在各個管線執行過程中持久保留檔案的方法。提供持續性磁碟區宣告名稱時，它可以儲存記錄、構件和快取。 如需有關建立持續性磁碟區宣告的詳細資訊，請參閱 Kubernetes 說明文件，網址為 `https://kubernetes.io/docs/concepts/storage/persistent-volumes/`。	特定於 Kubernetes 工作區。

表 4-1. 工作區區域、詳細資料和可用性 (續)

選取項目	說明	詳細資料和可用性
環境變數	管線執行時，此處傳遞的索引鍵-值配對將可用於該管線中的所有持續整合 (CI) 工作和自訂工作。	可用於 Docker 或 Kubernetes。 可以在此處傳遞對變數的參考。 在工作區中提供的環境變數將傳遞到管線中的所有持續整合 (CI) 工作和自訂工作。 如果未在此處傳遞環境變數，則必須將這些變數明確傳遞到管線中的每個持續整合 (CI) 工作和自訂工作。
CPU 限制	持續整合 (CI) 容器或自訂工作容器的 CPU 資源限制。	預設值為 1。
記憶體限制	持續整合 (CI) 容器或自訂工作容器的記憶體限制。	單位為 MB。
Git 複製	如果選取 **Git 複製**，並且 Git Webhook 叫用管線，則會將代碼複製到工作區 (容器)。	如果未啟用 **Git 複製**，則必須在管線中設定另一個明確持續整合 (CI) 工作以便先複製代碼，然後再執行其他步驟，例如建置和測試。

快取	在 Code Stream 工作區中，可以快取一組目錄或檔案，以便加快後續管線執行。這些目錄的範例包括 `.m2` 和 `npm_modules`。如果不需要在管線執行之間快取資料，則不需要持續性磁碟區宣告。
	將快取容器中的檔案或目錄之類的構件，以便在管線執行之間重複使用。例如，可以快取 `node_modules` 或 `.m2` 資料夾。**快取**接受路徑清單。
	例如：
	`workspace:`
	`  type: K8S`
	`  endpoint: K8S-Micro`
	`  image: fedora:latest`
	`  registry: Docker Registry`
	`  path: ''`
	`  cache:`
	`  - /path/to/m2`
	`  - /path/to/node_modules`
		特定於工作區類型。
		在 Docker 工作區中，透過使用 Docker 主機中的共用路徑持久保留快取的資料、構件和記錄來實現**快取**。
		在 Kubernetes 工作區中，若要啟用**快取**，必須提供持續性磁碟區宣告。否則，**快取**將不可用。

在管線工作區中使用 Kubernetes API 端點時， Code Stream 會建立必要的 Kubernetes 資源 (如 ConfigMap、密碼和網繭) 以執行持續整合 (CI) 工作或自訂工作。 Code Stream 使用 NodePort 與容器進行通訊。

若要在各個管線執行過程中共用資料，您必須提供持續性磁碟區宣告， Code Stream 會將此持續性磁碟區宣告掛接到容器以儲存資料，並將其用於後續管線執行。

使用智慧管線範本前在 Code Stream 中規劃 CICD 原生建置

若要在 Code Stream 中建立持續整合和持續交付 (CICD) 管線，您可以使用 CICD 智慧管線範本。若要規劃 CICD 原生建置，您需要在此範例計劃中建立管線之前收集智慧管線範本的資訊。

若要建立 CICD 管線，您必須規劃管線的持續整合 (CI) 和持續交付 (CD) 階段。

在智慧管線範本中輸入資訊並將其儲存後，它會建立包含階段和工作的管線。它還會根據所選環境類型 (例如，開發和生產) 指示映像的部署目標。管線會發佈容器映像，並執行運行所需的動作。執行管線後，您可以監控管線執行中的趨勢。

當管線包含來自 Docker Hub 的映像時，您必須在執行管線之前確保該映像已內嵌 `cURL` 或 `wget`。當管線執行時， Code Stream 將下載使用 `cURL` 或 `wget` 執行命令的二進位檔案。

如需設定工作區的相關資訊，請參閱設定管線工作區。

規劃持續整合 (CI) 階段

若要規劃管線的 CI 階段，您將設定外部和內部需求，並確定智慧管線範本的 CI 部分所需的資訊。以下是摘要。

此範例使用 Docker 工作區。

您將需要的端點和存放庫：

- Git 原始程式碼存放庫，開發人員將代碼簽入到該存放庫中。當開發人員認可變更時， Code Stream 會將最新程式碼提取到管線。
- 開發人員原始程式碼所在的存放庫的 Git 端點。
- 用於 Docker 建置主機的 Docker 端點，將在容器內執行建置命令。
- Kubernetes 端點，可讓 Code Stream 將您的映像部署到 Kubernetes 叢集。
- 產生器映像，用於建立執行持續整合測試的容器。
- 映像登錄端點，可讓 Docker 建置主機從中提取產生器映像。

您將需要專案的存取權。專案會對所有工作進行分組，包括您的管線、端點和儀表板。確認您是 Code Stream 中的專案的成員。如果您不是管理員，請要求 Code Stream 管理員將您新增為專案成員。請參閱如何在 Code Stream 中新增專案。

您將需要可讓 Code Stream 使用 Git 觸發器的 Git webhook，當開發人員 commit 程式碼變更時，會觸發管線。請參閱如何在 Code Stream 中使用 Git 觸發器執行管線。

您的建置工具集：

- 您的建置類型，例如 Maven。
- 您使用的所有後續處理建置工具，例如 JUnit、JaCoCo、Checkstyle 和 FindBugs。

您的發佈工具：

- 將部署建置容器的工具，例如 Docker。
- 映像標籤，即認可識別碼或組建編號。

您的建置工作區：

- Docker 建置主機，即 Docker 端點。
- 映像登錄。管線的 CI 部分會從選取的登錄端點提取映像。容器會執行 CI 工作並部署映像。如果登錄需要認證，您必須建立映像登錄端點，然後在此處選取該端點，以便主機可以從登錄提取映像。

- 產生器映像的 URL，用於建立執行持續整合工作的容器。

規劃持續交付 (CD) 階段

若要規劃管線的 CD 階段，您將設定外部和內部需求，並確定要在智慧管線範本的 CD 部分中輸入的資訊。

您將需要的端點：

- Kubernetes 端點，可讓 Code Stream 將您的映像部署到 Kubernetes 叢集。

環境類型和檔案：

- Code Stream 將在其中部署應用程式的所有環境類型，例如 Dev 和 Prod。智慧管線範本會根據您選取的環境類型建立管線中的階段和工作。

  表 4-2. CICD 智慧管線範本建立的管線階段

管線內容	執行的作業
建置 - 發佈階段	建置並測試您的程式碼、建立產生器映像，以及將映像發佈到 Docker 主機。
開發階段	使用開發 Amazon Web Services (AWS) 叢集來建立並部署您的映像。在此階段中，您可以在叢集上建立命名空間，並建立秘密金鑰。
生產階段	使用 VMware Tanzu Kubernetes Grid Integrated Edition (先前稱為 VMware Enterprise PKS) 的生產版本，將映像部署到生產 Kubernetes 叢集。

- 在 CICD 智慧管線範本的 CD 區段中選取的 Kubernetes YAML 檔案。

  Kubernetes YAML 檔案包含命名空間、服務和部署的三個必填區段，以及密碼的一個可選區段。如果您打算透過從專有存放庫下載映像來建立管線，則必須包含具有 Docker 組態密碼的區段。 如果建立的管線僅使用公開可用的映像，則不需要密碼。下列範例 YAML 檔案包含四個區段。

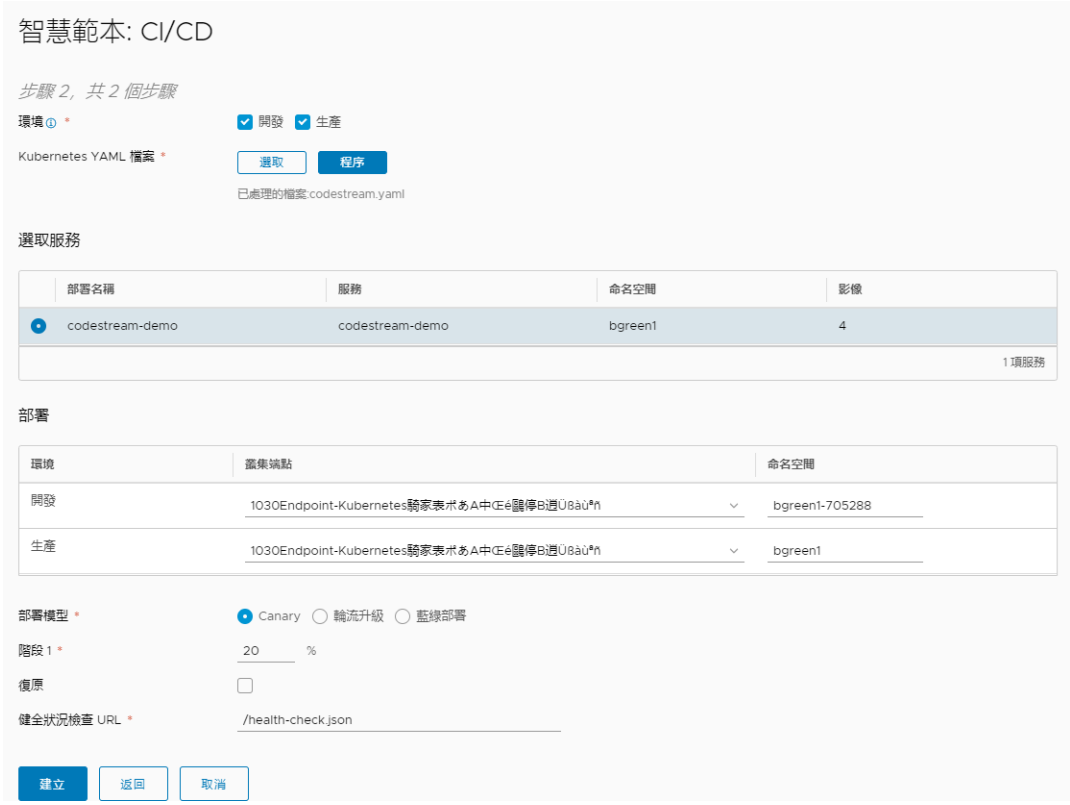
```
apiVersion: v1
kind: Namespace
metadata:
  name: codestream
  namespace: codestream
---
apiVersion: v1
data:
  .dockerconfigjson:
eyJhdXRocyI6eyJodHRwczovL2luZ1234567890lci5pby92MS8iOnsidXNlcm5hbWUiOiJhdXRvbWF0aW9uYmV0YSI
sInBhc3N3b3JkIjoiVk13YXJlQDEyMyIsImVtYWlsIjoiYXV0b21hdGlvbmJldGF1c2VyQGdtYWlsLmNvbSIsImF1dG
giOiJZWFYwYjIxaGRHbHZibUpsZEdFNlZrMTNZWEpsUURFeU13PT0ifX19
kind: Secret
metadata:
  name: dockerhub-secret
  namespace: codestream
type: kubernetes.io/dockerconfigjson
---
apiVersion: v1
kind: Service
metadata:
  name: codestream-demo
```

```
  namespace: codestream
  labels:
    app: codestream-demo
spec:
  ports:
    - port: 80
  selector:
    app: codestream-demo
    tier: frontend
  type: LoadBalancer
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: codestream-demo
  namespace: codestream
  labels:
    app: codestream-demo
spec:
  replicas: 10
  selector:
    matchLabels:
      app: codestream-demo
      tier: frontend
  template:
    metadata:
      labels:
        app: codestream-demo
        tier: frontend
    spec:
      containers:
      - name: codestream-demo
        image: automationbeta/codestream-demo:01
        ports:
        - containerPort: 80
          name: codestream-demo
      imagePullSecrets:
      - name: dockerhub-secret
```

備註 Kubernetes YAML 檔案也用於 CD 智慧管線範本，例如在下列使用案例範例中：

- 如何在 Code Stream 中將應用程式部署至藍綠部署
- 如何在 Code Stream 中復原我的部署
- 如何在 Code Stream 中使用 Docker 觸發器執行持續交付管線

若要在智慧範本中套用該檔案，請按一下**選取**，選取 Kubernetes YAML 檔案，然後按一下**處理**。智慧管線範本會顯示可用的服務和部署環境。選取服務、叢集端點和部署策略。例如，若要使用 Canary 部署模型，請選取 **Canary**，並輸入部署階段的百分比：

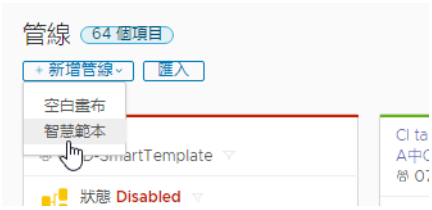


若要查看使用智慧管線範本為藍綠部署建立管線的範例，請參閱如何在 Code Stream 中將應用程式部署至藍綠部署。

如何使用智慧管線範本建立 CICD 管線

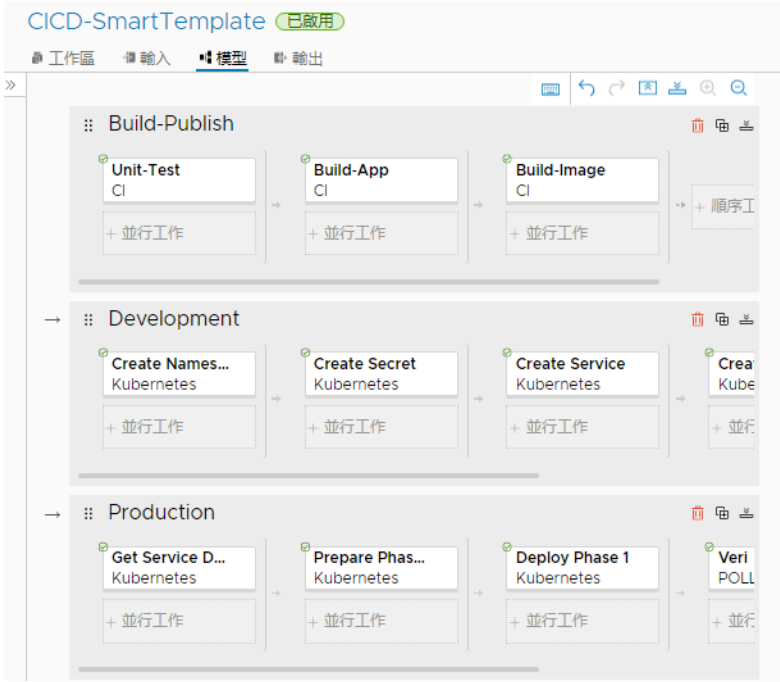
收集所有資訊並設定所需內容後，您可以從 CICD 智慧管線範本建立管線。

在管線中，您將選取**新增管線 > 智慧範本**。



將選取 CICD 智慧管線範本。

您將填寫範本，然後儲存管線及其建立的階段。如果需要進行任何最終變更，您可以編輯管線並將其儲存。



然後，您將啟用管線並加以執行。執行後，您可以查看以下內容：

- 確認您的管線已成功。按一下**執行**，並搜尋您的管線。如果失敗，請更正任何錯誤並再執行一次。
- 確認 Git webhook 運作正常。Git **活動**索引標籤隨即顯示事件。按一下**觸發器 > Git > 活動**。
- 查看管線儀表板，並檢查趨勢。按一下**儀表板**，並搜尋管線儀表板。您也可以建立自訂儀表板對其他 KPI 進行報告。

如需詳細範例，請參閱如何在 Code Stream 中將 GitHub 或 GitLab 存放庫中的程式碼持續整合到管線。

使用智慧管線範本前在 Code Stream 中規劃持續整合原生建置

若要在 VMware Code Stream 中建立持續整合 (CI) 管線，您可以使用持續整合智慧管線範本。若要規劃持續整合原生建置，您需要在此範例計劃中建立管線之前收集智慧管線範本的資訊。

當您填寫智慧管線範本時，會在存放庫中建立持續整合管線，並執行動作以便管線可以執行。執行管線後，您可以監控管線執行中的趨勢。

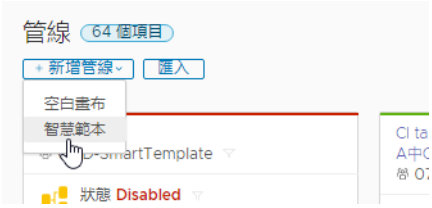
在使用持續整合智慧管線範本之前規劃建置：

- 識別專案以便對所有工作進行分組，包括您的管線、端點和儀表板。
- 按照使用智慧管線範本前在 Code Stream 中規劃 CICD 原生建置的持續交付部分中所述收集建置資訊。

  例如，新增一個 Kubernetes 端點， Code Stream 將在其中部署容器。

然後，使用持續整合智慧管線範本建立管線。

在管線中，選取**智慧範本**。



選取持續整合智慧管線範本。

若要儲存管線及其建立的階段，請填寫範本，然後輸入管線的名稱。若要儲存管線及其建立的階段，請按一下**建立**。

Code Stream 管線工作區支援使用 Docker 和 Kubernetes 執行持續整合工作和自訂工作。

如需設定工作區的相關資訊，請參閱設定管線工作區。

若要進行任何最終變更，您可以編輯管線。然後，您可以啟用管線並加以執行。管線執行後：

- 確認您的管線已成功。按一下**執行**，並搜尋您的管線。如果失敗，請更正任何錯誤並再執行一次。
- 確認 Git webhook 運作正常。Git **活動**索引標籤隨即顯示事件。按一下**觸發器** > **Git** > **活動**。
- 查看管線儀表板，並檢查趨勢。按一下**儀表板**，並搜尋管線儀表板。若要報告更多關鍵效能指標，您可以建立自訂儀表板。

如需詳細範例，請參閱如何在 Code Stream 中將 GitHub 或 GitLab 存放庫中的程式碼持續整合到管線。

使用智慧管線範本前在 Code Stream 中規劃持續交付原生建置

若要在 Code Stream 中建立持續交付 (CD) 管線，您可以使用持續交付智慧管線範本。若要規劃持續交付原生建置，您需要在此範例計劃中建立管線之前收集智慧管線範本的資訊。

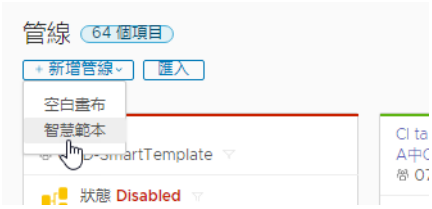
當您填寫智慧管線範本時，會在存放庫中建立持續交付管線，並執行動作以便管線可以執行。執行管線後，您可以監控管線執行中的趨勢。

在使用持續交付智慧管線範本之前規劃建置：

- 識別專案以便對所有工作進行分組，包括您的管線、端點和儀表板。
- 按照使用智慧管線範本前在 Code Stream 中規劃 CICD 原生建置的持續交付部分中所述收集建置資訊。例如：
  - 新增一個 Kubernetes 端點， Code Stream 將在其中部署容器。
  - 準備建立命名空間、服務和部署的 Kubernetes YAML 檔案。若要從專有存放庫下載映像，YAML 檔案必須包含具有 Docker 組態密碼的區段。

然後，使用持續交付智慧管線範本建立管線。

在管線中，選取**智慧範本**。



選取持續交付智慧管線範本。

填寫範本，然後輸入管線的名稱。若要儲存管線及其建立的階段，請按一下**建立**。

Code Stream 管線工作區支援使用 Docker 和 Kubernetes 執行持續整合工作和自訂工作。

如需設定工作區的相關資訊，請參閱設定管線工作區。

若要進行任何最終變更，您可以編輯管線。然後，您可以啟用管線並加以執行。管線執行後：

- 確認您的管線已成功。按一下**執行**，並搜尋您的管線。如果失敗，請更正任何錯誤並再執行一次。
- 確認 Git webhook 運作正常。Git **活動**索引標籤隨即顯示事件。按一下**觸發器 > Git > 活動**。
- 查看管線儀表板，並檢查趨勢。按一下**儀表板**，並搜尋管線儀表板。若要報告更多關鍵效能指標，您可以建立自訂儀表板。

如需詳細範例，請參閱如何在 Code Stream 中將 GitHub 或 GitLab 存放庫中的程式碼持續整合到管線。

手動新增工作前在 Code Stream 中規劃 CICD 原生建置

若要在 Code Stream 中建立持續整合和持續交付 (CICD) 管線，您可以手動新增階段和工作。若要規劃 CICD 原生建置，您將收集所需的資訊，然後建立管線並手動向其新增階段和工作。

您必須規劃管線的持續整合 (CI) 和持續交付 (CD) 階段。建立並執行管線後，您可以監控管線執行中的趨勢。

當管線包含來自 Docker Hub 的映像時，您必須在執行管線之前確保該映像已內嵌 `cURL` 或 `wget`。當管線執行時， Code Stream 將下載使用 `cURL` 或 `wget` 執行命令的二進位檔案。

Code Stream 管線工作區支援使用 Docker 和 Kubernetes 執行持續整合工作和自訂工作。

如需設定工作區的相關資訊，請參閱設定管線工作區。

規劃外部和內部需求

若要規劃管線的 CI 和 CD 階段，以下需求指明了在建立管線之前必須執行的動作。

此範例使用 Docker 工作區。

若要從此範例計劃建立管線，您將使用 Docker 主機、Git 存放庫、Maven 和數個後續處理建置工具。

您將需要的端點和存放庫：

- Git 原始程式碼存放庫，開發人員將代碼簽入到該存放庫中。當開發人員認可變更時， Code Stream 會將最新程式碼提取到管線。
- 用於 Docker 建置主機的 Docker 端點，將在容器內執行建置命令。
- 產生器映像，用於建立執行持續整合測試的容器。
- 映像登錄端點，可讓 Docker 建置主機從中提取產生器映像。

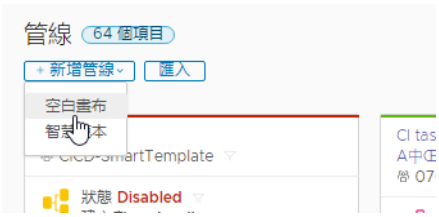
您將需要專案的存取權。專案會對所有工作進行分組，包括您的管線、端點和儀表板。確認您是 Code Stream 中的專案的成員。如果您不是管理員，請要求 Code Stream 管理員將您新增為專案成員。請參閱如何在 Code Stream 中新增專案。

您將需要可讓 Code Stream 使用 Git 觸發器的 Git webhook，當開發人員 commit 程式碼變更時，會觸發管線。請參閱如何在 Code Stream 中使用 Git 觸發器執行管線。

如何建立 CICD 管線並設定工作區

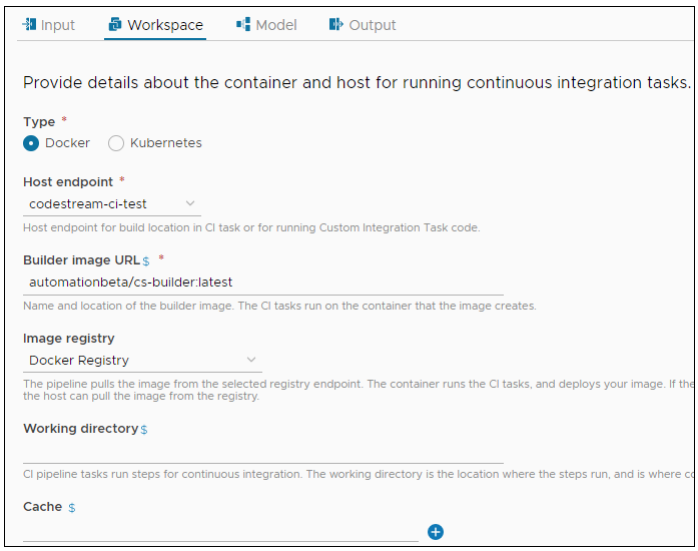
您將需要建立管線，然後設定工作區、管線輸入參數和工作。

若要建立管線，您可以按一下**管線 > 新增管線 > 空白畫布**。



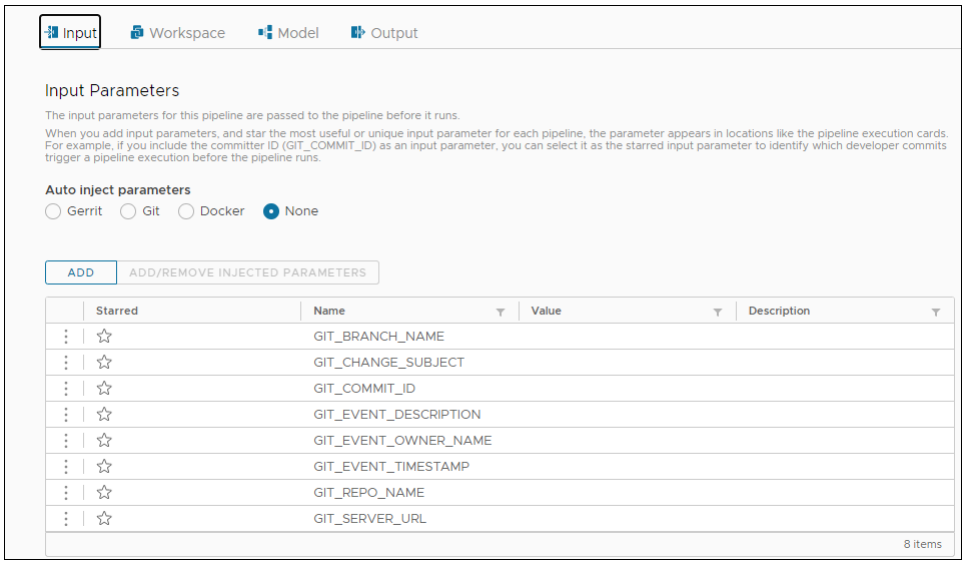
在 [工作區] 索引標籤上，輸入持續整合資訊：

- 包含您的 Docker 建置主機。
- 輸入產生器映像的 URL。
- 選取映像登錄端點，以便管線可從中提取映像。容器會執行 CI 工作並部署映像。如果登錄需要認證，您必須先建立映像登錄端點，然後在此處選取該端點，以便主機可以從登錄提取映像。
- 新增必須快取的構件。若要成功建置，可以將目錄等構件下載為相依性構件。快取是這些構件所在的位置。例如，相依構件可包含 Maven 的 `.m2` 目錄和 Node.js 的 `node_modules` 目錄。為了在建置期間節省時間，這些目錄會跨管線執行快取。



在 [輸入] 索引標籤上，設定管線輸入參數。

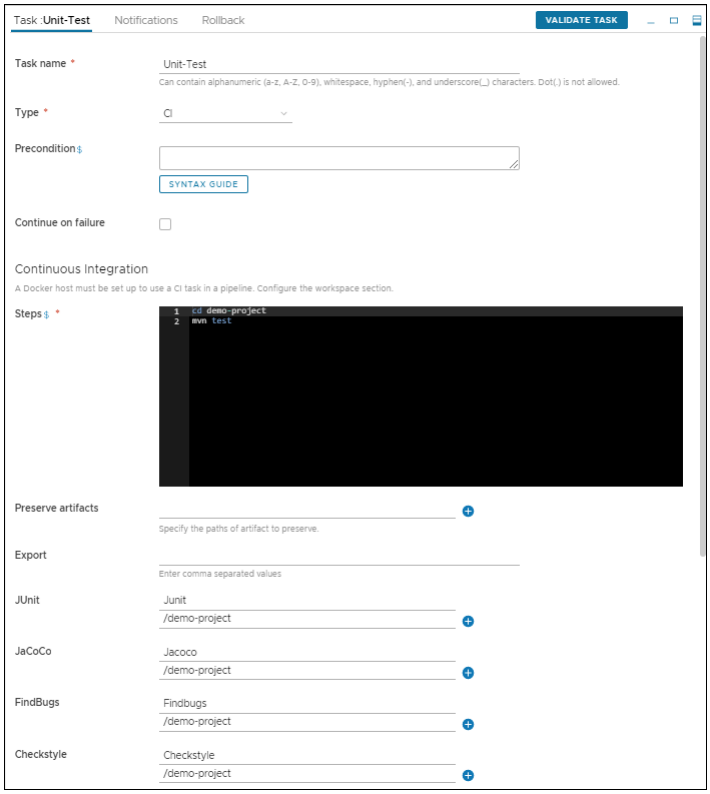
- 如果您的管線將使用來自 Git、Gerrit 或 Docker 觸發器事件的輸入參數，請選取自動插入參數的觸發器類型。事件可包含 Gerrit 或 Git 的變更主題，或 Docker 的事件擁有者名稱。如果管線將不會使用從該事件傳遞的任何輸入參數，請將自動插入參數設定為**無**。
- 若要將值和說明套用至管線輸入參數，請按一下三個垂直點，然後按一下**編輯**。您輸入的值將用作工作、階段或通知的輸入。
- 若要新增管線輸入參數，請按一下**新增**。例如，您可以新增 `approvers` 來顯示每次執行的預設值，但可以進行覆寫以在執行階段顯示不同的核准者。
- 若要新增或移除插入的參數，請按一下**新增/移除插入的參數**。例如，移除未使用的參數來減少結果頁面上雜亂的內容，並僅顯示正在使用的輸入參數。



設定管線以測試程式碼：

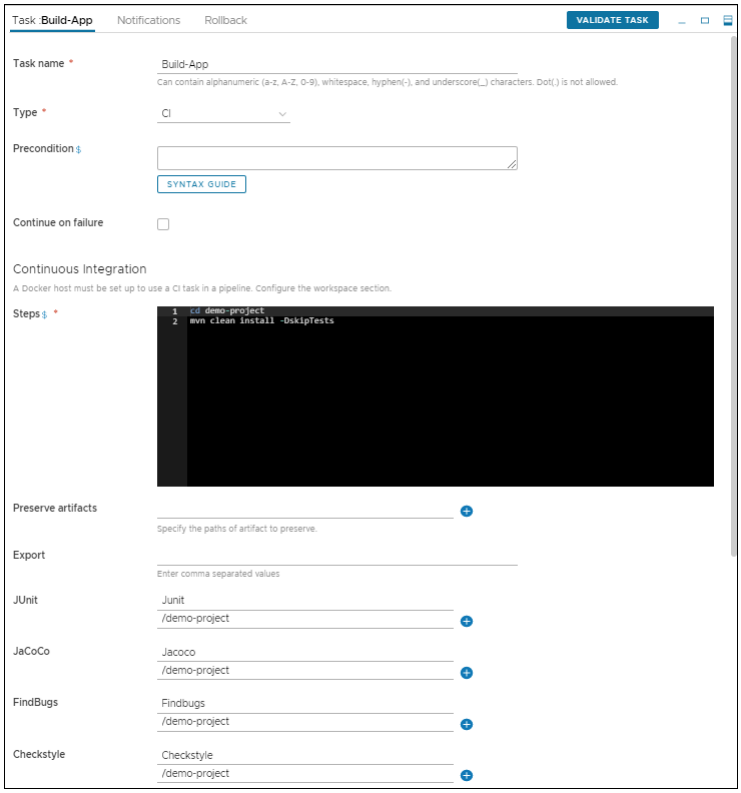
- 新增並設定 CI 工作。

- 包含對程式碼執行 `mvn test` 的步驟。
- 若要在工作執行後識別任何問題，請執行後續處理建置工具，例如 JUnit 和 JaCoCo、FindBugs 和 Checkstyle。



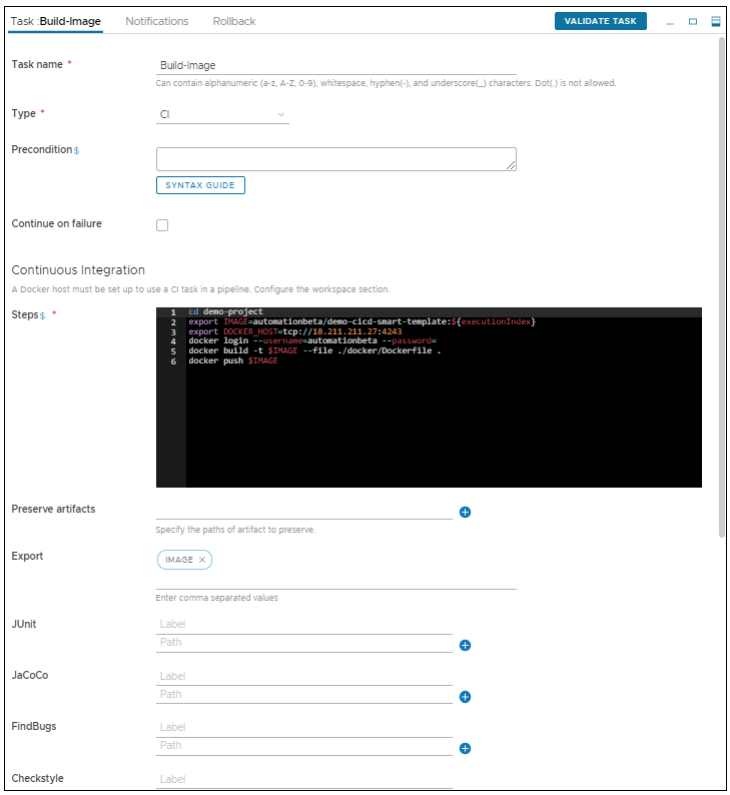
設定管線以建置程式碼：

- 新增並設定 CI 工作。
- 包括對代碼執行 `mvn clean install` 的步驟。
- 包括位置和 JAR 檔案名稱，以便保留您的構件。



設定管線以將映像發佈到 Docker 主機：

- 新增並設定 CI 工作。
- 新增將認可、匯出、建置和推送映像的步驟。
- 新增 `IMAGE` 的匯出金鑰，以供下一個工作耗用。



設定工作區、輸入參數、測試工作並建置工作之後，儲存您的管線。

如何啟用並執行管線

透過階段和工作設定管線後，您可以儲存並啟用管線。

然後，等待管線執行並完成，然後確認其是否成功。如果失敗，請更正任何錯誤並再執行一次。

管線成功後，您可能想要確認以下事項：

- 檢查管線執行並檢視工作步驟的結果。
- 在管線執行的工作區中，找到有關容器和已複製 Git 存放庫的詳細資料。
- 在工作區中，查看後續處理工具結果並檢查錯誤、程式碼涵蓋範圍、錯誤和樣式問題。
- 確認您的構件已保留。此外，請確認已使用 IMAGE 名稱和值匯出映像。
- 移至 Docker 存放庫，並確認管線已發佈您的容器。

如需顯示 Code Stream 如何持續整合程式碼的詳細範例，請參閱如何在 Code Stream 中將 GitHub 或 GitLab 存放庫中的程式碼持續整合到管線。

在 Code Stream 中規劃復原

如果管線執行失敗，您可以使用復原將環境復原為先前穩定的狀態。若要使用復原，需規劃復原流程並瞭解如何實作。

復原流程規定在部署中反向失敗所需的步驟。此流程採用復原管線的形式，此管線包含一或多個連續工作，這些工作視執行和失敗的部署類型而有所不同。例如，傳統應用程式的部署和復原不同於 容器應用程式的部署和復原。

若要還原為正常的部署狀態，復原管線通常包含下列工作：

- 清理狀態或環境。
- 執行使用者指定的指令碼以還原變更。
- 部署先前的部署修訂。

若要將復原新增至現有部署管線，請在執行部署管線之前，將復原管線連結至要復原的部署管線中的工作或階段。

如何設定復原

若要在部署中設定復原，您需要：

- 建立部署管線。
- 識別在部署管線中將觸發復原的潛在故障點，以便您可以連結復原管線。例如，您可以將復原管線連結至部署管線中的條件或輪詢工作類型，以檢查上一個工作是否成功完成。如需條件工作的相關資訊，請參閱如何在 Code Stream 中使用 [條件] 工作中的變數繫結來執行或停止管線。
- 確定將觸發復原管線的故障範圍，例如工作或階段故障。您還可以將復原連結至階段。
- 確定發生故障時要執行的復原工作。將建立包含這些工作的復原管線。

您可以手動建立復原管線，也可以使用 Code Stream 建立。

- 使用空白畫布，您可以手動建立遵循與現有部署管線平行流程的復原管線。然後，將復原管線連結至部署管線中 發生故障時觸發復原的一或多個工作。
- 透過智慧管線範本，您可以使用復原動作設定部署管線。然後， Code Stream 會自動建立一或多個預設復原管線，其中包含發生故障時復原部署的預先定義的工作。

如需有關如何使用智慧管線範本為 CD 管線設定復原的詳細範例，請參閱如何在 Code Stream 中復原我的部署。

如果我的部署管線有多個工作或階段具有復原，會怎樣

如果您有多個工作和階段新增了復原，請注意復原順序會有所不同。

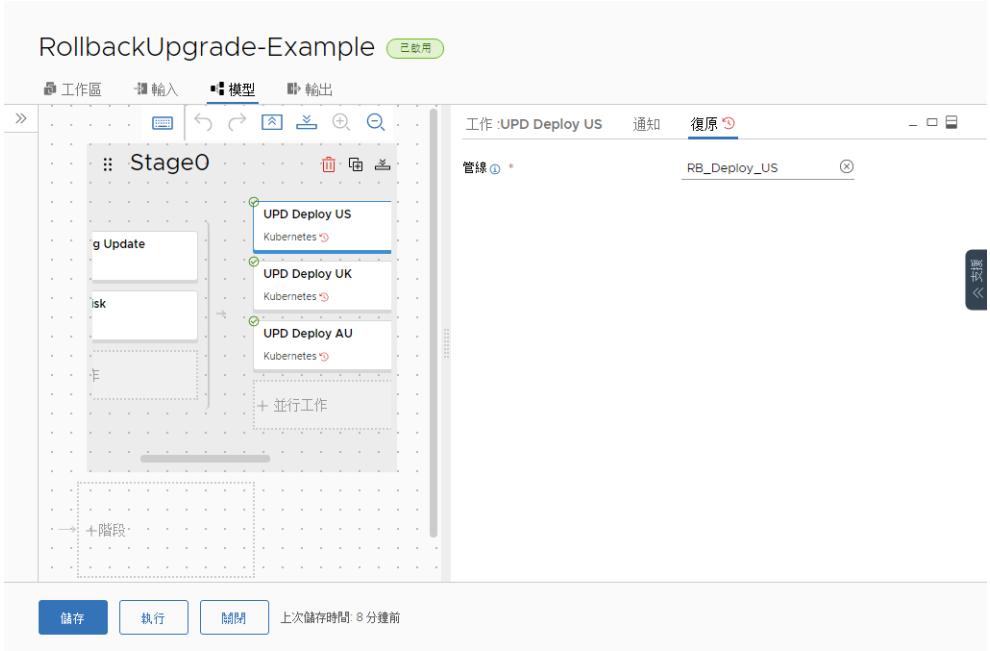
表 4-3. 判定復原順序

如果將復原新增至...	復原執行時間...
並行工作	如果其中一個並行工作失敗，則所有並行工作完成或失敗後會復原此工作。該工作失敗後，不會立即執行復原。
階段中的工作和階段	如果工作失敗，則執行此工作復原。如果工作是在一組並行工作中，則所有並行工作完成或失敗後會執行工作復原。工作復原完成或無法完成後，會執行階段復原。

考量具有下列階段或工作的管線：

- 具有復原的生產階段。
- 一組並行工作，且每個工作具有自己的復原。

名為 UPD Deploy US 的工作具有復原管線 RB_Deploy_US。如果 UPD Deploy US 失敗，則復原會遵循 RB_Deploy_US 管線中定義的流程。



如果 UPD Deploy US 失敗，RB_Deploy_US 管線會在 UPD Deploy UK 和 UPD Deploy AU 已完成或失敗後執行。UPD Deploy US 失敗後，不會立即執行復原。另外，由於生產階段也有復原，因此在 RB_Deploy_US 管線執行後，將執行階段復原管線。

使用 Code Stream 的教學課程

5

Code Stream 會建立模型和支援 DevOps 發行生命週期，持續測試應用程式並將其發行至開發環境和生產環境。

您已設定所需的所有項目，以便您可以使用 Code Stream。請參閱第 2 章 設定 Code Stream 以建立發佈程序模型。

現在，您可以建立管線，以便自動建置和測試開發人員程式碼，然後將此程式碼發佈至生產。您可以使 Code Stream 部署容器型或傳統的應用程式。

表 5-1. 在 DevOps 生命週期中使用 Code Stream

功能	可執行的操作範例
在 Code Stream 中使用原生建置功能。	建立持續整合和持續交付 (CICD)、持續整合 (CI) 和持續交付 (CD) 管線，以持續整合、容器化和交付代碼。 ■ 使用可為您建立管線的智慧管線範本。 ■ 手動將階段和工作新增至管線。
發行應用程式並自動發行。	以各種方式整合和發行應用程式。 ■ 持續將 GitHub 或 GitLab 存放庫中的程式碼整合到管線中。 ■ 整合 Docker 主機以執行持續整合工作，如此部落格文章〈為 vRealize Automation Code Stream 建立 Docker 主機〉中所述。 ■ 使用 YAML 雲端範本自動部署應用程式。 ■ 自動將應用程式部署到 Kubernetes 叢集。 ■ 將應用程式發行至藍綠部署。 ■ 將 Code Stream 與您自己的建置工具、測試工具和部署工具整合。 ■ 使用將 Code Stream 與其他應用程式整合的 REST API。
追蹤趨勢、度量和關鍵效能指標 (KPI)。	建立自訂儀表板，並深入瞭解管線的效能。
解決問題。	當管線執行失敗時，使 Code Stream 建立 Jira 票證。

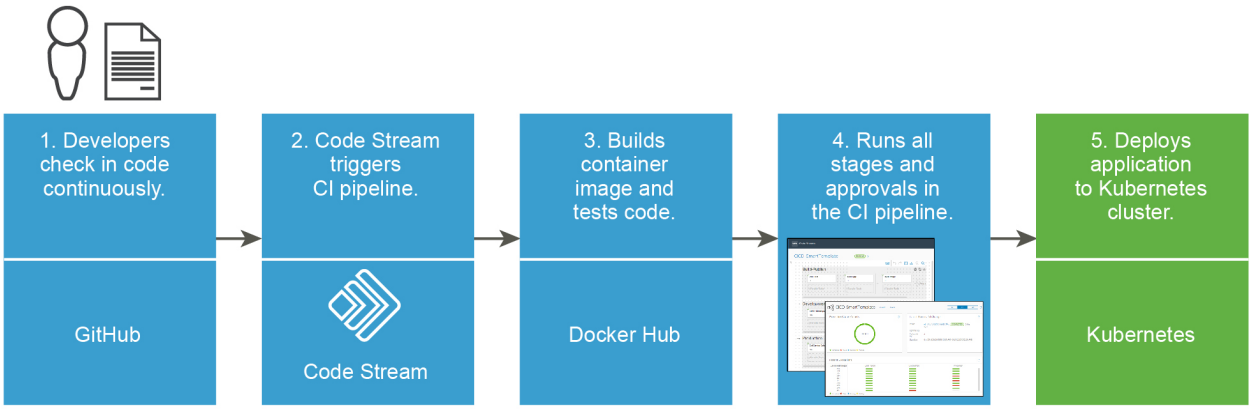
本章節討論下列主題：

- 如何在 Code Stream 中將 GitHub 或 GitLab 存放庫中的程式碼持續整合到管線
- 如何在 Code Stream 中自動發行從 YAML 雲端範本部署的應用程式
- 如何在 Code Stream 中自動將應用程式發行至 Kubernetes 叢集
- 如何在 Code Stream 中將應用程式部署至藍綠部署

- 如何將自己的建置工具、測試工具和部署工具與 Code Stream 整合
- 如何在下一個工作中使用雲端範本工作的資源內容
- 如何使用 REST API 將 Code Stream 與其他應用程式整合
- 如何在 Code Stream 中利用程式碼形式的管線

如何在 Code Stream 中將 GitHub 或 GitLab 存放庫中的程式碼持續整合到管線

身為開發人員，您想要持續整合 GitHub 存放庫或 GitLab Enterprise 存放庫中的代碼。每當開發人員更新其程式碼並認可存放庫變更時， Code Stream 可以接聽這些變更，並觸發管線。



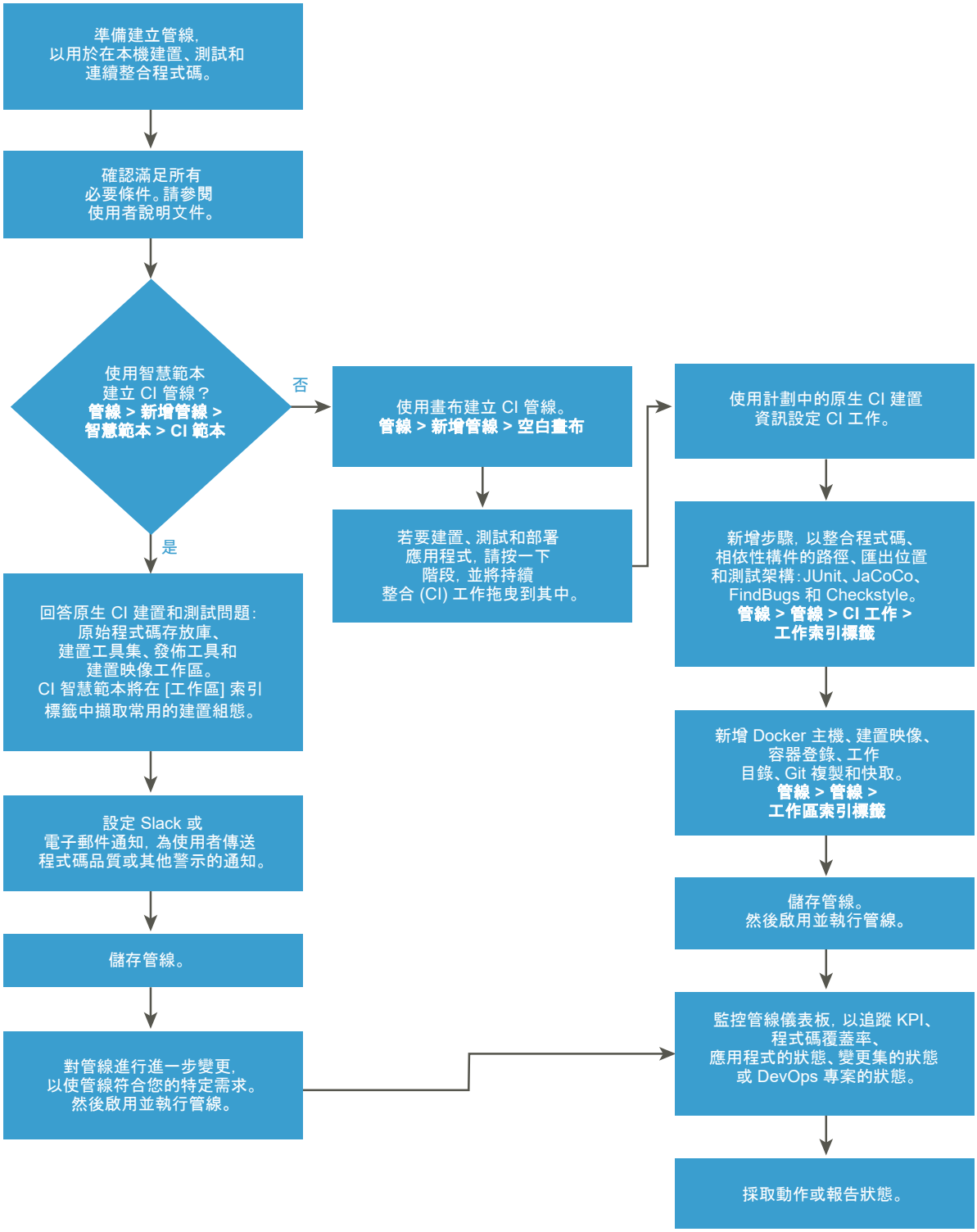
若要讓 Code Stream 在程式碼變更時觸發管線，請使用 Git 觸發器。之後，每次認可程式碼變更時，Code Stream 都會觸發管線。

Code Stream 管線工作區支援使用 Docker 和 Kubernetes 執行持續整合工作和自訂工作。

如需有關設定工作區的詳細資訊，請參閱設定管線工作區。

以下流程圖顯示了使用智慧管線範本建立管線或手動建置管線時可以採用的工作流程。

圖 5-1. 使用智慧管線範本或手動建立管線的工作流程



以下範例使用 Docker 工作區。

若要建置代碼，請使用 Docker 主機。您可以使用 JUnit 和 JaCoCo 作為測試架構工具並將其包括在管線中，這些工具將執行單元測試和代碼涵蓋範圍。

您可以使用持續整合智慧管線範本，以建立能夠建置、測試代碼並部署至專案團隊在 AWS 上的 Kubernetes 叢集的持續整合管線。若要儲存持續整合工作的代碼相依性構件 (這樣可以節省代碼建置時間)，您可以使用快取。

在建置和測試程式碼的管線工作中，可包括多個持續整合步驟。這些步驟會駐留在管線觸發時 Code Stream 複製原始程式碼的相同工作目錄中。

若要將程式碼部署至 Kubernetes 叢集，您可以在管線中使用 Kubernetes 工作。然後，您必須啟用並執行管線。然後，對存放庫中的程式碼進行變更，並觀察管線觸發器。若要在管線執行後監控和報告管線趨勢，可以使用儀表板。

在下列範例中，若要建立持續將程式碼整合到管線中的持續整合管線，可以使用持續整合智慧管線範本。此範例使用 Docker 工作區。

(選擇性) 您可以手動建立管線，並向其新增階段和工作。如需有關計劃產生持續整合與手動建立管線的詳細資訊，請參閱手動新增工作前在 Code Stream 中規劃 CICD 原生建置。

必要條件

- 對持續整合建置進行規劃。請參閱使用智慧管線範本前在 Code Stream 中規劃持續整合原生建置。
- 確認 GitLab 原始程式碼存放庫存在。如需說明，請諮詢您的 Code Stream 管理員。
- 新增 Git 端點。如需範例，請參閱如何在 Code Stream 中使用 Git 觸發器執行管線。
- 若要讓 Code Stream 接聽 GitHub 或 GitLab 存放庫中的變更並在發生變更時觸發管線，請新增 Webhook。如需範例，請參閱如何在 Code Stream 中使用 Git 觸發器執行管線。
- 新增 Docker 主機端點，此端點將為持續整合工作建立可供多個持續整合工作使用的容器。如需有關端點的詳細資訊，請參閱什麼是 Code Stream 中的端點 。
- 取得映像 URL、建置主機和建置映像的 URL。如需說明，請諮詢您的 Code Stream 管理員。
- 確認針對測試架構工具使用 JUnit 和 JaCoCo。
- 為持續整合建置設定外部執行個體：Jenkins、TFS 或 Bamboo。Kubernetes 外掛程式會部署您的程式碼。如需說明，請諮詢您的 Code Stream 管理員。

程序

1 遵循必要條件。

2 若要使用智慧管線範本建立管線，請開啟持續整合智慧管線範本，然後填寫表單。

   a 按一下**管線 > 新增管線 > 智慧範本 > 持續整合**。

   b 在範本中回答有關原始程式碼存放庫、建置工具集、發佈工具和建置映像工作區的問題。

   c 為您的團隊新增 Slack 通知或電子郵件通知。

   d 若要使用智慧管線範本建立管線，請按一下**建立**。

e 若要對管線進行任何進一步變更，請按一下**編輯**，做出變更，然後按一下**儲存**。

f 啟用管線並加以執行。

3 若要手動建立管線，請將階段和工作新增到畫布，並將原生持續整合建置資訊包括在持續整合工作中。

a 按一下**管線 > 新增管線 > 空白畫布**。

b 按一下階段，然後將數個持續整合工作從導覽窗格拖曳到階段。

c 若要設定持續整合工作，請按一下此工作，然後按一下**工作**索引標籤。

d 新增持續整合程式碼的步驟。

e 包含相依性構件的路徑。

f 新增匯出位置。

g 新增您要使用的測試架構工具。

h 新增 Docker 主機和建置映像。

i 新增容器登錄、工作目錄和快取。

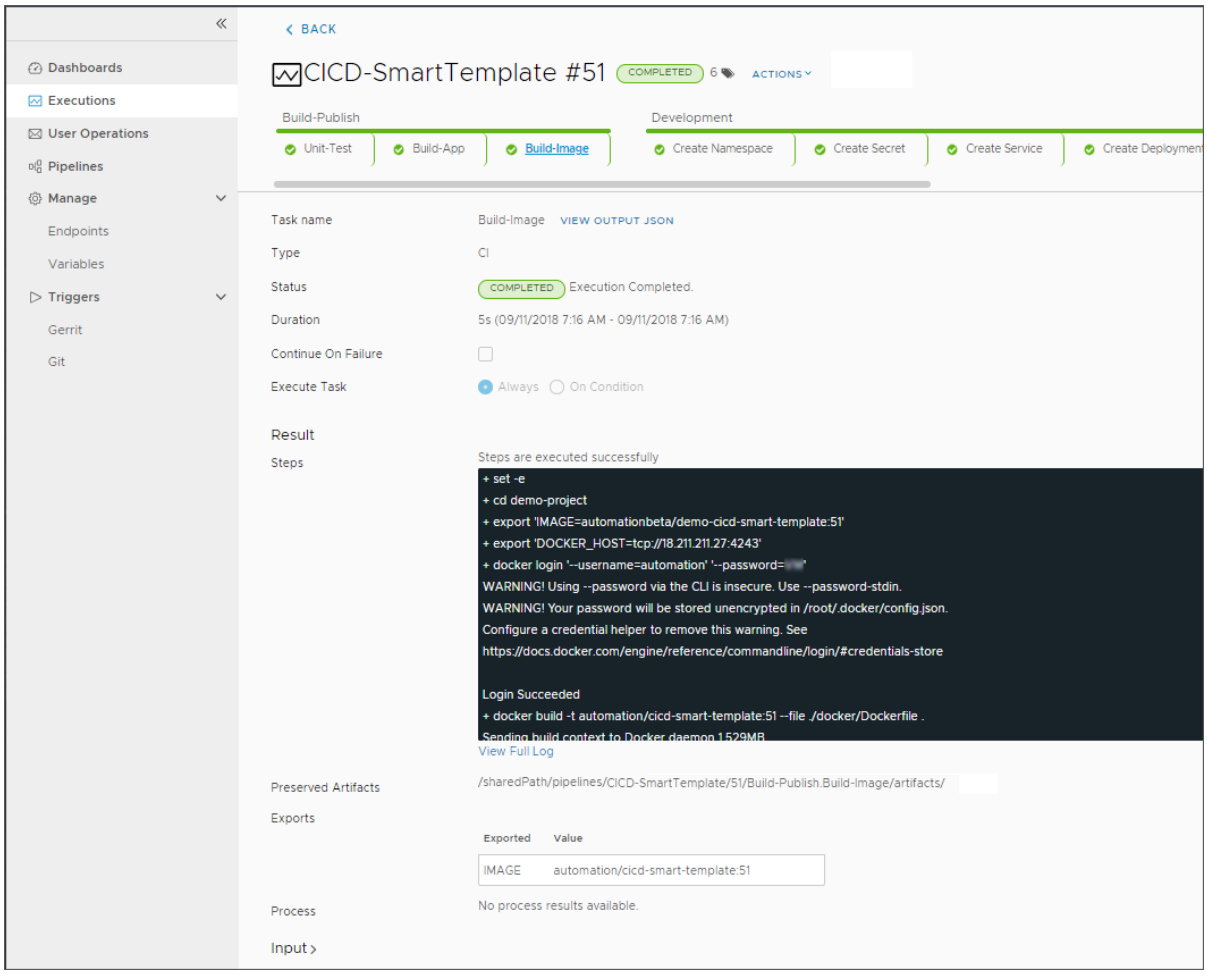
j 儲存管線，然後啟用它。

4 對 GitHub 存放庫或 GitLab 存放庫中的代碼進行變更。

Git 觸發器將啟動管線，此管線開始執行。

5 若要確認程式碼變更是否觸發了管線，請按一下**觸發器 > Git > 活動**。

6 若要檢視管線執行，請按一下**執行**，並確認建立和匯出建置映像的步驟。



7 若要監控管線儀表板，以便您追蹤 KPI 與趨勢，請按一下**儀表板 > 管線儀表板**。

結果

恭喜您！您已建立一個管線，可將 GitHub 存放庫或 GitLab 存放庫中的代碼持續整合至管線，並部署建置映像。

後續步驟

若要瞭解詳細資訊，請參閱供 Code Stream 管理員和開發人員使用的更多資源。

如何在 Code Stream 中自動發行從 YAML 雲端範本部署的應用程式

身為開發人員，您需要在每次認可變更時都從內部部署 GitHub 執行個體擷取自動化雲端範本的管線。您需要使用管線將 WordPress 應用程式部署到 Amazon Web Services (AWS) EC2 或資料中心。 Code Stream 從管線呼叫雲端範本，並自動執行該雲端範本的持續整合與持續交付 (CICD) 以部署應用程式。

若要建立並觸發管線，您將需要一個 VMware 雲端範本。

對於 Code Stream 雲端範本工作中的**雲端範本來源**，您可以選取下列其中一項：

- **Cloud Assembly 範本**，以作為原始檔控制。在此情況下，無需 GitLab 或 GitHub 存放庫。
- **原始檔控制** (如果將 GitLab 或 GitHub 用於原始檔控制)。在此情況下，您必須具有 Git webhook，並透過 webhook 觸發管線。

如果 GitHub 存放庫中有 YAML 雲端範本，並且想要在管線中使用該雲端範本，則以下是您需要執行的動作。

1 在 Cloud Assembly 中，將雲端範本推送至 GitHub 存放庫。

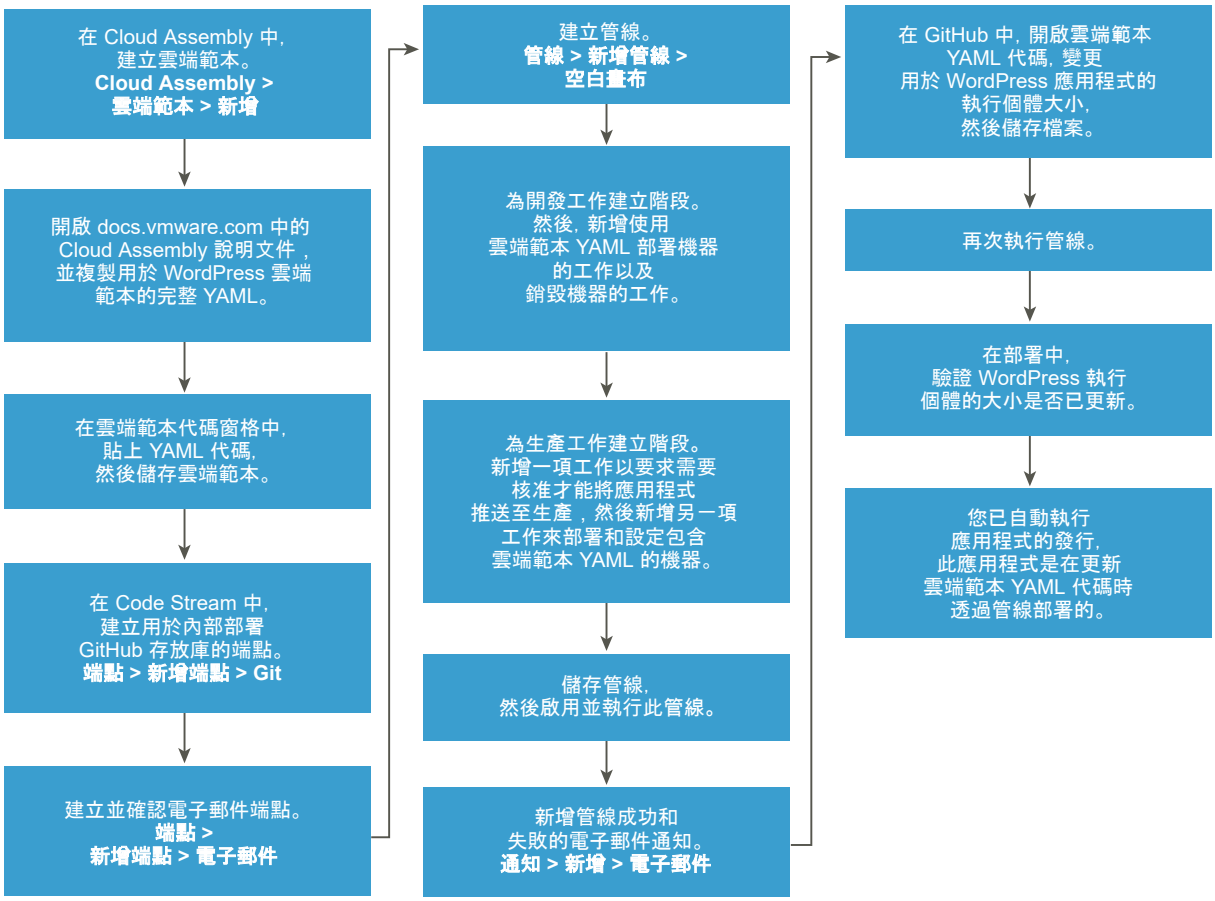
2 在 Code Stream 中，建立 Git 端點。然後，建立使用 Git 端點和管線的 Git webhook。

3 若要觸發管線，請更新 GitHub 存放庫中的任何檔案，並認可變更。

如果 GitHub 存放庫中沒有 YAML 雲端範本，並且想要從原始檔控制使用雲端範本，請使用此程序來瞭解如何操作。此程序說明了如何為 WordPress 應用程式建立雲端範本，以及如何從內部部署 GitHub 存放庫進行觸發。每當您變更 YAML 雲端範本時，管線都會觸發並自動執行應用程式的發行程序。

- 在 Cloud Assembly 中，您將新增雲端帳戶、新增雲端區域，以及建立雲端範本。
- 在 Code Stream 中，您將為主控雲端範本的內部部署 GitHub 存放庫新增端點。然後，將雲端範本新增至管線。

此使用案例範例顯示如何使用內部部署 GitHub 存放庫中的雲端範本。



必要條件

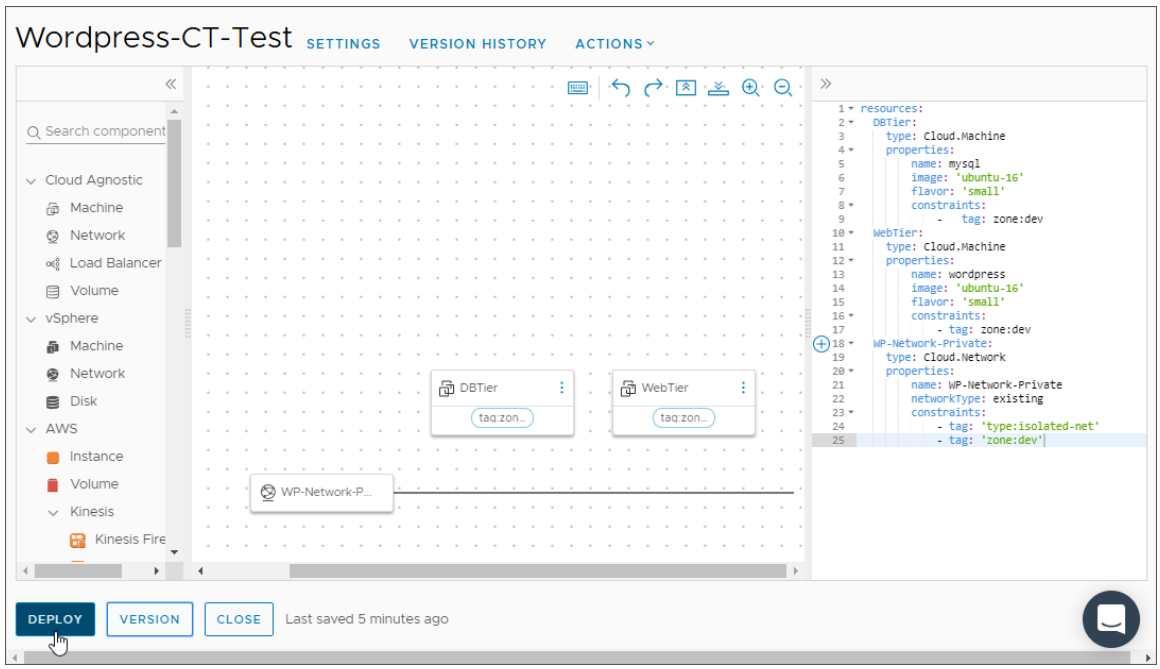
- 在 vRealize Automation Cloud Assembly 基礎結構中新增雲端帳戶和雲端區域。請參閱 vRealize Automation Cloud Assembly 說明文件。
- 若要在下列程序中建立雲端範本，請將 WordPress YAML 代碼複製到剪貼簿。請參閱 vRealize Automation Cloud Assembly 說明文件中 WordPress 使用案例中的雲端範本 YAML 代碼。
- 將 WordPress 應用程式的 YAML 程式碼新增至 GitHub 執行個體。
- 為 Git 觸發器新增 webhook，以便每次 commit 變更時，管線均可提取 YAML 程式碼。在 Code Stream 中，按一下**觸發器 > Git > 適用於 Git 的 Webhook**。
- 若要使用雲端範本工作，您必須擁有任一 Cloud Assembly 角色。

程序

1 在 Cloud Assembly 中，依照下列步驟進行操作。

   a 按一下 **VMware Cloud Templates**，然後為 WordPress 應用程式建立雲端範本和部署。

   b 將複製到剪貼簿的 WordPress YAML 代碼貼至您的雲端範本，然後進行部署。

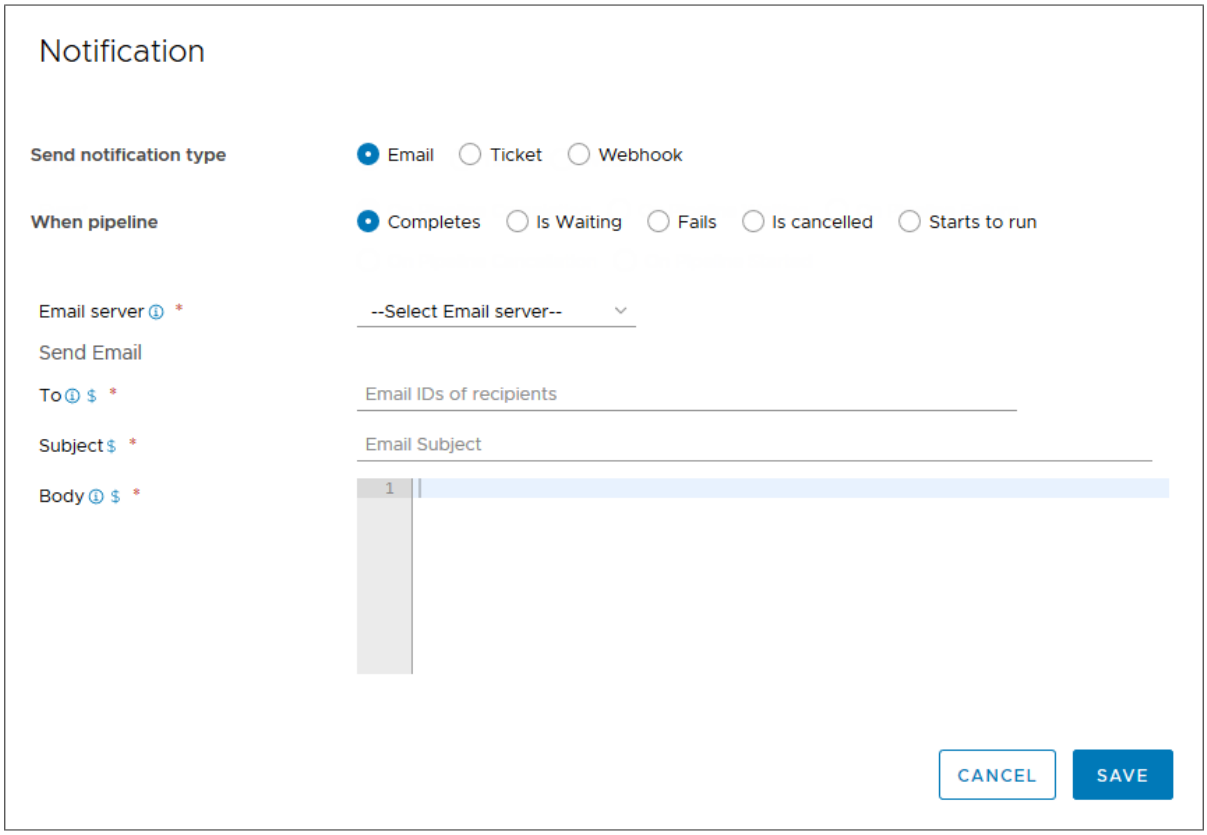


2 在 Code Stream 中，建立端點。

a 針對 YAML 檔案所在的內部部署 GitHub 存放庫建立 Git 端點。

b 新增可在管線執行時向使用者通知管線狀態的電子郵件端點。

3 建立管線，並新增管線成功和失敗的通知。



4 新增開發階段，並新增雲端範本工作。

a 新增用於部署機器的雲端範本工作，並將此工作設定為使用 WordPress 應用程式的雲端範本 YAML。

```
resources:
  DBTier:
    type: Cloud.Machine
    properties:
        name: mysql
        image: 'ubuntu-16'
        flavor: 'small'
        constraints:
            -   tag: zone:dev
  WebTier:
    type: Cloud.Machine
    properties:
        name: wordpress
        image: 'ubuntu-16'
        flavor: 'small'
        constraints:
            - tag: zone:dev
 WP-Network-Private:
    type: Cloud.Network
    properties:
        name: WP-Network-Private
        networkType: existing
        constraints:
            - tag: 'type:isolated-net'
            - tag: 'zone:dev'
```

b 新增用於銷毀機器以釋放資源的雲端範本工作。

5 新增用於生產的階段，並加入核准和部署工作。

a 新增使用者作業工作，以要求需要核准才能將 WordPress 應用程式推送至生產環境。

b 新增用於部署機器的雲端範本工作，並使用 WordPress 應用程式的雲端範本 YAML 對其進行設定。

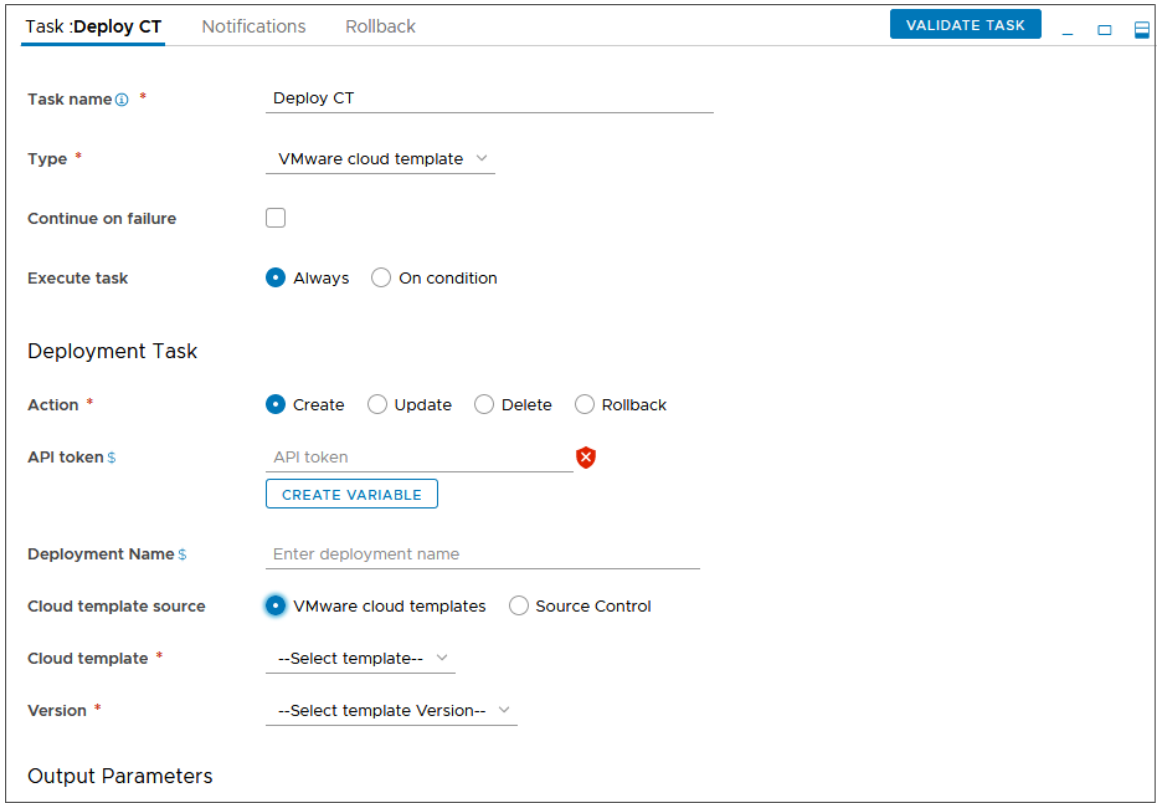
當您選取**建立**時，部署名稱必須是唯一的。如果將名稱保留空白， Code Stream 將為其隨機指派唯一的名稱。

如果在自己的使用案例中選取**復原**，則必須瞭解以下內容：如果選取**復原**動作並輸入**復原版本**，則版本必須採用 `n-x` 格式。例如，`n-1`、`n-2`、`n-3` 等等。如果在 Code Stream 以外的任何位置建立和更新部署，將允許執行復原。

當您登入 Code Stream 時，便會取得使用者 Token，此 Token 的有效期為 30 分鐘。對於執行持續時間較長的管線，如果雲端範本工作之前的工作需要 30 分鐘或更長時間來執行，則使用者 Token 會到期。因此，雲端範本工作會失敗。

若要確保管線的執行時間超過 30 分鐘，您可以輸入選擇性 API Token。當 Code Stream 叫用雲端範本時，API Token 會持續存在，且雲端範本工作會繼續使用 API Token。

當您將 API Token 當作變數使用時，會將其加密。否則，它會以純文字形式使用。



6 執行管線。

若要確認每項工作是否已成功完成，請按一下執行中的工作，然後檢查部署詳細資料中的狀態，以查看詳細的資源資訊。

7 在 GitHub 中，將 WordPress 伺服器執行個體的類型模板從 `small` 修改為 `medium`。

當您 commit 變更時，管線會觸發。它會從 GitHub 存放庫中提取更新的程式碼，並建置應用程式。

```
WebTier:
    type: Cloud.Machine
    properties:
        name: wordpress
        image: 'ubuntu-16'
        flavor: 'medium'
        constraints:
            - tag: zone:dev
```

8 再次執行管線，確認其是否成功，以及是否將 WordPress 執行個體的類型模板從小變更為中。

結果

恭喜您！您已設定自動發行從 YAML 雲端範本部署的應用程式。

後續步驟

若要進一步瞭解如何使用 Code Stream，請參閱第 5 章 使用 Code Stream 的教學課程。

如需其他參考，請參閱供 Code Stream 管理員和開發人員使用的更多資源。

如何在 Code Stream 中自動將應用程式發行至 Kubernetes 叢集

身為 Code Stream 管理員或開發人員，您可以使用 Code Stream 和 VMware Tanzu Kubernetes Grid Integrated Edition (先前稱為 VMware Enterprise PKS) 自動將軟體應用程式部署到 Kubernetes 叢集。此使用案例提及可用於自動發佈應用程式的其他方法。

在此使用案例中，您將建立包含兩個階段的管線，然後使用 Jenkins 建置並部署應用程式。

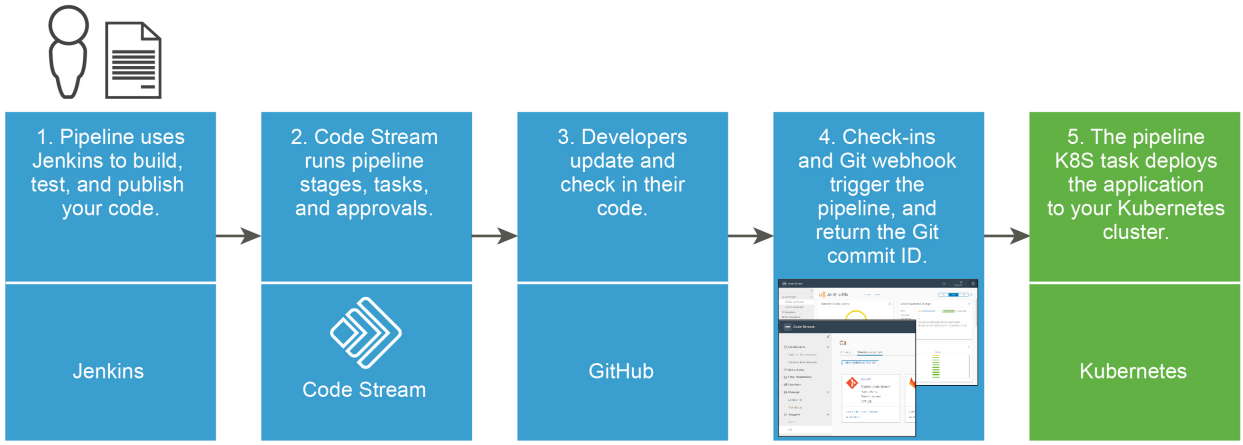
- 第一個階段是用於開發。此階段使用 Jenkins 從 GitHub 存放庫中的分支提取程式碼，然後建置、測試並將其發佈。
- 第二個階段是用於部署。此階段執行使用者操作工作，該工作需要得到關鍵使用者的核准，然後管線才能將應用程式部署至 Kubernetes 叢集。

在管線工作區中使用 Kubernetes API 端點時， Code Stream 會建立必要的 Kubernetes 資源 (如 ConfigMap、密碼和網繭) 以執行持續整合 (CI) 工作或自訂工作。 Code Stream 使用 NodePort 與容器進行通訊。

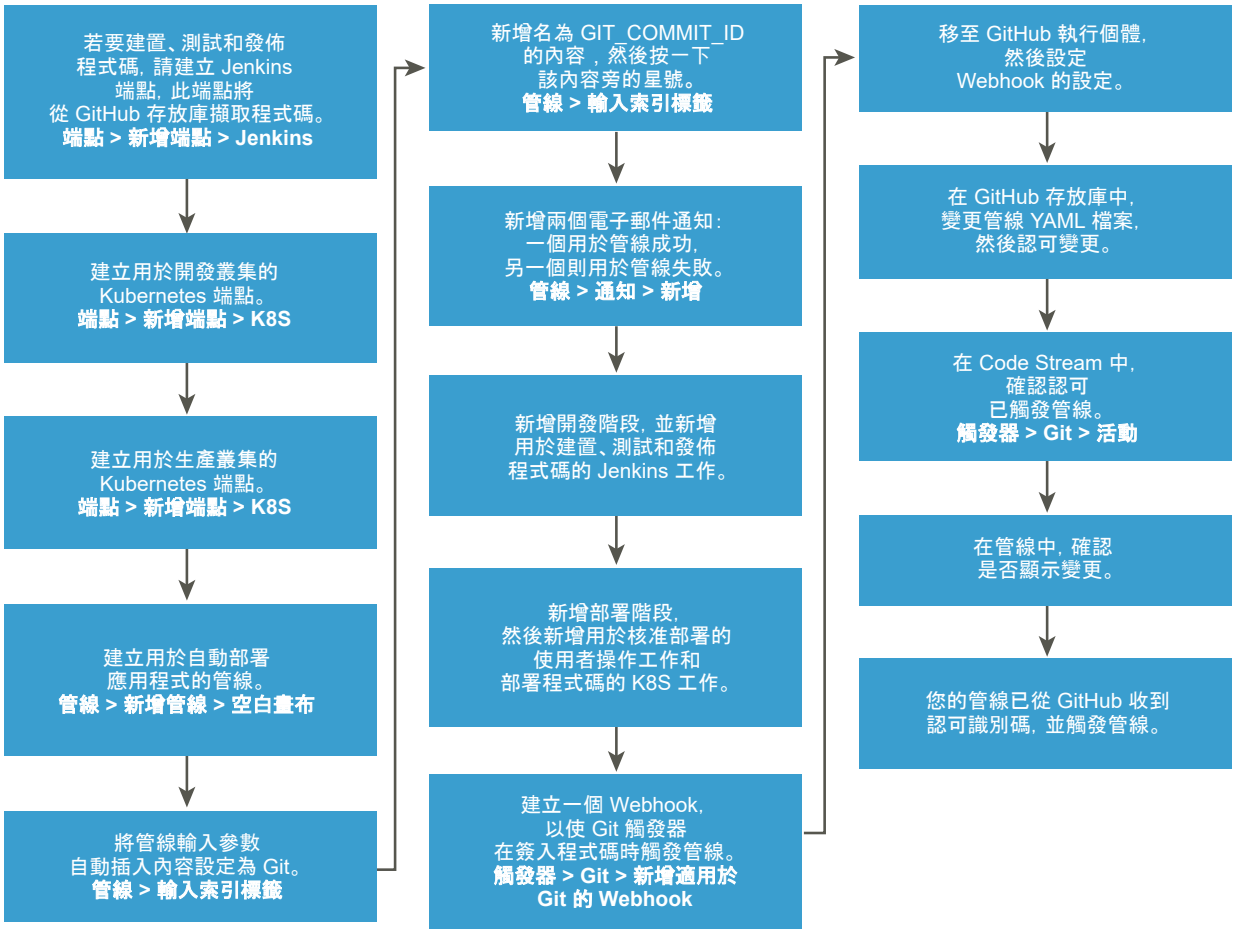
若要在各個管線執行過程中共用資料，您必須提供持續性磁碟區宣告， Code Stream 會將此持續性磁碟區宣告掛接到容器以儲存資料，並將其用於後續管線執行。

Code Stream 管線工作區支援使用 Docker 和 Kubernetes 執行持續整合工作和自訂工作。

如需有關設定工作區的詳細資訊，請參閱設定管線工作區。



開發工具、部署執行個體和管線 YAML 檔案必須可用，以便管線可以建置、測試、發佈及部署應用程式。管線會將應用程式部署到 AWS 上 Kubernetes 叢集的開發和生產執行個體。



自動發佈應用程式的其他方法：

- 您可以使用 Code Stream 原生建置功能和 Docker 建置主機，而不是使用 Jenkins 建置應用程式。
- 您可以將應用程式部署到 Amazon Web Services (AWS) 叢集，而不是將其部署到 Kubernetes 叢集。

如需有關使用 Code Stream 原生建置功能和 Docker 主機的詳細資訊，請參閱：

- 使用智慧管線範本前在 Code Stream 中規劃 CICD 原生建置
- 手動新增工作前在 Code Stream 中規劃 CICD 原生建置

必要條件

- 確認要部署的應用程式代碼位於正在運作的 GitHub 存放庫中。
- 確認您具有有效的 Jenkins 執行個體。
- 確認您具有有效的電子郵件伺服器。
- 在 Code Stream 中，建立用於連線至電子郵件伺服器的電子郵件端點。
- 在 Amazon Web Services (AWS) 上設定兩個 Kubernetes 叢集分別用於開發和生產，您的管線將在其中部署應用程式。
- 確認 GitHub 存放庫包含管線的 YAML 程式碼，或包含用於定義環境之中繼資料和規格的 YAML 檔案。

程序

1 在 Code Stream 中，按一下**端點 > 新增端點**，然後建立您將在管線中使用以從 GitHub 存放庫提取程式碼的 Jenkins 端點。

2 若要建立 Kubernetes 端點，請按一下**新增端點**。

   a 為開發 Kubernetes 叢集建立端點。

   b 為生產 Kubernetes 叢集建立端點。

   Kubernetes 叢集的 URL 可能包含連接埠號碼，也可能不包含連接埠號碼。

   例如：

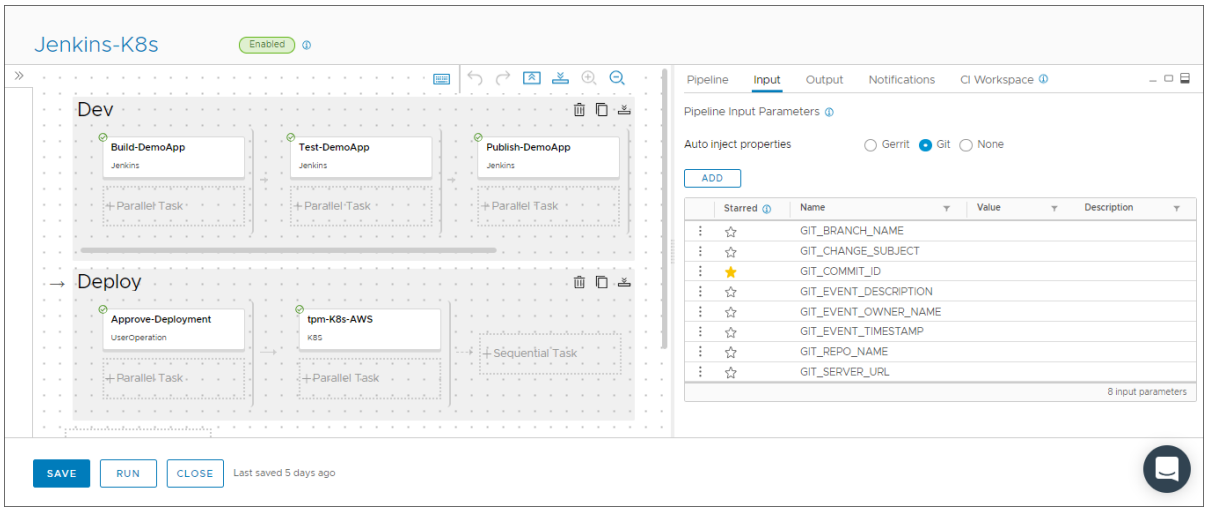
   ```
   https://10.111.222.333:6443
   https://api.kubernetesserver.fa2c1d78-9f00-4e30-8268-4ab81862080d.k8s-user.com
   ```

3 建立一個管線，將應用程式的容器 (例如 Wordpress) 部署到開發 Kubernetes 叢集，並設定管線的輸入內容。

a 若要允許管線辨識觸發管線之 GitHub 中的程式碼認可，請在管線中按一下**輸入**索引標籤，然後選取**自動插入內容**。

b 新增名為 **GIT_COMMIT_ID** 的內容，然後按一下旁邊的星號。

當管線執行時，管線執行會顯示 Git 觸發器傳回的認可識別碼。

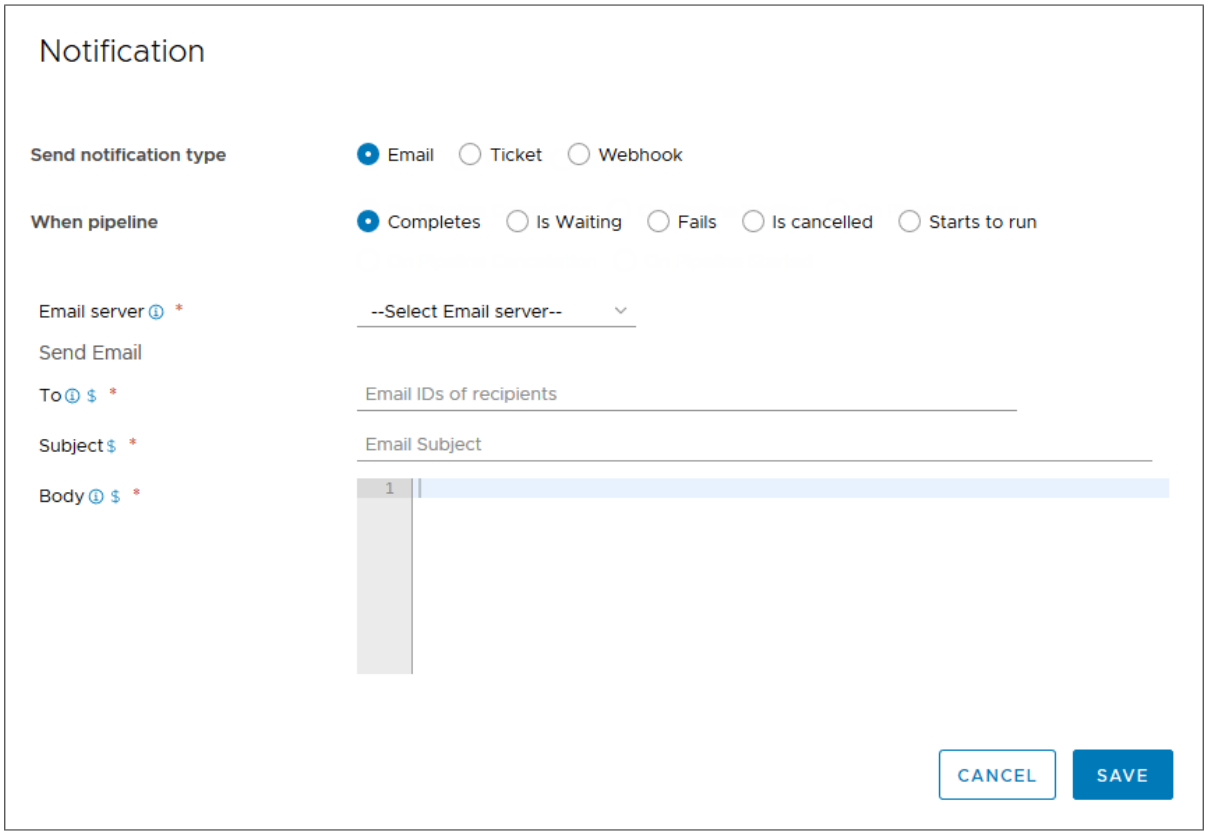


4 新增通知以在管線成功或失敗時傳送電子郵件。

a 在管線中，按一下**通知**索引標籤，然後按一下**新增**。

b 若要在管線執行完成時新增電子郵件通知，請選取**電子郵件**，並選取**完成**。然後，選取電子郵件伺服器，輸入電子郵件地址，並按一下**儲存**。

c 若要針對管線失敗新增另一個電子郵件通知，請選取**失敗**，然後按一下**儲存**。

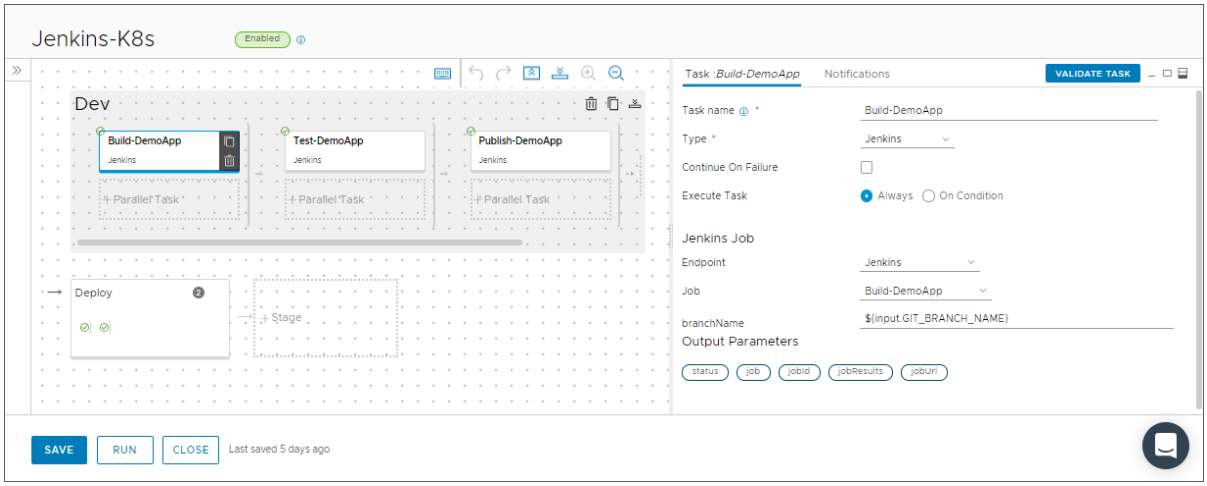


5 將開發階段新增至管線，並新增建置、測試和發佈應用程式的工作。然後，驗證每個工作。

a 若要建置應用程式，請新增使用 Jenkins 端點的 Jenkins 工作，然後從 Jenkins 伺服器執行建置工作。然後，若要使管線提取程式碼，請輸入以下格式的 Git 分支：
`${input.GIT_BRANCH_NAME}`

b 若要測試應用程式，請新增使用相同 Jenkins 端點的 Jenkins 工作，然後從 Jenkins 伺服器執行測試工作。然後，輸入相同的 Git 分支。

c 若要發佈應用程式，請新增使用相同 Jenkins 端點的 Jenkins 工作，然後從 Jenkins 伺服器執行發佈工作。然後，輸入相同的 Git 分支。

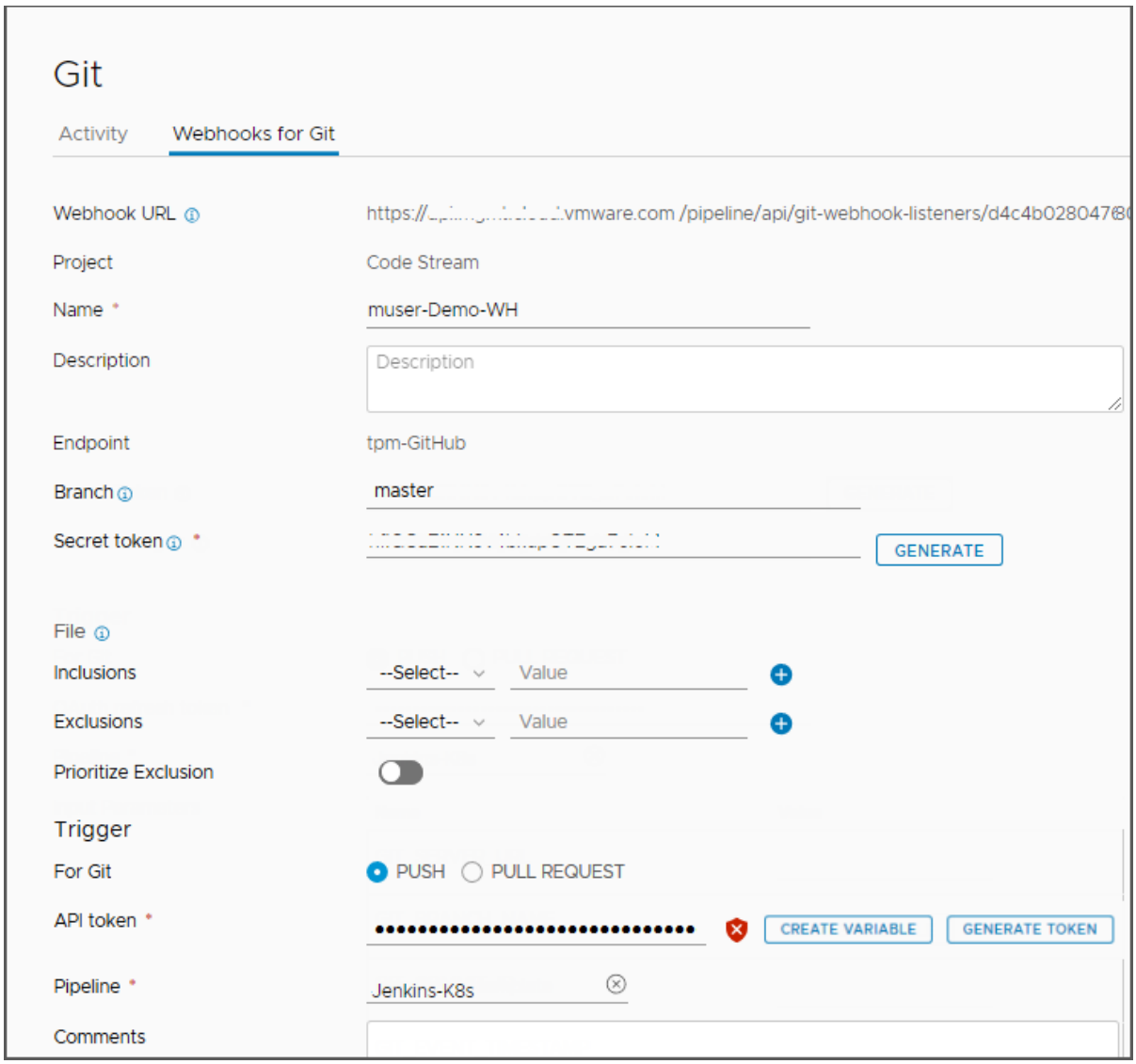


6 將部署階段新增至管線，然後新增需要核准應用程式部署的工作，以及將應用程式部署到 Kubernetes 叢集的另一個工作。然後，驗證每個工作。

a 若要申請核准應用程式部署，請新增使用者操作工作，新增必須予以核准的使用者的電子郵件地址，並輸入訊息。然後，啟用**傳送電子郵件**。

b 若要部署應用程式，請新增 Kubernetes 工作。然後，在 Kubernetes 工作內容中，選取開發 Kubernetes 叢集，選取**建立**動作，然後選取**本機定義**裝載來源。然後，選取本機 YAML 檔案。

7 新增可讓 Code Stream 使用 Git 觸發器的 Git webhook，當開發人員認可其程式碼時，會觸發管線。



8 若要測試管線，請移至 GitHub 存放庫，更新應用程式 YAML 檔案，並認可變更。

a 在 Code Stream 中，確認認可將會顯示。

a 按一下**觸發器 > Git > 活動**。

b 尋找管線的觸發器。

c 按一下**儀表板 > 管線儀表板**。

d 在管線儀表板上的最新成功變更區域中，找到 GIT_COMMIT_ID。

9 檢查您的管線程式碼並確認該變更會出現。

結果

恭喜您！您已自動將軟體應用程式部署至 Kubernetes 叢集。

範例： 將應用程式部署至 Kubernetes 叢集的管線 YAML 範例

對於此範例中使用的管線類型，YAML 類似於下列程式碼：

```
apiVersion: v1
kind: Namespace
metadata:
  name: ${input.GIT_BRANCH_NAME}
  namespace: ${input.GIT_BRANCH_NAME}
---
apiVersion: v1
data:
  .dockercfg:
eyJzeW1waG9ueS10YW5nby1iZXRhMi5qZnJvZy5pbyI6eyJ1c2VybmFtZSI6InRhbmdvLWJldGEyIiwicGFzc3dvcmQiOi
JhRGstcmVOLW1UQi1IejciLCJlbWFpbCI6InRhbmdvLWJldGEyQHZtd2FyZS5jb20iLCJhdXRoIjoiZEdGdVoyOHRZbVYw
WVRJNllVUnJMWEpsVGkxdFZFSXRTSG8zIn19
kind: Secret
metadata:
  name: jfrog
  namespace: ${input.GIT_BRANCH_NAME}
type: kubernetes.io/dockercfg
---
apiVersion: v1
kind: Service
metadata:
  name: codestream
  namespace: ${input.GIT_BRANCH_NAME}
  labels:
    app: codestream
spec:
  ports:
    - port: 80
  selector:
    app: codestream
    tier: frontend
  type: LoadBalancer
---
apiVersion: extensions/v1
kind: Deployment
metadata:
  name: codestream
  namespace: ${input.GIT_BRANCH_NAME}
  labels:
    app: codestream
spec:
  selector:
    matchLabels:
      app: codestream
      tier: frontend
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: codestream
```

```
        tier: frontend
    spec:
      containers:
      - name: codestream
        image: cas.jfrog.io/codestream:${input.GIT_BRANCH_NAME}-${Dev.PublishApp.output.jobId}
        ports:
        - containerPort: 80
          name: codestream
      imagePullSecrets:
      - name: jfrog
```

後續步驟

若要將軟體應用程式部署至生產 Kubernetes 叢集，請再次執行這些步驟，然後選取生產叢集。

若要瞭解有關將 Code Stream 與 Jenkins 整合的詳細資訊，請參閱如何整合 Code Stream 與 Jenkins。

如何在 Code Stream 中將應用程式部署至藍綠部署

藍綠部署是一個部署模型，使用在 Kubernetes 叢集中以相同方式部署和設定的兩個 Docker 主機。當 Code Stream 中的管線部署應用程式時，藍綠部署模型可縮短環境中出現的停機時間。

部署模型中的藍綠部署執行個體都有不同的用途。一次只能有一個執行個體接受部署應用程式的即時流量，並且每個執行個體都在特定時間接受該流量。藍部署執行個體接收應用程式的第一個版本，而綠部署執行個體接收第二個。

藍綠部署環境中的負載平衡器會確定即時流量在部署應用程式時採用哪一個路由。透過使用藍綠部署模型，您的環境仍可正常運作，使用者不會注意到任何停機時間，並且管線會持續將應用程式整合和部署至生產環境。

Code Stream 中建立的管線代表包含兩個階段的藍綠部署模型。一個階段適用於開發，而另一個階段適用於生產。

Code Stream 管線工作區支援使用 Docker 和 Kubernetes 執行持續整合工作和自訂工作。

如需設定工作區的相關資訊，請參閱設定管線工作區。

表 5-2. 藍綠部署的開發階段工作

工作類型	工作
Kubernetes	為藍綠部署建立命名空間。
Kubernetes	為 Docker Hub 建立秘密金鑰。
Kubernetes	建立用來部署應用程式的服務。
Kubernetes	建立藍部署。
輪詢	驗證藍部署。
Kubernetes	移除命名空間。

表 5-3. 藍綠部署的生產階段工作

工作類型	工作
Kubernetes	綠部署會從藍部署取得服務詳細資料。
Kubernetes	取得綠部署複本集的詳細資料。
Kubernetes	建立綠部署，並使用秘密金鑰提取容器映像。
Kubernetes	更新服務。
輪詢	確認生產 URL 上的部署已成功。
Kubernetes	完成藍部署。
Kubernetes	移除藍部署。

若要在您自己的藍綠部署模型中部署應用程式，您可以在 Code Stream 中建立包含兩個階段的管線。第一個階段包括將應用程式部署至藍部署執行個體的藍部署工作，而第二個階段包括將應用程式部署至綠部署執行個體的綠部署工作。

您可以使用 CICD 智慧管線範本來建立您的管線。此範本可以為您建立管線階段和工作，並包含部署選取項目。

如果您手動建立管線，則必須規劃管線階段。如需範例，請參閱手動新增工作前在 Code Stream 中規劃 CICD 原生建置。

在此範例中，您可以使用 CICD 智慧管線範本建立藍綠部署管線。

必要條件

- 確認您可以存取 AWS 上正在運作的 Kubernetes 叢集。
- 確認您已設定藍綠部署環境，並將藍綠部署執行個體設定為完全相同。
- 在 Code Stream 中建立 Kubernetes 端點，將應用程式映像部署到 AWS 上的 Kubernetes 叢集中。
- 自行熟悉使用 CICD 智慧管線範本。請參閱使用智慧管線範本前在 Code Stream 中規劃 CICD 原生建置。

程序

1 按一下**管線 > 新增管線 > 智慧範本 > CI/CD 範本**。

2 輸入 CICD 智慧管線範本的 CI 部分資訊，然後按**下一步**。

如需說明，請參閱使用智慧管線範本前在 Code Stream 中規劃 CICD 原生建置。

3 完成智慧管線範本的 CD 部分

a 選取應用程式部署的環境。例如，**開發**與**生產**。

b 選取管線將用於部署的服務。

c 在 [部署] 區域中，選取用於開發環境和生產環境的叢集端點。

d 針對生產部署模型，選取 **藍綠部署**，然後按一下**建立**。

結果

恭喜您！您可以使用智慧管線範本來建立管線，該管線可將應用程式部署至 AWS 上 Kubernetes 生產叢集中的藍綠部署執行個體。

範例： 針對部分藍綠部署工作的範例 YAML 程式碼

在 Kubernetes 管線工作中出現的用於藍綠部署的 YAML 程式碼可能類似於下列建立命名空間、服務及部署的範例。如果需要從專有存放庫下載映像，YAML 檔案必須包含具有 Docker 組態密碼的區段。請參閱使用智慧管線範本前在 Code Stream 中規劃 CICD 原生建置的 CD 部分。

智慧管線範本建立管線後，您可以視需要針對您自己的部署修改工作。

用於建立範例命名空間的 YAML 程式碼：

```
apiVersion: v1
kind: Namespace
metadata:
  name: codestream-82855
  namespace: codestream-82855
```

用於建立範例服務的 YAML 程式碼：

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app: codestream-demo
  name: codestream-demo
  namespace: bluegreen-799584
spec:
  minReadySeconds: 0
  ports:
  - port: 80
  selector:
    app: codestream-demo
    tier: frontend
  type: LoadBalancer
```

用於建立範例部署的 YAML 程式碼：

```
apiVersion: extensions/v1
kind: Deployment
metadata:
  labels:
    app: codestream-demo
  name: codestream-demo
  namespace: bluegreen-799584
spec:
  minReadySeconds: 0
  replicas: 1
  selector:
    matchLabels:
      app: codestream-demo
      tier: frontend
  template:
    metadata:
      labels:
        app: codestream-demo
        tier: frontend
    spec:
      containers:
      - image: ${input.image}:${input.tag}
        name: codestream-demo
        ports:
        - containerPort: 80
          name: codestream-demo
```

```
      imagePullSecrets:
      - name: jfrog-2
      minReadySeconds: 0
```

後續步驟

若要進一步瞭解如何使用 Code Stream，請參閱第 5 章 使用 Code Stream 的教學課程。

若要復原部署，請參閱如何在 Code Stream 中復原我的部署。

如需其他參考，請參閱供 Code Stream 管理員和開發人員使用的更多資源。

如何將自己的建置工具、測試工具和部署工具與 Code Stream 整合

做為 DevOps 管理員或開發人員，您可以建立自訂指令碼來延伸 Code Stream 的功能。

您可以使用指令碼將 Code Stream 與您自己用於建置、測試和部署應用程式的持續整合 (CI) 和持續交付 (CD) 工具及 API 整合。如果您不公開您的 API 應用程式，則自訂指令碼特別有用。

自訂指令碼幾乎可以執行建置工具、測試工具和部署工具與 Code Stream 整合所需的任何動作。例如，指令碼可與管線工作區搭配使用，以支援建置和測試應用程式的持續整合工作以及部署應用程式的持續交付工作。當管線已完成以及發生其他更多情況時，自訂指令碼可以向 Slack 傳送訊息。

Code Stream 管線工作區支援使用 Docker 和 Kubernetes 執行持續整合工作和自訂工作。

如需有關設定工作區的詳細資訊，請參閱設定管線工作區。

您可以使用其中一種支援的語言撰寫自訂指令碼。在指令碼中，您可以包含業務邏輯，並定義輸入與輸出。輸出類型可包含數字、字串、文字和密碼。您可以建立具有不同業務邏輯、輸入和輸出的多個版本的自訂指令碼。

您可以讓管線在自訂工作中執行一個版本的指令碼。所建立的指令碼位於 Code Stream 執行個體中。

當管線使用自訂整合時，如果您嘗試刪除自訂整合，則會顯示一則錯誤訊息，指出您無法將其刪除。

刪除自訂整合會移除所有版本的自訂指令碼。如果您的現有管線具有使用任何版本指令碼的自訂工作，該管線將會失敗。若要確保現有管線不會失敗，您可以取代並撤回您不再需要使用的指令碼版本。如果沒有管線使用該版本，您可以將其刪除。

表 5-4. 撰寫自訂指令碼後執行的操作

您執行的操作...	有關該動作的詳細資訊...
將自訂工作新增到管線。	自訂工作： ■ 在與管線中其他 CI 工作相同的容器中執行。 ■ 包含輸入變數和輸出變數，指令碼會在管線執行自訂工作之前填入這些變數。 ■ 支援您在指令碼中定義為輸入和輸出的多種資料類型和各種中繼資料類型。
在自訂工作中選取指令碼。	在指令碼中宣告輸入內容和輸出內容。

表 5-4. 撰寫自訂指令碼後執行的操作 (續)

您執行的操作...	有關該動作的詳細資訊...
儲存管線，然後啟用並執行此管線。	管線執行時，自訂工作會呼叫指定的指令碼版本，並執行其中的業務邏輯，這可將建置、測試和部署工具與 Code Stream 整合。
管線執行後，查看執行情況。	確認管線已交付預期結果。

使用呼叫自訂整合版本的自訂工作時，可以將自訂環境變數作為名稱值配對包含在管線**工作區**索引標籤上。當產生器映像建立用於執行 CI 工作並部署映像的工作區容器時， Code Stream 會將環境變數傳遞至該容器。

例如，當 Code Stream 執行個體需要 Web Proxy 並且您使用 Docker 主機建立用於自訂整合的容器時，Code Stream 會執行管線並將 Web Proxy 設定變數傳遞至該容器。

表 5-5. 環境變數名稱-值配對範例

名稱	值
HTTPS_PROXY	http://10.0.0.255:1234
https_proxy	http://10.0.0.255:1234
NO_PROXY	10.0.0.32, *.dept.vsphere.local
no_proxy	10.0.0.32, *.dept.vsphere.local
HTTP_PROXY	http://10.0.0.254:1234
http_proxy	http://10.0.0.254:1234
PATH	/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin

名稱-值配對將按如下所示顯示在使用者介面中：

此範例將建立自訂整合，該整合會將 Code Stream 連線至 Slack 執行個體，並向 Slack 通道發佈訊息。

必要條件

- 若要撰寫自訂指令碼，請確認您熟悉下列其中一種語言：Python 2、Python 3、Node.js，或下列任何 Shell 語言：Bash、sh 或 zsh。
- 使用已安裝的 Node.js 或 Python 執行階段產生容器映像。

程序

1 建立自訂整合。

a 按一下**自訂整合 > 新增**，然後輸入相關的名稱。

b 選取偏好的執行期間環境。

c 按一下**建立**。

指令碼會開啟並顯示程式碼，其中包含所需的執行期間環境。例如，`runtime: "nodejs"`。指令碼必須包含產生器映像使用的執行期間，以便新增到管線中的自訂工作在管線執行時會成功。否則，自訂工作會失敗。

自訂整合 YAML 的主要區域包括執行階段、代碼、輸入內容和輸出內容。此程序說明了各種類型和語法。

自訂整合 YAML 金鑰	說明
runtime	Code Stream 執行代碼的工作執行階段環境，可以是以下不區分大小寫的字串之一： ■ nodejs ■ python2 ■ python3 ■ shell 如果未提供任何內容，則預設值為 shell。
code	做為自訂工作的一部分執行的自訂業務邏輯。
inputProperties	做為自訂工作組態的一部分進行擷取的輸入內容陣列。這些內容通常會在代碼中使用。
outputProperties	可以從自訂工作匯出以散佈到管線的輸出內容陣列。

2 使用可用的資料類型和中繼資料，在指令碼中宣告輸入內容。

輸入內容會在 YAML 的 `code:` 區段中做為內容傳遞到指令碼。

自訂工作 YAML 輸入金鑰	說明	必要
type	要顯示的輸入類型： ■ **text** ■ **textarea** ■ **number** ■ **checkbox** ■ **password** ■ **select**	是
name	自訂工作的輸入的名稱或字串，將插入自訂整合 YAML 程式碼中。為自訂整合定義的每個輸入內容都必須是唯一的。	是
title	管線模型畫布上自訂工作的輸入內容的文字字串標籤。如果保留空白，則預設會使用 **name**。	否
required	決定使用者在設定自訂工作時是否必須輸入該輸入內容。設定為 true 或 false。設定為 true 時，如果使用者在管線畫布上設定自訂工作時未提供值，則工作狀態會保持為未設定。	否

自訂工作 YAML 輸入金鑰	說明	必要
`placeHolder`	輸入內容項目區域的預設文字 (如果不存在值)。對應至 html 預留位置屬性。僅支援特定的輸入內容類型。	否
`defaultValue`	自訂工作顯示在管線模型頁面上時，用於填入輸入內容項目區域的預設值。	否
`bindable`	確定建立管線畫布上自訂工作的模型時，輸入內容是否接受貨幣符號變數。在標題旁邊新增 **$** 指示符號。僅支援特定的輸入內容類型。	否
`labelMessage`	用作使用者的說明工具提示的字串。在輸入標題旁邊新增工具提示圖示 **i**。	否
`enum`	接受顯示選取輸入內容選項的值陣列。僅支援特定的輸入內容類型。 當使用者選取選項並將其儲存用於自訂工作時，**inputProperty** 的值會與此值對應，並顯示在自訂工作建模中。 例如，值 2015。 ■ 2015 ■ 2016 ■ 2017 ■ 2018 ■ 2019 ■ 2020	否
`options`	透過使用 **optionKey** 和 **optionValue** 來接受物件陣列。 ■ **optionKey**。散佈到工作的代碼區段的值。 ■ **optionValue**。在使用者介面中顯示選項的字串。 僅支援特定的輸入內容類型。 選項： **optionKey**：key1。選取並儲存用於自訂工作後，此 inputProperty 的值將與代碼區段中的 **key1** 對應。 **optionValue**：'Label for 1'。使用者介面中 **key1** 的顯示值，不會顯示在自訂工作的任何其他位置。 **optionKey**：key2 **optionValue**：'Label for 2' **optionKey**：key3 **optionValue**：'Label for 3'	否
`minimum`	接受做為此輸入內容的有效最小值的數字。僅支援數字類型的輸入內容。	否
`maximum`	接受做為此輸入內容的有效最大值的數字。僅支援數字類型的輸入內容。	否

表 5-6. 自訂指令碼支援的資料類型和中繼資料

支援的資料類型	支援的輸入中繼資料
■ 字串 ■ 文字 ■ 清單：顯示為任何類型的清單 ■ 對應：顯示為 map[string]any ■ 密碼：顯示為密碼文字方塊，在儲存自訂工作時進行加密 ■ 數字 ■ 布林值：顯示為文字方塊 ■ URL：與字串相同，需要進行其他驗證 ■ 複選、選項按鈕	■ type：字串或文字... ■ Default：預設值 ■ options：清單或選項對應，要與「複選」或「選項按鈕」搭配使用 ■ min：最小值或最小大小 ■ max：最大值或最大大小 ■ title：文字方塊的詳細名稱 ■ placeHolder：使用者介面預留位置 ■ description：將成為工具提示

例如：

```
inputProperties:
        - name: message
          type: text
          title: Message
          placeHolder: Message for Slack Channel
          defaultValue: Hello Slack
          bindable: true
          labelInfo: true
          labelMessage: This message is posted to the Slack channel link provided in the
code
```

3 在指令碼中宣告輸出內容。

指令碼會從指令碼的業務邏輯 `code:` 區段擷取輸出內容，您可以在該區段宣告輸出的內容。

當管線執行時，您可以輸入工作輸出的回應代碼。例如，200。

Code Stream 為每個 **outputProperty** 提供支援的金鑰。

金鑰	說明
類型	目前包含 **label** 的單一值。
name	自訂整合 YAML 的代碼區塊所發出的金鑰。
title	使用者介面中顯示 **outputProperty** 的標籤。

例如：

```
outputProperties:
  - name: statusCode
    type: label
    title: Status Code
```

4 若要與自訂指令碼的輸入和輸出互動，請使用 **`context`** 取得輸入內容或設定輸出內容。

針對輸入內容：`(context.getInput("key"))`

針對輸出內容：`(context.setOutput("key", "value"))`

針對 Node.js：

```
var context = require("./context.js")
var message = context.getInput("message");
//Your Business logic
context.setOutput("statusCode", 200);
```

針對 Python：

```
from context import getInput, setOutput
message = getInput('message')
//Your Business logic
setOutput('statusCode', '200')
```

針對 Shell：

```
# Input, Output properties are environment variables
echo ${message} # Prints the input message
//Your Business logic
export statusCode=200 # Sets output property statusCode
```

5 在 `code:` 區段中，宣告自訂整合的所有業務邏輯。

例如，在 Node.js 執行期間環境中：

```
code: |
    var https = require('https');
    var context = require("./context.js")

    //Get the entered message from task config page and assign it to message var
    var message = context.getInput("message");
    var slackPayload = JSON.stringify(
        {
            text: message
        });

    const options = {
        hostname: 'hooks.slack.com',
        port: 443,
        path: '/YOUR_SLACK_WEBHOOK_PATH',
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
            'Content-Length': Buffer.byteLength(slackPayload)
        }
    };

    // Makes a https request and sets the output with statusCode which
    // will be displayed in task result page after execution
    const req = https.request(options, (res) => {
        context.setOutput("statusCode", res.statusCode);
    });

    req.on('error', (e) => {
```

```
        console.error(e);
    });
    req.write(slackPayload);
    req.end();
```

6 對自訂整合指令碼進行版本控制並發行之前，請下載適用於 Python 或 Node.js 的內容檔案，並測試指令碼中包含的業務邏輯。

   a 將指標放在指令碼中，然後按一下畫布頂端的內容檔案按鈕。例如，如果指令碼類型為 Python，則按一下 **CONTEXT.PY**。

   b 修改檔案並儲存。

   c 在開發系統上，借助內容檔案執行並測試自訂指令碼。

7 將某個版本套用至自訂整合指令碼。

   a 按一下**版本**。

   b 輸入版本資訊。

   c 按一下**發行版本**，以在自訂工作中選取指令碼。

   d 若要建立版本，請按一下**建立**。

8 若要儲存指令碼，請按一下**儲存**。

9 在管線中，設定工作區。

   此範例使用 Docker 工作區。

a 按一下**工作區**索引標籤。

b 選取 Docker 主機和產生器映像 URL。

10 向管線新增自訂工作並設定該工作。

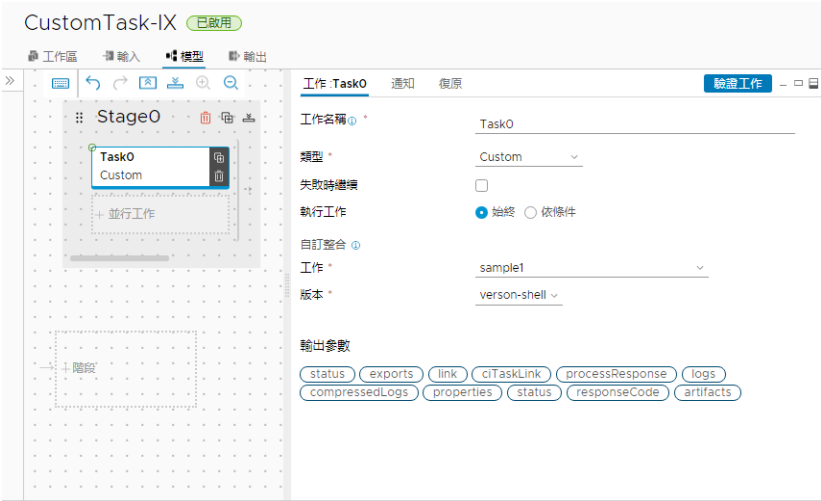
a 按一下**模型**索引標籤。

b 新增工作，為類型選取**自訂**，然後輸入相關的名稱。

c 選取您的自訂整合指令碼和版本。

d 若要在 Slack 中顯示自訂訊息，請輸入訊息文字。

您輸入的任何文字會覆寫自訂整合指令碼中的 `defaultValue`。例如：



11 儲存並啟用管線。

a 按一下**儲存**。

b 在 [管線] 索引標籤中，按一下**啟用管線**以使圓形移到右側。

12 執行管線。

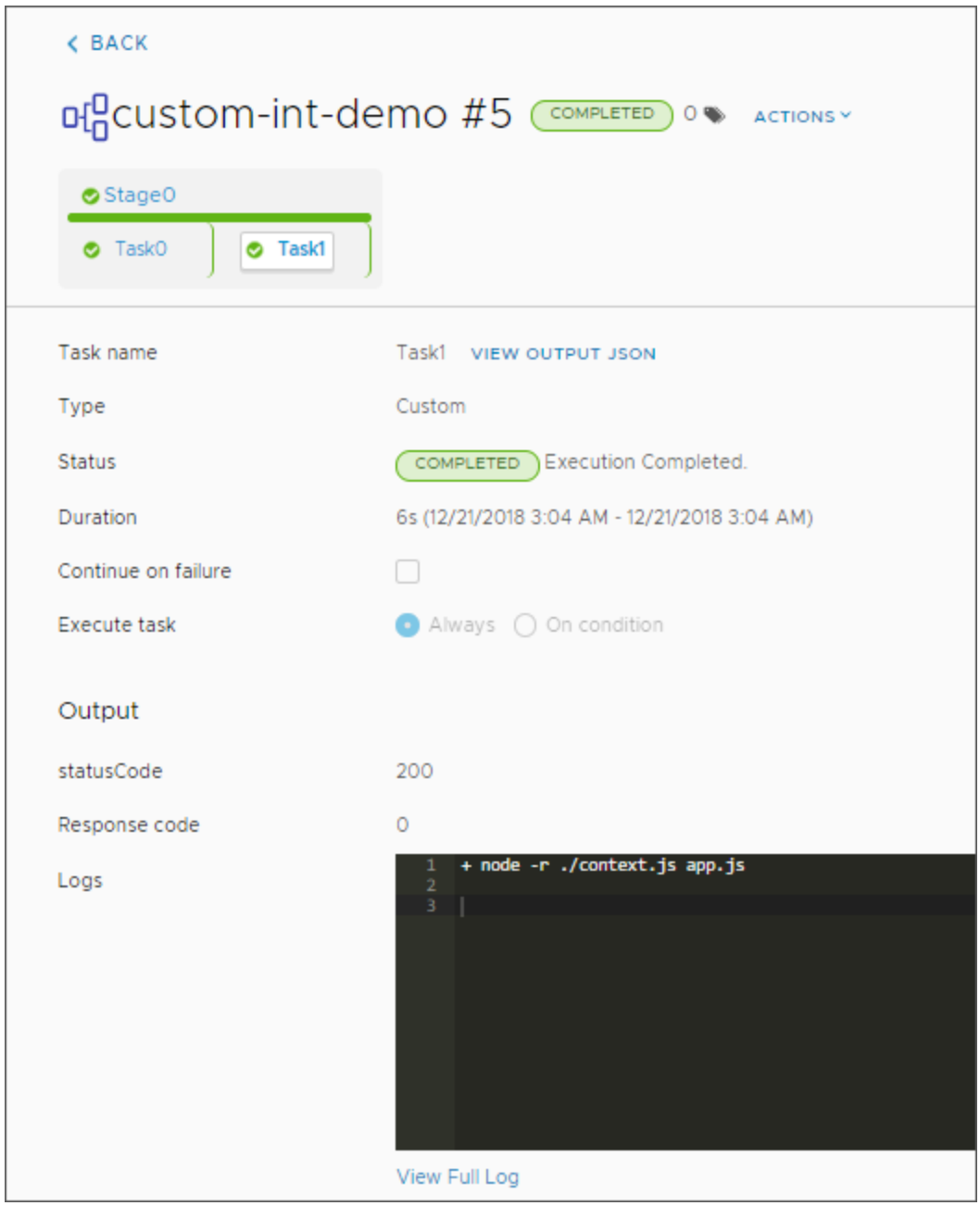
a 按一下**執行**。

b 查看管線執行情況。

c 確認輸出包含預期的狀態碼、回應代碼、狀態以及已宣告的輸出。

您已將 **statusCode** 定義為輸出內容。例如，**statusCode** 為 200 可能表示 Slack 發佈成功，而 **responseCode** 為 0 可能表示指令碼已成功且未發生錯誤。

d 若要確認執行記錄中的輸出，請按一下**執行**及管線連結，然後按一下此工作查看記錄的資料。例如：



13 如果發生錯誤，請疑難排解問題，然後重新執行管線。

例如，如果您的基礎映像中遺失檔案或模組，您必須建立包含遺失檔案的另一個基礎映像。然後，提供 Docker 檔案，並透過管線推送映像。

結果

恭喜您！您已建立自訂整合指令碼，該指令碼將 Code Stream 連線至 Slack 執行個體，並向 Slack 通道發佈訊息。

後續步驟

繼續建立自訂整合，以支援在管線中使用自訂工作，以便延伸 Code Stream 在自動執行軟體發佈生命週期中的功能。

如何在下一個工作中使用雲端範本工作的資源內容

當您在 Code Stream 中使用雲端範本工作時，一般問題是如何在管線的後續工作中使用該工作的輸出。若要使用雲端範本工作的輸出 (例如雲端機器)，必須瞭解如何在雲端範本工作的部署詳細資料中找到資源內容以及雲端機器的 IP 位址。

例如，VMware 雲端範本的部署詳細資料包括雲端機器資源及其 IP 位址。在您的管線中，可以使用雲端機器和 IP 位址作為變數，將雲端範本工作繫結至 REST 工作。

用於尋找雲端機器 IP 位址的方法並不常見，因為必須先完成 VMware 雲端範本的部署，然後才會提供部署詳細資料。接著，您可以使用 VMware 雲端範本部署中的資源來繫結管線工作。

- 在管線的雲端範本工作中顯示的資源內容會在 Cloud Assembly 的 VMware 雲端範本中進行定義。
- 您可能不知道該雲端範本的部署何時完成。
- Code Stream 中的雲端範本工作只能在部署完成後顯示 VMware 雲端範本的輸出內容。

如果您要部署應用程式並叫用各種 API，則此範例非常有用。例如，如果您使用的雲端範本工作呼叫 VMware 雲端範本，且該範本透過 REST API 部署 Wordpress 應用程式，則可以在部署詳細資料中找到已部署機器的 IP 位址，並使用 API 進行測試。

雲端範本工作可透過顯示預先輸入自動填入詳細資料，支援您使用變數繫結。這取決於繫結變數的方式。

此範例顯示了如何執行以下操作：

- 在已成功執行的管線中找到雲端範本工作的部署詳細資料和資源內容。
- 在部署詳細資料的 [資源] 區段中找到雲端機器 IP 位址。
- 在管線中的雲端範本工作之後新增 REST 工作。
- 透過在 REST 工作的 URL 中使用雲端機器 IP 位址，將雲端範本工作繫結至 REST 工作。
- 執行管線，並監視從雲端範本工作到 REST 工作的繫結工作。

必要條件

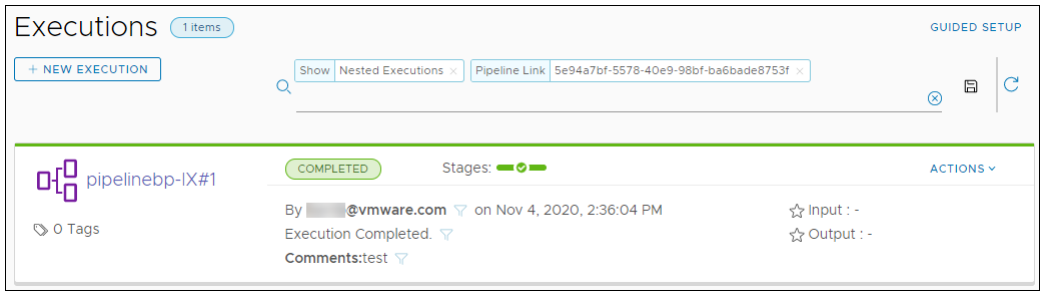
- 確認您的工作 VMware 雲端範本已進行版本設定。
- 確認已在 Cloud Assembly 中成功部署 VMware 雲端範本。
- 確認您的管線包括使用該 VMware 雲端範本的雲端範本工作。
- 確認您的管線已成功執行。

程序

1 在您的管線中，於雲端範本工作部署詳細資料的 [資源] 區段中找到雲端機器的 IP 位址。

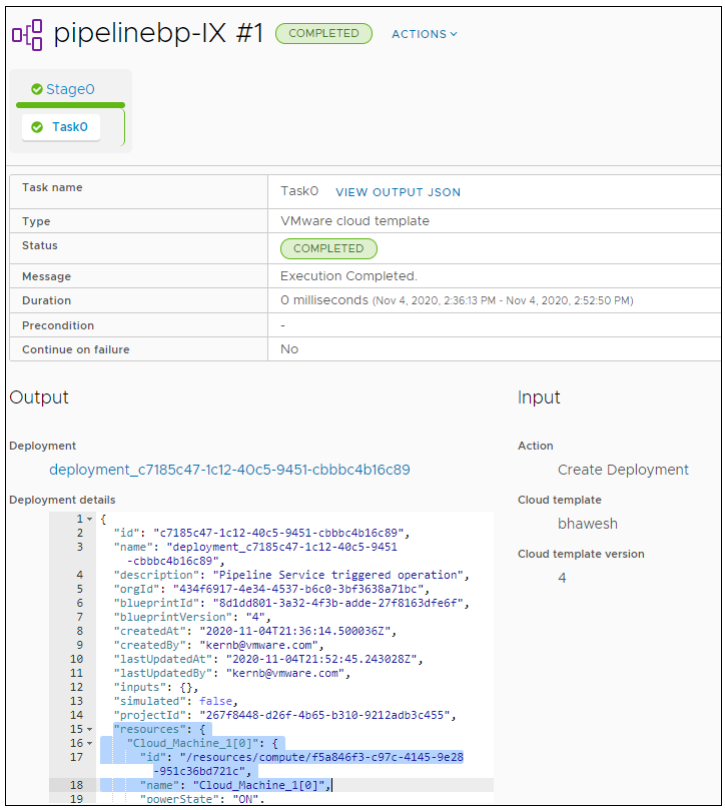
a 按一下**動作 > 檢視執行**。

b 在已成功執行的管線中，按一下管線執行的連結。



c 在管線名稱下，按一下**工作**的連結。

d 在 [輸出] 區域中，找到部署詳細資料。

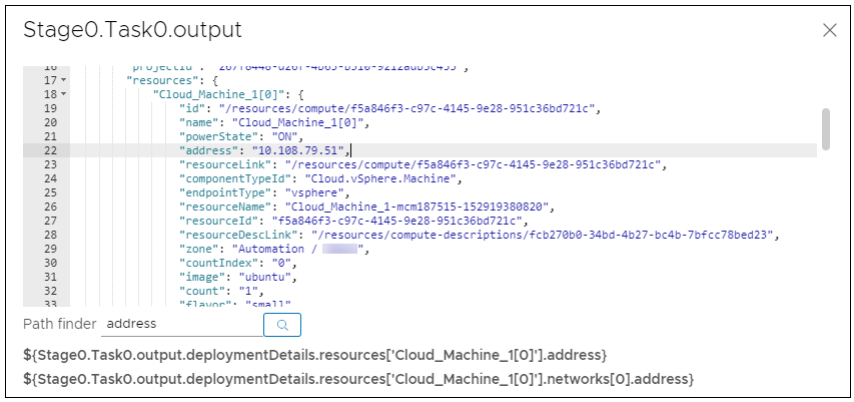


e 在部署詳細資料的 [資源] 區段中，找到雲端機器名稱。

將在 REST 工作的 URL 中包含雲端機器名稱的語法。

f 若要尋找雲端範本工作之輸出內容的繫結運算式，請按一下**檢視輸出 JSON**，搜尋位址內容，然後找到雲端機器的 IP 位址。

繫結運算式隨即顯示在 JSON 輸出的內容和搜尋圖示下方。



位址資源內容會顯示雲端機器的 IP 位址。例如：

```
"resources": {
        "Cloud_Machine_1[0]": {
            "name": "Cloud_Machine_1[0]",
            "powerState": "ON",
            "address": "10.108.79.51",
            "resourceName": "Cloud_Machine_1-mcm187515-152919380820"
```

2 返回管線模型，然後在 REST 工作中輸入 URL。

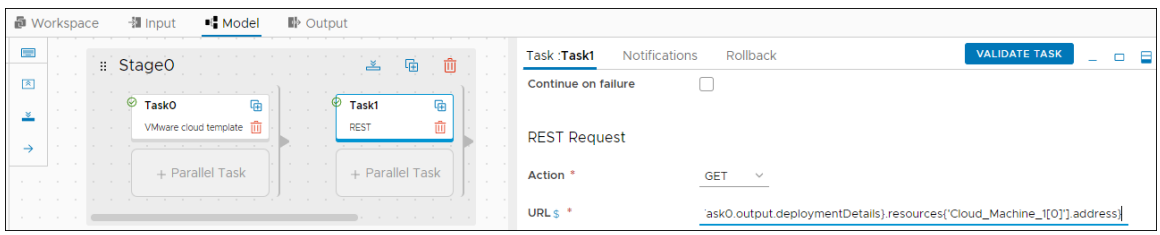
a 按一下**動作 > 檢視管線**。

b 按一下 REST 工作。

c 在 REST 請求的 URL 區域中，輸入 **$**，選取 **Stage**、**Task**、**output**、**deploymentDetails**，然後輸入 **resources**。

可以使用透過自動填入進行預先輸入的功能，直到必須輸入 **resources**。

d 輸入部署詳細資料中雲端機器資源的其餘部分：**{'Cloud_Machine_1[0]'].address}**



對於雲端機器項目，必須使用方括弧標記法，如下所示。

完整的 URL 格式為：$

{Stage0.Task0.output.deploymentDetails.resources{'Cloud_Machine_1[0]'].address}

3 執行管線並監視 REST 工作使用雲端範本工作輸出中的雲端機器和 IP 位址作為要測試的 URL。

結果

恭喜您！您在部署詳細資料和雲端範本工作的 JSON 輸出中找到了雲端機器名稱和 IP 位址，並使用它們將您的雲端範本工作輸出繫結到管線中的 REST 工作 URL 輸入。

後續步驟

繼續瞭解如何將雲端範本工作的資源中的繫結變數用於管線中的其他工作。

如何使用 REST API 將 Code Stream 與其他應用程式整合

Code Stream 提供一個 REST 外掛程式，透過該外掛程式可以將 Code Stream 與使用 REST API 的其他應用程式整合，以便持續開發並傳遞必須彼此互動的軟體應用程式。此 REST 外掛程式會叫用 API，該 API 會在 Code Stream 與其他應用程式之間傳送和接收資訊。

透過此 REST 外掛程式，您可以：

- 將以外部 REST API 為基礎的系統整合到 Code Stream 管線中。
- 將 Code Stream 管線整合為外部系統流程的一部分。

此 REST 外掛程式可與任何 REST API 搭配運作，且支援使用 GET、POST、PUT、PATCH 和 DELETE 方法在 Code Stream 與其他應用程式之間傳送或接收資訊。

表 5-7. 準備管線以透過 REST API 進行通訊

您執行的操作	產生的結果
將 REST 工作新增到管線。	此 REST 工作在應用程式之間傳遞資訊，並且可提供管線階段中連續工作的狀態資訊。
在 REST 工作中，選取 REST 動作並包含 URL。	當管線執行時，管線工作會呼叫該 URL。 對於 POST、PUT 和 PATCH 動作，必須包含裝載。在裝載中，您可以在管線執行時將管線與工作內容繫結。
以此範例為例。	REST 外掛程式的使用範例： 您可以新增 REST 工作以在建置的 Git 認可上建立標籤，並讓該工作發佈請求以從存放庫取得簽入識別碼。此工作可傳送裝載至存放庫，並為建置建立標籤，而存放庫可以透過該標籤傳回回應。

與使用 REST 外掛程式叫用 API 類似，您可以在管線中包含 Poll 工作以叫用 REST API 並進行輪詢，直到其完成且管線工作滿足結束條件為止。

您也可以使用 REST API 匯入和匯出管線，以及使用範例指令碼執行管線。

下列程序會取得簡單 URL。

程序

1 若要建立管線，請按一下**管線 > 新增管線 > 空白畫布**。

2 在您的管線階段中，按一下 **+ 連續工作**。

3 在工作窗格中，新增 REST 工作：

a 輸入工作名稱。

b 在 [類型] 下拉式功能表中，選取 **REST**。

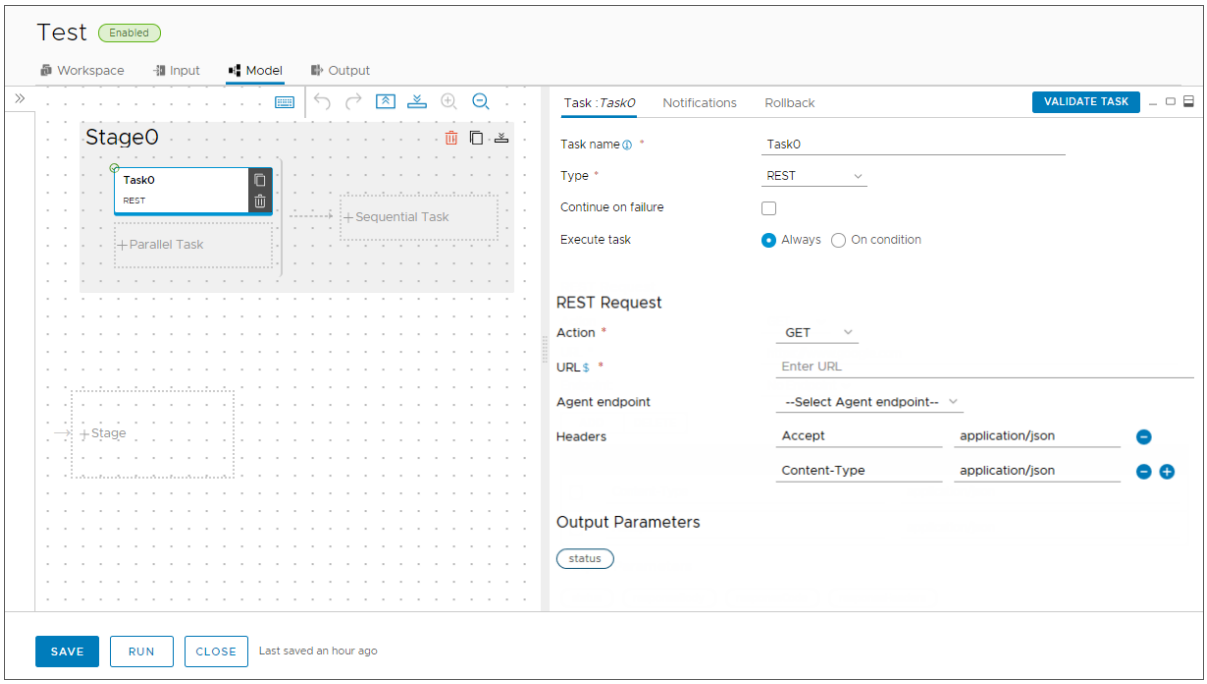
c 在 [REST 請求] 區域中，選取 **GET**。

若要讓 REST 工作從其他應用程式請求資料，請選取 GET 方法。若要向其他應用程式傳送資料，請選取 POST 方法。

d 輸入用於識別 REST API 端點的 URL。例如，`https://www.google.com`。

若要讓 REST 工作從其他應用程式匯入資料，可包含裝載變數。例如，針對匯入動作，您可以輸入 `${Stage0.export.responseBody}`。如果回應資料大小超過 5 MB，REST 工作可能會失敗。

e 若要為工作提供授權，請按一下**新增標頭**，然後輸入標頭索引鍵和值。



4 若要儲存管線，請按一下**儲存**。

5 在 [管線] 索引標籤中，按一下**啟用管線**。

6 按一下**儲存**，然後按一下**關閉**。

7 按一下**執行**。

8 若要監視管線的執行狀況，請按一下**執行**。

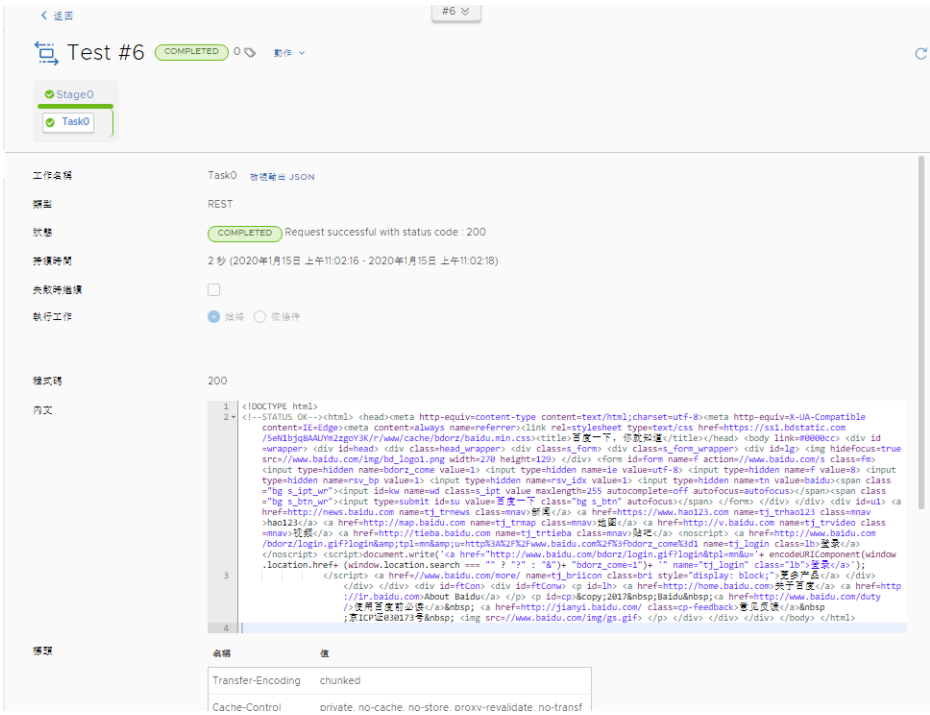
9 若要確認 REST 工作是否傳回預期的資訊，請檢查管線執行和工作結果。

a 管線完成後，若要確認其他應用程式是否傳回了所要求的資料，請按一下管線執行的連結。

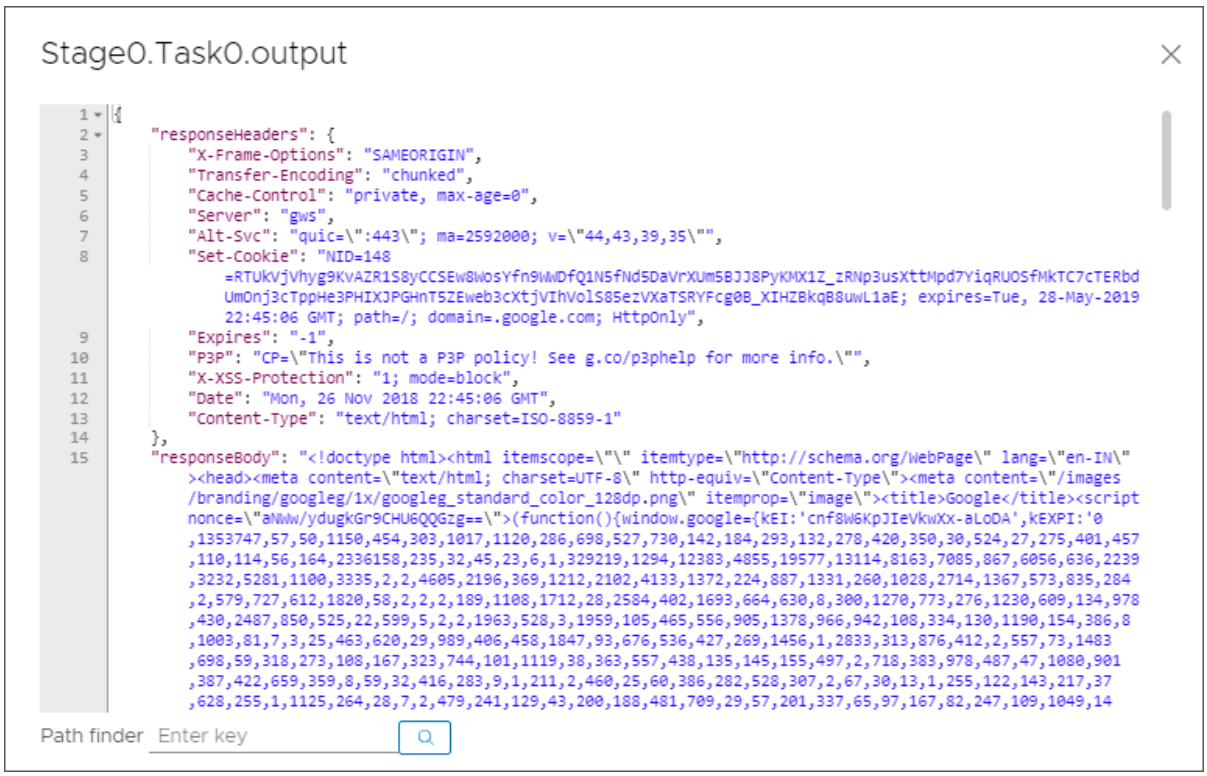
b 按一下管線中的 REST 工作。

c 在管線執行中，按一下相關工作，觀察工作詳細資料，並確認 REST 工作是否傳回了預期結果。

工作詳細資料顯示回應代碼、本文、標頭索引鍵和值。



10 若要查看 JSON 輸出，請按一下**檢視輸出 JSON**。




```
{
    "responseHeaders": {
        "X-Frame-Options": "SAMEORIGIN",
        "Transfer-Encoding": "chunked",
        "Cache-Control": "private, max-age=0",
        "Server": "gws",
        "Alt-Svc": "quic=\":443\"; ma=2592000; v=\"44,43,39,35\"",
        "Set-Cookie": "NID=148
            =RTUkVjVhyg9KvAZR1S8yCCSEw8WosYfn9WWDfQ1N5fNd5DaVrXUm5BJJ8PyKMX1Z_zRNp3usXttMpd7YiqRUOSfMkTC7cTERbd
            UmOnj3cTppHe3PHIXJPGHnT5ZEweb3cXtjVIhVolS85ezVXaTSRYFcg0B_XIHZBkqB8uwL1aE; expires=Tue, 28-May-2019
            22:45:06 GMT; path=/; domain=.google.com; HttpOnly",
        "Expires": "-1",
        "P3P": "CP=\"This is not a P3P policy! See g.co/p3phelp for more info.\"",
        "X-XSS-Protection": "1; mode=block",
        "Date": "Mon, 26 Nov 2018 22:45:06 GMT",
        "Content-Type": "text/html; charset=ISO-8859-1"
    },
    "responseBody": "<!doctype html><html itemscope=\"\" itemtype=\"http://schema.org/WebPage\" lang=\"en-IN\"
        ><head><meta content=\"text/html; charset=UTF-8\" http-equiv=\"Content-Type\"><meta content=\"/images
        /branding/googleg/1x/googleg_standard_color_128dp.png\" itemprop=\"image\"><title>Google</title><script
        nonce=\"aNWw/ydugkGr9CHU6QQGzg==\">(function(){window.google={kEI:'cnf8W6KpJIeVkwXx-aLoDA',kEXPI:'0
        ,1353747,57,50,1150,454,303,1017,1120,286,698,527,730,142,184,293,132,278,420,350,30,524,27,275,401,457
        ,110,114,56,164,2336158,235,32,45,23,6,1,329219,1294,12383,4855,19577,13114,8163,7085,867,6056,636,2239
        ,3232,5281,1100,3335,2,2,4605,2196,369,1212,2102,4133,1372,224,887,1331,260,1028,2714,1367,573,835,284
        ,2,579,727,612,1820,58,2,2,2,189,1108,1712,28,2584,402,1693,664,630,8,300,1270,773,276,1230,609,134,978
        ,430,2487,850,525,22,599,5,2,2,1963,528,3,1959,105,465,556,905,1378,966,942,108,334,130,1190,154,386,8
        ,1003,81,7,3,25,463,620,29,989,406,458,1847,93,676,536,427,269,1456,1,2833,313,876,412,2,557,73,1483
        ,698,59,318,273,108,167,323,744,101,1119,38,363,557,438,135,145,155,497,2,718,383,978,487,47,1080,901
        ,387,422,659,359,8,59,32,416,283,9,1,211,2,460,25,60,386,282,528,307,2,67,30,13,1,255,122,143,217,37
        ,628,255,1,1125,264,28,7,2,479,241,129,43,200,188,481,709,29,57,201,337,65,97,167,82,247,109,1049,14
```


結果

恭喜您！您已設定了一個 REST 工作，該工作叫用 REST API，並使用 REST 外掛程式在 Code Stream 與其他應用程式之間傳送資訊。

後續步驟

繼續在管線中使用 REST 工作執行命令，並將 Code Stream 與其他應用程式整合，以便開發並傳遞軟體應用程式。考慮使用輪詢工作對 API 進行輪詢，直到完成且管線工作滿足結束條件為止。

如何在 Code Stream 中利用程式碼形式的管線

做為 DevOps 管理員或開發人員，您可能想要使用 YAML 程式碼 (而非使用者介面) 在 Code Stream 中建立管線。當您建立程式碼形式的管線時，可以使用任何編輯器，並在管線程式碼中插入註解。

在管線程式碼中，您可以參考外部組態，例如環境變數和安全性認證。當您更新在管線程式碼中使用的變數時，無需更新管線程式碼即可更新這些變數。

您可以將管線 YAML 程式碼用作範本來複製和建立其他管線，並與其他人共用範本。

您可以將管線程式碼範本儲存在原始檔控制存放庫中，從而對其進行版本設定並追蹤更新。透過使用原始檔控制系統，您可以輕鬆備份管線程式碼，並視需要進行還原。

必要條件

- 確認您具有代碼編輯器。

- 如果您打算將管線程式碼儲存在原始檔控制存放庫中，請確認您可以存取工作執行個體。

程序

1 在代碼編輯器中，建立檔案。

2 複製並貼上範例管線程式碼，並將其更新以反映您的特定管線需求。

3 若要將端點納入管線程式碼，請複製並貼上範例端點程式碼，並將其更新以反映您的端點。

在管線工作區中使用 Kubernetes API 端點時， Code Stream 會建立必要的 Kubernetes 資源 (如 ConfigMap、密碼和網繭) 以執行持續整合 (CI) 工作或自訂工作。 Code Stream 使用 NodePort 與容器進行通訊。

Code Stream 管線工作區支援使用 Docker 和 Kubernetes 執行持續整合工作和自訂工作。

如需有關設定工作區的詳細資訊，請參閱設定管線工作區。

4 儲存程式碼。

5 若要儲存管線程式碼並對其進行版本設定，請將該程式碼簽入原始檔控制存放庫。

6 建立持續整合和交付管線時，必須匯入 Kubernetes YAML 檔案。

若要匯入 Kubernetes YAML 檔案，請在智慧管線範本的「持續交付」區域中選取該檔案，然後按一下**處理**。或者，使用 API。

結果

透過使用程式碼範例，您已建立表示管線和端點的 YAML 程式碼。

範例： 管線和端點的範例 YAML 程式碼

此範例 YAML 程式碼包括一些區段，用於表示管線中的 Code Stream 原生建置、階段、工作、通知等的工作區。

如需支援外掛程式的程式碼範例，請參閱第 6 章 將 Code Stream 連線至端點

```
---
kind: PIPELINE
name: myPipelineName
tags:
  - tag1
  - tag2


# Ready for execution
enabled: false

#Max number of concurrent executions
concurrency: 10

#Input Properties
input:
  input1: '30'
  input2: 'Hello'
```

```
#Output Properties
output:
  BuildNo: '${Dev.task1.buildNo}'
  Image: '${Dev.task1.image}'

#Workspace Definition
ciWorkspace:
  image: docker:maven-latest
  path: /var/tmp
  endpoint: my-k8s
  cache:
    - ~/.m2

# Starred Properties
starred:
  input: input1
  output: output1

# Stages in order of execution
stageOrder:
  - Dev
  - QA
  - Prod

# Task Definition Section
stages:
  Dev:
    taskOrder:
      - Task1, Task6
      - Task2 Long, Task Long Long
      - Task5
    tasks:
      Task1:
        type: jenkins
        ignoreFailure: false
        preCondition: ''
        endpoints:
          jenkinsServer: myJenkins
        input:
          job: Add Two Numbers
          parameters:
            number1: 10
            number2: 20
      Task2:
        type: blah
        # repeats like Task1 above
  QA:
    taskOrder:
      - TaskA
      - TaskB
    tasks:
      TaskA:
        type: ssh
        ignoreFailure: false
```

```
        preCondition: ''
        input:
          host: x.y.z.w
          username: abcd
          password: ${var.mypassword}
          script: >
            echo "Hello, remote server"
      TaskB:
        type: blah
        # repeats like TaskA above

# Notificatons Section
notifications:
  email:
    - stage: Dev #optional ; if not found - use pipeline scope
      task: Task1 #optional; if not found use stage scope
      event: SUCCESS
      endpoint: default
      to:
         - user@yourcompany.com
         - abc@yourcompany.com
      subject: 'Pipeline ${name} has completed successfully'
      body: 'Pipeline ${name} has completed successfully'

  jira:
    - stage: QA #optional ; if not found - use pipeline scope
      task: TaskA #optional; if not found use stage scope
      event: FAILURE
      endpoint: myJiraServer
      issuetype: Bug
      project: Test
      assignee: abc
      summary: 'Pipeline ${name} has failed'
      description: |-
        Pipeline ${name} has failed
        Reason - ${resultsText}
  webhook:
    - stage: QA #optional ; if not found - use pipeline scope
      task: TaskB #optional; if not found use stage scope
      event: FAILURE
      agent: my-remote-agent
      url: 'http://www.abc.com'
      headers: #requestHeaders: '{"build_no":"123","header2":"456"}'
          Content-Type: application/json
          Accept: application/json
      payload: |-
        Pipeline ${name} has failed
        Reason - ${resultsJson}
---
```

此 YAML 程式碼表示範例 Jenkins 端點。

```
---
name: My-Jenkins
tags:
- My-Jenkins
- Jenkins
kind: ENDPOINT
properties:
  offline: true
  pollInterval: 15.0
  retryWaitSeconds: 60.0
  retryCount: 5.0
  url: http://urlname.yourcompany.com:8080
description: Jenkins test server
type: your.jenkins:JenkinsServer
isLocked: false
---
```

此 YAML 程式碼表示範例 Kubernetes 端點。

```
---
name: my-k8s
tags: [
  ]
kind: ENDPOINT
properties:
  kubernetesURL: https://urlname.examplelocation.amazonaws.com
  userName: admin
  password: encryptedpassword
description: ''
type: kubernetes:KubernetesServer
isLocked: false
---
```

後續步驟

執行管線，並視需要進行任何調整。請參閱如何執行管線和查看結果。

將 Code Stream 連線至端點

6

Code Stream 透過外掛程式與開發工具進行整合。支援的外掛程式包括 Jenkins、Bamboo、vRealize Operations、Bugzilla、Team Foundation Server、Git 等。

也可以開發您自己的外掛程式來整合 Code Stream 與其他開發應用程式。

若要將 Code Stream 與 Jira 整合，則不需要外部外掛程式，因為 Code Stream 會將 Jira 票證建立功能作為通知類型納入。若要建立關於管線狀態的 Jira 票證，您必須新增 Jira 端點。

本章節討論下列主題：

- 什麼是 Code Stream 中的端點
- 如何整合 Code Stream 與 Jenkins
- 如何整合 Code Stream 與 Git
- 如何整合 Code Stream 與 Gerrit
- 如何將 Code Stream 與 vRealize Orchestrator 整合

什麼是 Code Stream 中的端點

端點是 DevOps 應用程式的執行個體，它連線至 Code Stream 並提供用於執行管線的資料，例如資料來源、存放庫或通知系統。

您在 Code Stream 中的角色會決定使用端點的方式。

- 管理員和開發人員可以建立、更新、刪除和檢視端點。
- 管理員可以將端點標記為受限制，並執行使用受限制端點的管線。
- 具有檢視者角色的使用者可以查看端點，但無法建立、更新或刪除它們。

如需詳細資訊，請參閱如何在 Code Stream 中管理使用者存取和核准。

若要將 Code Stream 連線到端點，請執行以下步驟。

1 在管線中新增工作

2 設定工作，使其與端點進行通訊。

3 按一下**驗證**，確認 Code Stream 是否可以連線到端點。

4 然後，當您執行管線時，該工作將連線到端點，以便可以執行該工作。

如需有關使用這些端點的工作類型的資訊，請參閱在 Code Stream 中提供哪些類型的工作。

表 6-1. Code Stream 支援的端點

端點	它提供什麼	支援的版本	需求
Bamboo	建立建置計劃。	6.9.*	
Docker	原生建置可以使用 Docker 主機進行部署。		當管線包含來自 Docker Hub 的映像時，您必須在執行管線之前確保該映像已內嵌 `cURL` 或 `wget`。當管線執行時，Code Stream 將下載使用 `cURL` 或 `wget` 執行命令的二進位檔案。
Docker 登錄	登錄容器映像，以便 Docker 建置主機可以提取映像。	2.7.1	
Gerrit	連線至 Gerrit 伺服器以供檢閱和觸發	2.14.*	
Git	當開發人員更新程式碼並將其簽入存放庫時觸發管線。	Git Hub Enterprise 2.1.8 Git Lab Enterprise 11.9.12-ee	
Jenkins	建置程式碼構件。	1.6.* 和 2.*	
Jira	將在管線工作失敗時建立 Jira 票證。	8.3.*	
Kubernetes	自動執行部署、縮放和管理容器化應用程式的步驟。	支援的所有版本 (對於 Cloud Assembly 8.4 及更新版本) 1.18 (對於 Cloud Assembly 8.3 及更早版本)	在管線工作區中使用 Kubernetes API 端點時， Code Stream 會建立必要的 Kubernetes 資源 (如 ConfigMap、密碼和網繭) 以執行持續整合 (CI) 工作或自訂工作。 Code Stream 使用 NodePort 與容器進行通訊。 如需有關設定工作區的詳細資訊，請參閱設定管線工作區。
PowerShell	建立在 Windows 或 Linux 機器上執行 PowerShell 指令碼的工作。	4 和 5	
SSH	建立在 Windows 或 Linux 機器上執行 SSH 指令碼的工作。	7.0	
TFS，Team Foundation Server	管理原始程式碼、自動化建置、測試和相關活動。	2015 和 2017	
vRealize Orchestrator	排列和自動執行建置程序中的工作流程。	7.* 和 8.*	

GitHub 端點的範例 YAML 程式碼

此範例 YAML 程式碼會定義一個 GitHub 端點，您可以在 Git 工作中參考此端點。

```
---
name: github-k8s
tags: [
  ]
kind: ENDPOINT
properties:
  serverType: GitHub
  repoURL: https://github.com/autouser/testrepok8s
  branch: master
  userName: autouser
  password: encryptedpassword
  privateToken: ''
description: ''
type: scm:git
isLocked: false
---
```

如何整合 Code Stream 與 Jenkins

Code Stream 提供 Jenkins 外掛程式，此外掛程式可觸發建置和測試原始程式碼的 Jenkins 工作。Jenkins 外掛程式可執行測試案例，並且可以使用自訂指令碼。

若要在管線中執行 Jenkins 工作，您可以使用 Jenkins 伺服器，並且在 Code Stream 中新增 Jenkins 端點。然後，您可以建立管線並向其新增 Jenkins 工作。

使用 Code Stream 中的 Jenkins 工作和 Jenkins 端點時，可以在 Jenkins 中建立支援多分支工作的管線。多分支工作包括 Git 存放庫之每個分支中的個別工作。在支援多分支工作的 Code Stream 中建立管線時：

- Jenkins 工作可以執行位於 Jenkins 伺服器上多個資料夾中的 Jenkins 工作。
- 您可以覆寫 Jenkins 工作組態中的資料夾路徑，以便其使用不同的資料夾路徑，這將覆寫在 Code Stream 的 Jenkins 端點中定義的預設路徑。
- Code Stream 中的多分支管線會在 Git 存放庫或 GitHub 存放庫中偵測類型為 `.groovy` 的 Jenkins 工作檔案，然後開始為存放庫中掃描的每個分支建立工作。
- 您可以使用 Jenkins 工作組態中提供的路徑來取代 Jenkins 端點中定義的預設路徑，然後執行與主要 Jenkins 工作內的任何分支相關聯的工作與管線。

必要條件

- 設定執行 1.561 版或更新版本的 Jenkins 伺服器。
- 確認您是 Code Stream 中的專案的成員。如果您不是成員，請要求 Code Stream 管理員將您新增為專案成員。請參閱如何在 Code Stream 中新增專案。
- 確認 Jenkins 伺服器上存在某個工作，以便管線工作能夠執行此工作。

程序

1 新增並驗證 Jenkins 端點。

   a 按一下**端點 > 新增端點**。

   b 選取專案，針對端點類型選取 **Jenkins**。然後，輸入名稱與說明。

   c 如果此端點是您基礎結構中的業務關鍵元件，請啟用**標記為受限制**。

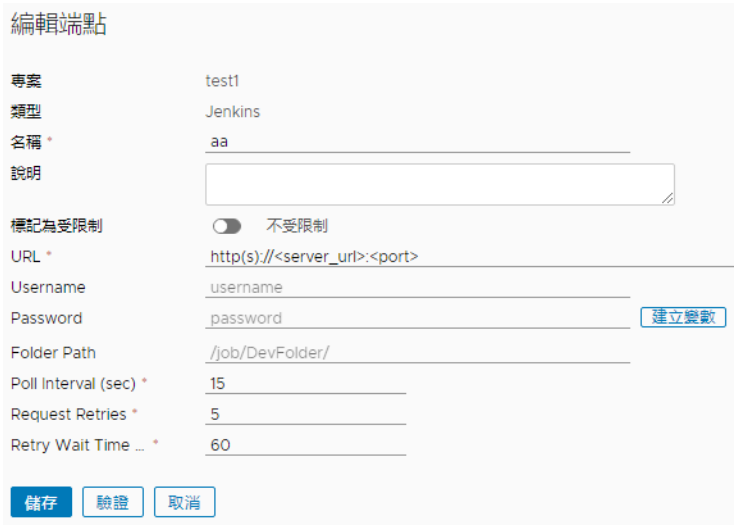
   d 輸入 Jenkins 伺服器的 URL。

e 輸入用於登入 Jenkins 伺服器的使用者名稱和密碼。然後輸入其餘資訊。

表 6-2. Jenkins 端點的其餘資訊

端點項目	說明
資料夾路徑	將工作進行分組的資料夾的路徑。Jenkins 可以執行資料夾中的所有工作。您可以建立子資料夾。例如： ■ `folder_1` 可包含 `job_1` ■ `folder_1` 可包含 `folder_2` (其中可能包含 `job_2`) 針對 `folder_1` 建立端點時，資料夾路徑為 `job/folder_1`，且端點僅列出 `job_1`。 若要取得名為 `folder_2` 之子資料夾中的工作清單，您必須建立另一個使用資料夾路徑 `/job/folder_1/job/folder_2/` 的端點。
多分支 Jenkins 工作的資料夾路徑	若要支援多分支 Jenkins 工作，請在 Jenkins 工作中輸入包括 Jenkins 伺服器 URL 的完整路徑和完整工作路徑。在 Jenkins 工作中包含資料夾路徑時，該路徑會覆寫 Jenkins 端點中顯示的路徑。透過 Jenkins 工作中的自訂資料夾路徑， Code Stream 僅顯示存在於該資料夾中的工作。 ■ 例如：`https://server.yourcompany.com/job/project` ■ 如果管線還必須觸發主要 Jenkins 工作，則使用 `https://server.yourcompany.com/job/project/job/main`
URL	Jenkins 伺服器的主機 URL。輸入 `protocol://host:port` 格式的 URL。例如：`http://192.10.121.13:8080`
輪詢間隔	Code Stream 輪詢 Jenkins 伺服器以進行更新的間隔持續時間。
請求重試計數	針對 Jenkins 伺服器重試已排程的建置請求的次數。
重試等待時間	針對 Jenkins 伺服器重試建置請求前等待的秒數。

f 按一下**驗證**，並確認端點會連線至 Code Stream。如果無法連線，請更正任何錯誤，然後按一下**儲存**。



2 若要建置程式碼，請建立管線，並新增使用 Jenkins 端點的工作。

a 按一下**管線 > 新增管線 > 空白畫布**。

b 按一下預設階段。

c 在 [工作] 區域中，輸入工作的名稱。

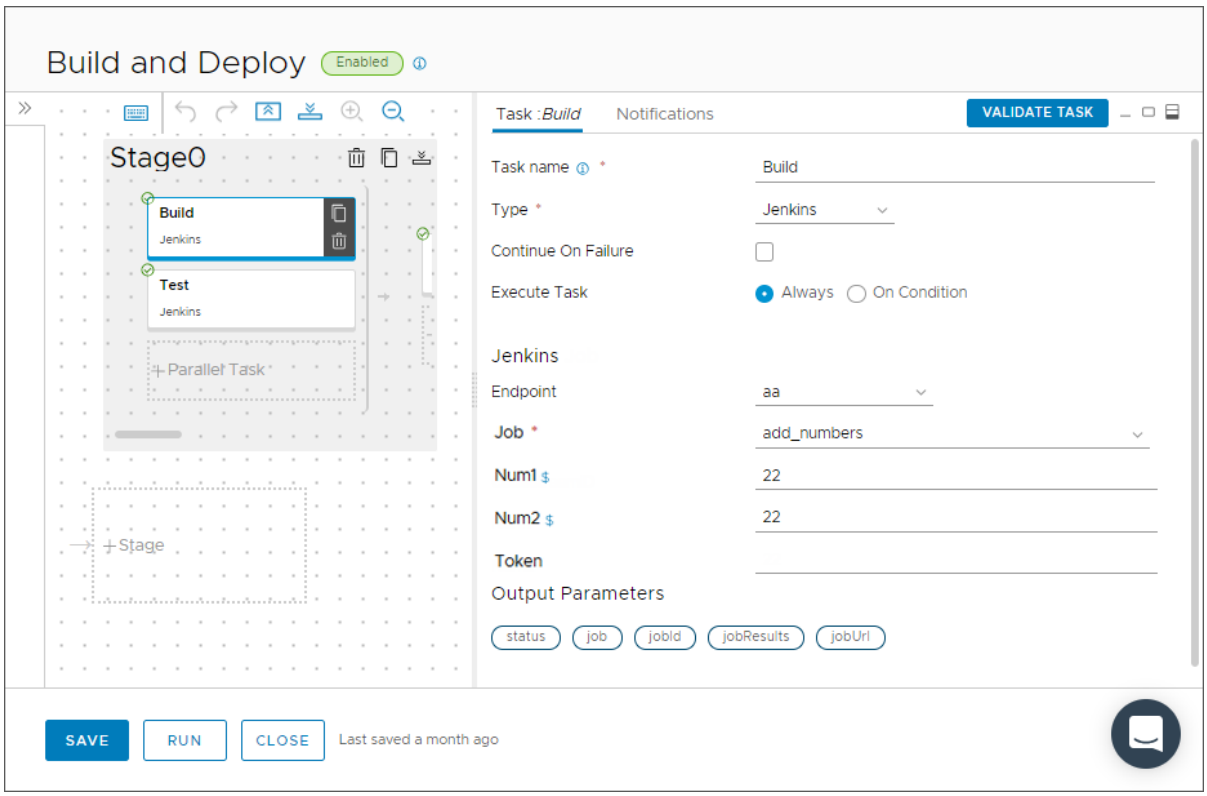
d 選取工作類型為 **Jenkins**。

e 選取您建立的 Jenkins 端點。

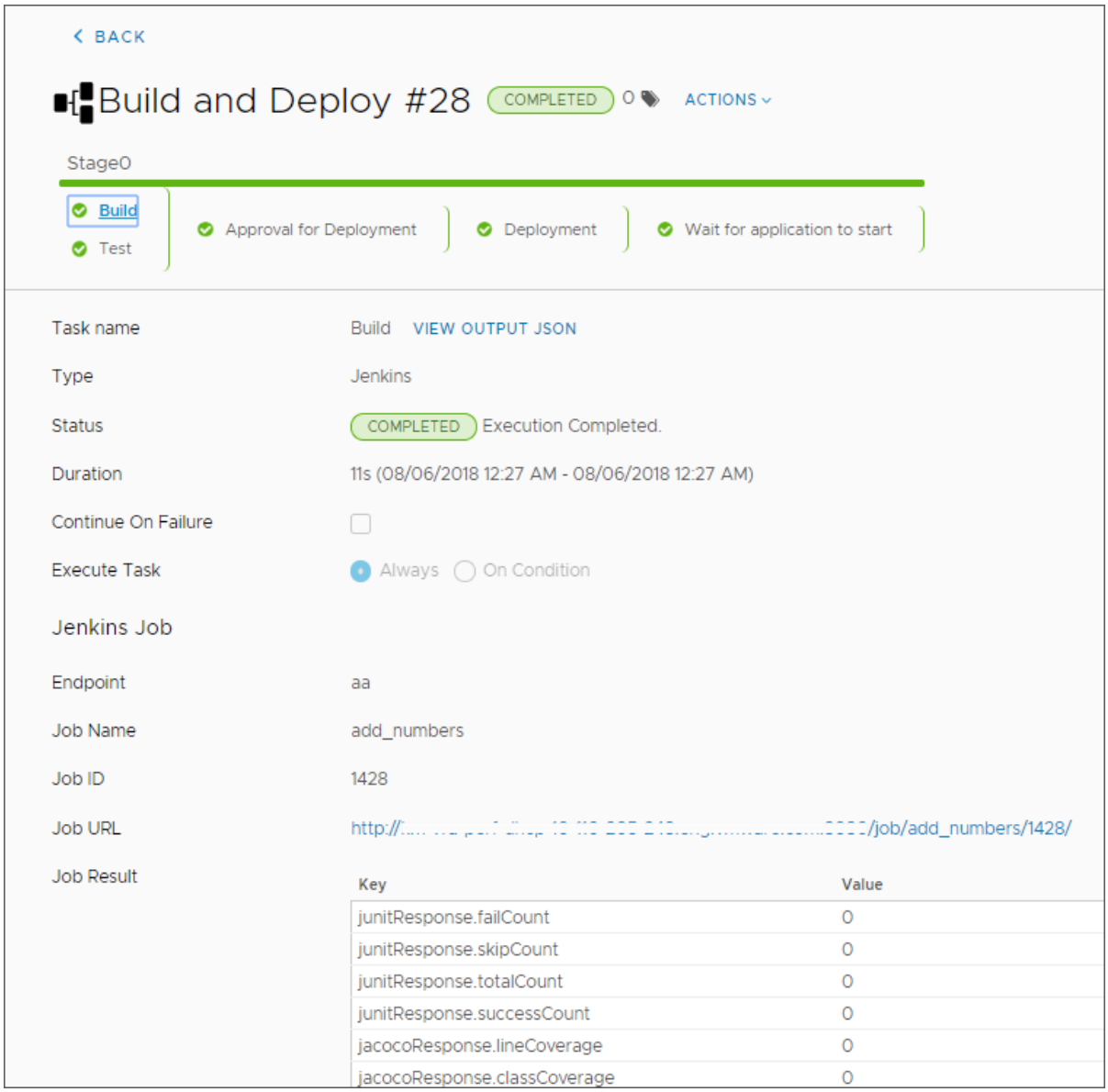
f 從下拉式功能表中，從管線將執行的 Jenkins 伺服器中選取工作。

g 輸入工作的參數。

h 輸入 Jenkins 工作的驗證 Token。

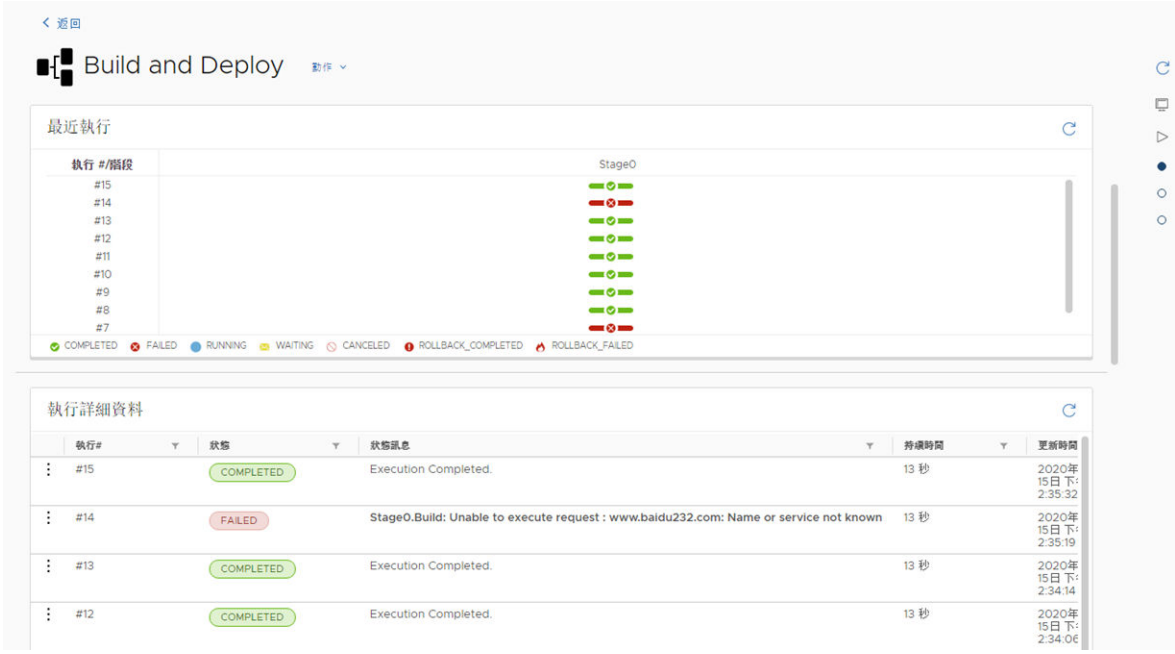


3 啟用和執行管線，並檢視管線執行。



4 在管線儀表板上查看執行詳細資料和狀態。

您可以識別任何失敗及失敗原因。您還可以查看有關管線執行持續時間、完成和失敗的趨勢。



結果

恭喜您！透過新增端點、建立管線，並設定建置程式碼的 Jenkins 工作，以整合 Code Stream 與 Jenkins。

範例： Jenkins 建置工作的範例 YAML

對於此範例中使用的 Jenkins 建置工作類型，YAML 類似於下列程式碼，其中通知已開啟：

```
test:
  type: Jenkins
  endpoints:
    jenkinsServer: jenkins
  input:
    job: Add two numbers
    parameters:
      Num1: '23'
      Num2: '23'
```

後續步驟

檢閱其他區段以瞭解更多資訊。請參閱第 6 章 將 Code Stream 連線至端點。

如何整合 Code Stream 與 Git

如果您的 GitHub、GitLab 或 Bitbucket 存放庫中發生程式碼變更， Code Stream 會提供觸發管線的方式。Git 觸發器在要監控的存放庫的分支上使用 Git 端點。 Code Stream 透過 webhook 連線至 Git 端點。

若要在 Code Stream 中定義 Git 端點，您可以選取專案，並輸入端點所在的 Git 存放庫的分支。專案將管線與端點及其他相關物件進行分組。當您在 webhook 定義中選擇專案時，請選取要觸發的端點和管線。

備註 如果定義包含端點的 webhook，並稍後編輯此端點，則無法在 webhook 中變更端點詳細資料。若要變更端點詳細資料，您必須刪除並重新定義包含此端點的 webhook。請參閱如何在 Code Stream 中使用 Git 觸發器執行管線。

透過使用相同的 Git 端點並在 Webhook 組態頁面中為分支名稱提供不同的值，可以為不同的分支建立多個 Webhook。若要為相同 Git 存放庫中的其他分支建立其他 Webhook，無需針對多個分支複製 Git 端點多次。而是在 Webhook 中提供分支名稱，以便重複使用 Git 端點。如果 Git Webhook 中的分支與端點中的分支相同，則無需在 Git Webhook 頁面中提供分支名稱。

必要條件

- 確認您可以存取您打算連線的 GitHub、GitLab 或 Bitbucket 存放庫。
- 確認您是 Code Stream 中的專案的成員。如果您不是管理員，請要求 Code Stream 管理員將您新增為專案成員。請參閱如何在 Code Stream 中新增專案。

程序

1 定義 Git 端點。

   a 按一下**端點 > 新增端點**。

   b 選取專案，並針對端點類型選取 **Git**。然後，輸入名稱與說明。

   c 如果此端點是您基礎結構中的業務關鍵元件，請啟用**標記為受限制**。

   在管線中使用受限制的端點時，管理員可以執行管線，並且必須核准管線執行。如果端點或變數標記為受限制且非管理使用者觸發了管線，則管線會在該工作暫停並等待管理員繼續執行。

   專案管理員可以啟動包括受限制端點或變數在內的管線，前提是這些資源位於使用者作為專案管理員的專案中。

   如果非管理員使用者嘗試執行包含受限制資源的管線，則管線會在使用受限制資源的工作中停止。然後，管理員必須恢復管線。

   如需有關受限制資源以及包括稱為**管理受限制的管線**之權限的自訂角色的詳細資訊，請參閱：

   - 如何在 Code Stream 中管理使用者存取和核准
   - 第 2 章 設定 Code Stream 以建立發佈程序模型

   d 選取其中一個支援的 Git 伺服器類型。

   e 在路徑中輸入具有伺服器之 API 閘道之存放庫的 URL。例如：

   對於 GitHub，輸入：**`https://api.github.com/vmware-example/repo-example`**

   對於 BitBucket，輸入：**`https://api.bitbucket.org/{user}/{repo name}`** 或 **`http(s)://{bitbucket-enterprise-server}/rest/api/1.0/users/{username}/repos/{repo name}`**

f 輸入端點所在的存放庫中的分支。

g 選取驗證類型，然後輸入 GitHub、GitLab 或 BitBucket 的使用者名稱。然後輸入與使用者名稱相符的私人 Token。

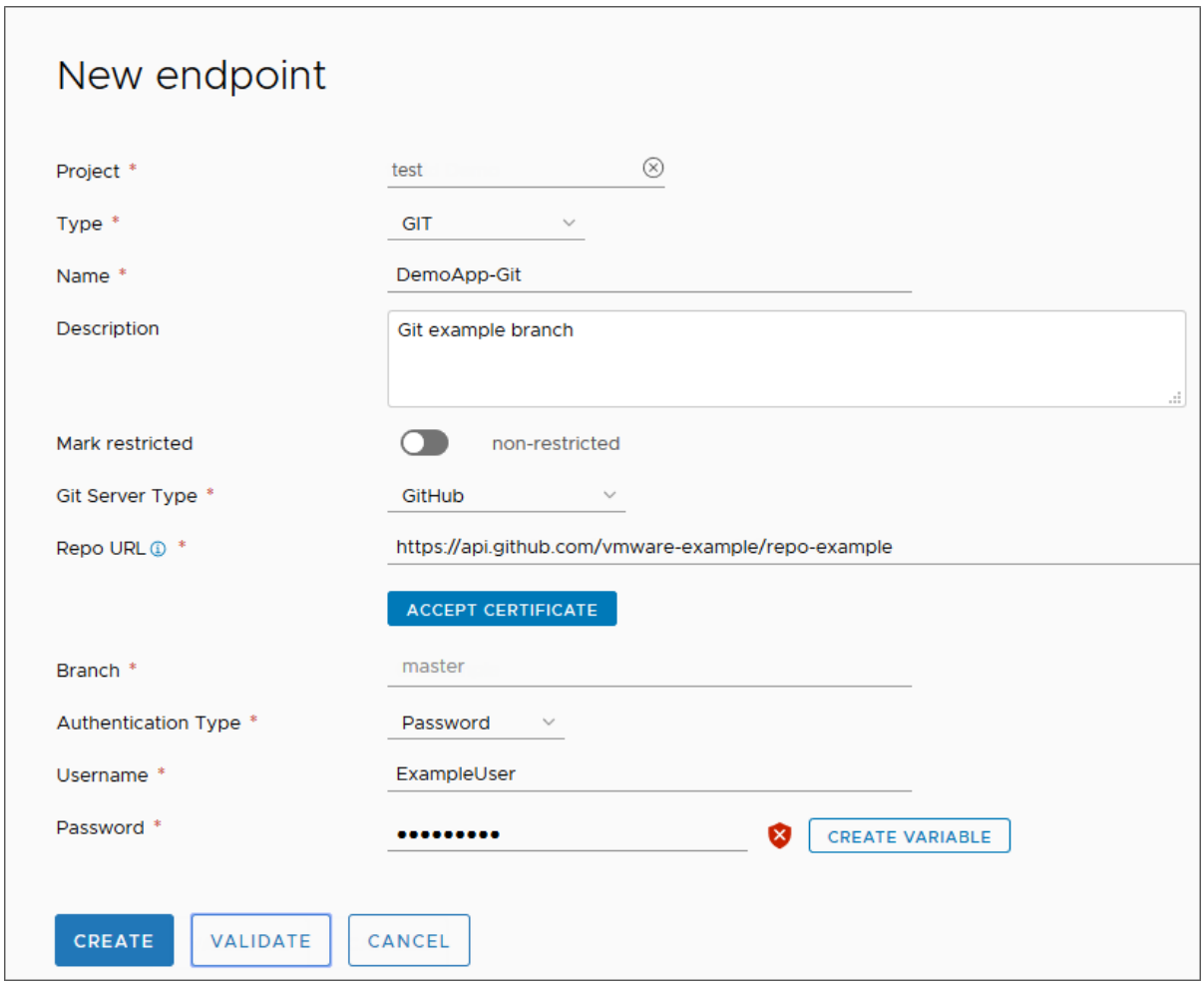
- 密碼。若要稍後建立 Webhook，您必須輸入密碼的私人 Token。適用於 Git 的 Webhook 不支援使用基本驗證建立的端點。

  使用密碼變數隱藏和加密敏感資訊。針對必須隱藏和加密以及限制在執行中使用的字串、密碼和 URL 使用受限制的變數。例如，針對密碼或 URL 使用密碼變數。您可以在管線中任何類型的工作中使用密碼和受限制的變數。

- 私人 Token。此 Token 為 Git 專屬，並且提供對特定動作的存取權。請參閱 https://docs.gitlab.com/ee/user/profile/personal_access_tokens.html。您也可以為私人 Token 建立變數。

2 按一下**驗證**，並確認端點會連線至 Code Stream。

如果無法連線，請更正任何錯誤，然後按一下**建立**。



後續步驟

若要進一步瞭解，請檢閱其他區段。請參閱如何在 Code Stream 中使用 Git 觸發器執行管線。

如何整合 Code Stream 與 Gerrit

透過 Code Stream，可以在 Gerrit 專案中進行代碼檢閱時觸發管線。適用於 Gerrit 的觸發器定義包括 Gerrit 專案和必須針對不同事件類型執行的管線。

適用於 Gerrit 的觸發器在將要監控的 Gerrit 伺服器上使用 Gerrit 接聽程式。若要在 Code Stream 中定義 Gerrit 端點，您可以選取專案，並輸入 Gerrit 伺服器的 URL。然後，在該伺服器上建立 Gerrit 接聽程式時指定端點。

如果在已啟用 FIPS 的 vRealize Automation 執行個體中將 Gerrit 伺服器用作 Code Stream 端點，則必須確認您的 Gerrit 組態檔是否包含正確的訊息驗證金鑰。如果 Gerrit 伺服器組態檔不包含正確的訊息驗證金鑰，則伺服器無法正確啟動，並顯示此訊息：`PrivateKey/PassPhrase is incorrect`

必要條件

- 確認您可以存取打算連線的 Gerrit 伺服器。
- 確認您是 Code Stream 中的專案的成員。如果您不是成員，請要求 Code Stream 管理員將您新增為專案成員。請參閱如何在 Code Stream 中新增專案。

程序

1. 定義 Gerrit 端點。
   a. 按一下**設定 > 端點**，然後按一下**新增端點**。
   b. 選取專案，針對端點類型選取 **Gerrit**。然後，輸入名稱與說明。
   c. 如果此端點是您基礎結構中的業務關鍵元件，請啟用**標記為受限制**。
   d. 輸入 Gerrit 伺服器的 URL。

      若要使用預設連接埠，您可以提供帶有 URL 的連接埠號碼或將該值留空。
   e. 輸入 Gerrit 伺服器的使用者名稱和密碼。

      如果必須加密密碼，請按一下**建立變數**，然後選取類型：
      - 密碼。當具有任何角色的使用者執行管線時，密碼即會解析。
      - 受限制。當具有管理員角色的使用者執行管線時，密碼即會解析。

      對於值，請輸入必須是安全的密碼，例如 Jenkins 伺服器的密碼。
   f. 對於私密金鑰，輸入用於安全存取 Gerrit 伺服器的 SSH 金鑰。

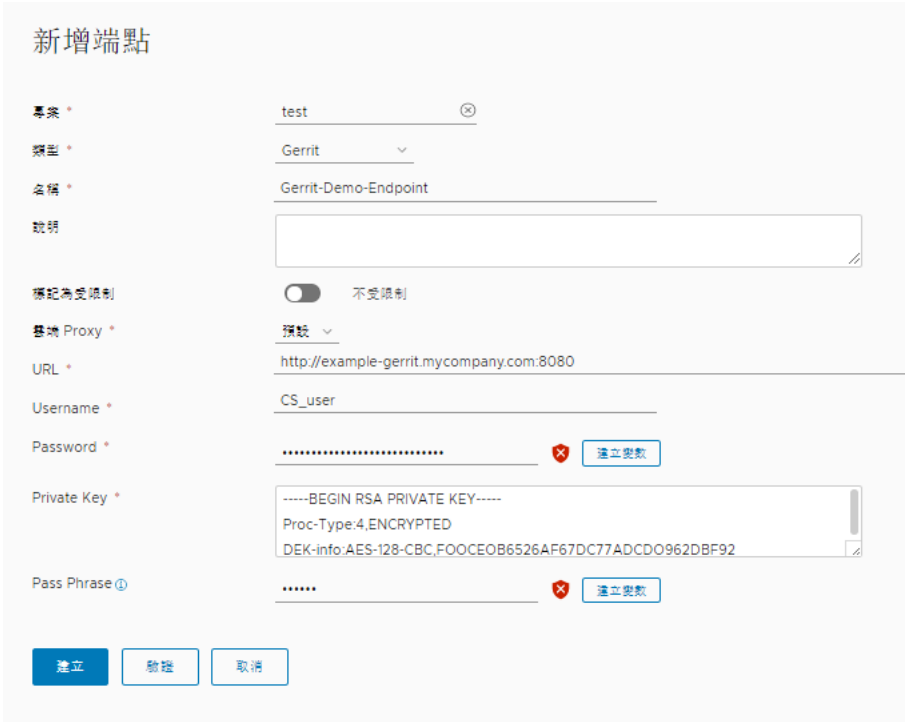
      此金鑰是位於 `.ssh` 目錄中的 RSA 私密金鑰。
   g. (選擇性) 如果複雜密碼與私密金鑰相關聯，請輸入複雜密碼。

      若要加密複雜密碼，請按一下**建立變數**並選取類型：
      - 密碼。當具有任何角色的使用者執行管線時，密碼即會解析。
      - 受限制。當具有管理員角色的使用者執行管線時，密碼即會解析。

      對於值，請輸入必須是安全的複雜密碼，例如 SSH 伺服器的複雜密碼。

2 按一下**驗證**，然後確認 Code Stream 中的 Gerrit 端點是否連線至 Gerrit 伺服器。

如果無法連線，請更正任何錯誤，然後再次按一下**驗證**。



3 按一下**建立**。

4 確認 vRealize Automation 環境是否已啟用 FIPS，或者是否透過 Jenkins 工作使用 Jenkins URL 建立已啟用 FIPS 的環境。

a 若要從命令列執行命令，請透過 SSH 連線至 vRealize Automation 8.x 應用裝置，並以 root 使用者身分登入。例如，透過連接埠 22、5480 或 443 連線至完整網域名稱 URL，例如 `https://cava-1-234-567.yourcompanyFQDN.com`。

b 若要檢查 vRealize Automation 上的 FIPS，請執行命令 `vracli security fips`。

c 確認命令是否傳回 `FIPS mode: strict`。

5 如果您的 Gerrit 伺服器是已啟用 FIPS 的 vRealize Automation 執行個體中的端點，請確保您的 Gerrit 組態檔包含正確的訊息驗證 (MAC) 金鑰。

a 開啟 Gerrit 並建立 SSH 金鑰配對。

b 在 `'$site_path'/etc/gerrit.config` 中找到 Gerrit 伺服器組態檔。

c 確認 Gerrit 伺服器組態檔是否包含一或多個訊息驗證碼 (MAC) 金鑰 (`hmac-MD5` 除外)。

備註 在 FIPS 模式中，`hmac-MD5` 不是支援的 MAC 演算法。若要確保 Gerrit 伺服器正確啟動，Gerrit 伺服器組態檔必須排除此演算法。如果 Gerrit 伺服器未正確啟動，會顯示此訊息：`PrivateKey/PassPhrase is incorrect`

啟用了以加號 (+) 開頭的受支援訊息驗證碼 (MAC) 金鑰名稱。從預設 MAC 清單中移除了以連字號 (-) 開頭的 MAC 金鑰名稱。依預設，在 Code Stream 中針對 Gerrit 伺服器提供了以下支援的 MAC：

- `hmac-md5-96`
- `hmac-sha1`
- `hmac-sha1-96`
- `hmac-sha2-256`
- `hmac-sha2-512`

後續步驟

若要進一步瞭解，請檢閱其他區段。請參閱如何在 Code Stream 中使用 Gerrit 觸發器執行管線。

如何將 Code Stream 與 vRealize Orchestrator 整合

Code Stream 可以與 vRealize Orchestrator (`vRO`) 進行整合，以透過執行 `vRO` 工作流程來延伸其功能。vRealize Orchestrator 包含許多可與第三方工具整合的預先定義的工作流程。這些工作流程可協助您自動執行和管理 DevOps 程序，自動執行大量作業等。

例如，您可以在管線的 `vRO` 工作中使用工作流程，以啟用使用者、移除使用者、移動虛擬機器，以及與測試架構整合以在管線執行時測試程式碼等。您可以在 code.vmware.com 中瀏覽 vRealize Orchestrator 工作流程的程式碼範例。

透過 vRealize Orchestrator 工作流程，您的管線可以在建置、測試和部署應用程式時執行動作。您可以將預先定義的工作流程包含在管線中，也可以建立並使用自訂工作流程。每個工作流程皆包含輸入、工作和輸出。

若要在管線中執行 `vRO` 工作流程，該工作流程必須顯示在管線中包含的 `vRO` 工作的可用工作流程清單中。

管理員必須先在 vRealize Orchestrator 中執行下列步驟，工作流程才會顯示在管線的 `vRO` 工作中：

1 將 `CODESTREAM` 標籤套用至 `vRO` 工作流程。

2 將 `vRO` 工作流程標記為全域工作流程。

必要條件

- 確認您能夠以管理員身分存取 vRealize Orchestrator 的內部部署執行個體。若要取得協助，請向您自己的管理員詢問並參閱 vRealize Orchestrator 說明文件。
- 確認您是 Code Stream 中的專案的成員。如果您不是管理員，請要求 Code Stream 管理員將您新增為專案成員。請參閱如何在 Code Stream 中新增專案。
- 在 Code Stream 中，建立管線並新增一個階段。

程序

1 做為管理員，請準備 vRealize Orchestrator 工作流程以讓管線執行。

   a 在 vRealize Orchestrator 中，找到您需要在管線中使用的工作流程，例如，啟用使用者的工作流程。

   如果需要的工作流程不存在，您可以建立該工作流程。

   b 在搜尋列中，輸入 **`Tag workflow`**，以找到名為 `Tag workflow` 的工作流程。

   c 在名為 `Tag workflow` 的卡上，按一下**執行**，這將會顯示組態區域。

   d 在標記的工作流程文字區域中，輸入要在 Code Stream 管線中使用的工作流程名稱，然後從清單中加以選取。

   e 在標籤和值文字區域中，以大寫字母輸入 `CODESTREAM`。

   f 按一下名為**全域標籤**的核取方塊。

   g 按一下**執行**，將名為 `CODESTREAM` 的標籤連結至需要在 Code Stream 管線中選取的工作流程。

   h 在導覽窗格中，按一下**工作流程**，並確認名為 `CODESTREAM` 的標籤會顯示在管線將執行的工作流程卡上。

   登入 Code Stream 並將 `vRO` 工作新增至管線後，標記的工作流程會出現在工作流程清單中。

2 在 Code Stream 中，為 vRealize Orchestrator 執行個體建立端點。

   a 按一下**端點 > 新增端點**。

   b 選取專案。

   c 輸入相關的名稱。

d 輸入 vRealize Orchestrator 端點的 URL。

請使用以下格式：**https://vro-appliance.yourdomain.local:8281**

請勿使用此格式：https://vro-appliance.yourdomain.local:8281/vco/api

vRealize Automation 應用裝置中內嵌的 vRealize Orchestrator 執行個體的 URL 是應用裝置的 FQDN，不具有連接埠。例如：**https://vra-appliance.yourdomain.local/vco**

從 vRealize Automation 8.x 開始，對於外部 vRealize Orchestrator Appliance，應用裝置的 FQDN 為 **https://vro-appliance.yourdomain.local**

對於 vRealize Automation 7.x 中包含的外部 vRealize Orchestrator Appliance，應用裝置的 FQDN 為 **https://vro-appliance.yourdomain.local:8281/vco**

如果在新增端點時發生問題，您可能需要匯入具有 SHA-256 憑證指紋且已移除冒號的 YAML 組態。例如，**B0:01:A2:72...** 變為 **B001A272...**。範例 YAML 程式碼類似於：

```
```
---
project: Demo
kind: ENDPOINT
name: external-vro
description: ''
type: vro
properties:
  url: https://yourVROhost.yourdomain.local
  username: yourusername
  password: yourpassword
  fingerprint: <your_fingerprint>
```
```

e 如果您輸入的 URL 需要憑證，請按一下**接受憑證**。

f 輸入 vRealize Orchestrator 伺服器的使用者名稱和密碼。

如果您使用非本機使用者進行驗證，則必須省略使用者名稱的網域部分。例如，若要使用 **svc_vro@yourdomain.local** 進行驗證，則必須在**使用者名稱**文字區域中輸入 **svc_vro**。

3 準備管線以執行 vRO 工作。

a 將 vRO 工作新增至管線階段。

b 輸入相關的名稱。

c 在 [工作流程內容] 區域，選取 vRealize Orchestrator 端點。

d 選取您在 vRealize Orchestrator 中標記為 CODESTREAM 的工作流程。

如果您選取您建立的自訂工作流程，您可能需要輸入輸入參數值。

e 對於**執行工作**，按一下**依條件**。

f 輸入要在管線執行時套用的條件。

何時執行管線...	選取條件...
依條件	只有在已定義的條件評估為 true 時才執行管線工作。如果條件為 false，則略過工作。 vRO 工作允許包含布林運算式，該運算式使用下列運算元和運算子。 ■ 管線變數，例如 `${pipeline.variableName}`。輸入變數時，僅可使用大括號。 ■ 工作輸出變數，例如 `${Stage1.task1.machines[0].value.hostIp[0]}`。 ■ 預設管線繫結變數，例如 `${releasePipelineName}`。 ■ 不區分大小寫的布林值，例如 `true`、`false`、`'true'` 和 `'false'`。 ■ 不含引號的整數值或小數值。 ■ 具有單引號或雙引號的字串值，例如 `"test"` 和 `'test'`。 ■ 字串和數字類型的值，例如 `== Equals` 和 `!= Not Equals`。 ■ 關聯性運算子，例如 `>`、`>=`、`<` 和 `<=`。 ■ 布林邏輯，例如 `&&` 和 `\|\|`。 ■ 算術運算子，例如 `+`、`-`、`*` 和 `/`。 ■ 使用圓括弧的巢狀運算式。 ■ 包含文字值 `ABCD` 的字串評估為 false，因此會略過工作。 ■ 不支援一元運算子。 範例條件可以是 `${Stage1.task1.output} == “Passed” || ${pipeline.variableName} == 39`
一律	如果您選取**一律**，管線執行工作，而不需條件。

g 輸入問候訊息。

h 按一下**驗證工作**，並更正發生的任何錯誤。

4 儲存、啟用並執行管線。

5 管線執行後，檢查結果。

a 按一下**執行**。

b 按一下管線。

c 按一下工作。

d 檢查結果、輸入值和內容。

您可以識別工作流程執行識別碼、回應工作的人員及時間，以及他們加入的任何註解。

結果

恭喜您！您已標記要在 Code Stream 中使用的 vRealize Orchestrator 工作流程，並在 Code Stream 管線中新增了 vRO 工作，以使其執行在 DevOps 環境中自動執行動作的工作流程。

範例： vRO 工作輸出格式

vRO 工作的輸出格式類似於以下範例。

```
[{
                "name": "result",
                "type": "STRING",
                "description": "Result of workflow run.",
                "value": ""
},
{
                "name": "message",
                "type": "STRING",
                "description": "Message",
                "value": ""
}]
```

後續步驟

繼續將 vRO 工作流程工作包含在管線中，以便在開發、測試和生產環境中自動執行工作。

在 Code Stream 中觸發管線

7

您可以使 Code Stream 在發生特定事件時觸發管線。

例如：

- Docker 觸發器可以在新構件建立或更新時執行管線。
- 適用於 Git 的觸發器可以在開發人員更新代碼時觸發管線。
- 適用於 Gerrit 的觸發器可以在開發人員檢閱代碼時觸發管線。

本章節討論下列主題：

如何在 Code Stream 中使用 Docker 觸發器執行持續交付管線

身為 Code Stream 管理員或開發人員，您可以在 Code Stream 中使用 Docker 觸發器。每當建立或更新建置構件時，Docker 觸發器都會執行獨立的持續交付 (CD) 管線。Docker 觸發器將執行 CD 管線，而此管線會將新的或已更新的構件做為容器映像推送至 Docker Hub 存放庫。CD 管線可以做為自動化建置的一部分執行。

例如，若要透過 CD 管線持續部署已更新的容器映像，請使用 Docker 觸發器。當容器映像簽入 Docker 登錄時，Docker Hub 中的 webhook 會通知 Code Stream 該映像已變更。此通知會觸發 CD 管線以使用已更新的容器映像執行，並將映像上傳至 Docker Hub 存放庫。

若要使用 Docker 觸發器，您需要在 Code Stream 中執行數個步驟。

表 7-1. 如何使用 Docker 觸發器

您執行的操作...	有關該動作的詳細資訊...
建立 Docker 登錄端點。	若要讓 Code Stream 觸發管線，您必須有 Docker 登錄端點。如果端點不存在，您可以選取為 Docker 觸發器新增 webhook 時建立端點的選項。 Docker 登錄端點包含 Docker Hub 存放庫的 URL。
向管線新增可在管線執行時自動插入 Docker 參數的輸入參數。	您可以將 Docker 參數插入管線中。參數可包含 Docker 事件擁有者名稱、映像、存放庫名稱、存放庫命名空間和標籤。 您可以在 CD 管線中包含輸入參數，Docker webhook 會在管線觸發之前將這些輸入參數傳遞至管線。
建立 Docker webhook。	在 Code Stream 中建立 Docker webhook 時，它也會在 Docker Hub 中建立對應的 webhook。 Code Stream 中的 Docker webhook 透過包含在 webhook 中的 URL 連線到 Docker Hub。 webhook 彼此通訊，並在 Docker Hub 中建立或更新構件時觸發管線。 如果您稍後更新或刪除 Code Stream 中的 Docker webhook，Docker Hub 中的 webhook 會一併更新或刪除。
在管線中新增和設定 Kubernetes 工作。	在 Docker Hub 存放庫中建立或更新構件時，管線會觸發。然後，它會透過管線將構件部署到 Kubernetes 叢集中的 Docker 主機。
將本機 YAML 定義包含在工作中。	套用至部署工作的 YAML 定義包括 Docker 容器映像。如果需要從專有存放庫下載映像，YAML 檔案必須包含具有 Docker 組態密碼的區段。請參閱使用智慧管線範本前在 Code Stream 中規劃 CICD 原生建置的 CD 部分

在 Docker Hub 存放庫中建立或更新構件時，Docker Hub 中的 webhook 會通知 Code Stream 中的 webhook，而後者會觸發管線。將執行下列動作：

1 Docker Hub 向 webhook 中的 URL 傳送 POST 請求。

2 Code Stream 執行 Docker 觸發器。

3 Docker 觸發器啟動 CD 管線。

4 CD 管線將構件推送至 Docker Hub 存放庫。

5 Code Stream 觸發其 Docker webhook，後者會執行 CD 管線以將構件部署到 Docker 主機。

在此範例中，您在 Code Stream 中建立一個 Docker 端點和一個 Docker webhook，以將應用程式部署到 Kubernetes 開發叢集。步驟包括 Docker 發佈至 Webhook 中的 URL 之裝載的範例程式碼、其使用的 API 程式碼以及具有安全 Token 的驗證碼。

必要條件

- 確認 Code Stream 執行個體中存在持續交付 (CD) 管線。此外，請確認此管線包含部署應用程式的一或多項 Kubernetes 工作。請參閱第 4 章 在 Code Stream 中規劃原生建置、整合及交付程式碼 。
- 確認您可存取現有 Kubernetes 叢集，以便 CD 管線將應用程式部署到該叢集進行開發。

- 確認您是 Code Stream 中的專案的成員。如果您不是管理員，請要求 Code Stream 管理員將您新增為專案成員。請參閱如何在 Code Stream 中新增專案。

程序

1 建立 Docker 登錄端點。

   a 按一下**端點**。

   b 按一下**新增端點**。

   c 開始輸入現有專案的名稱。

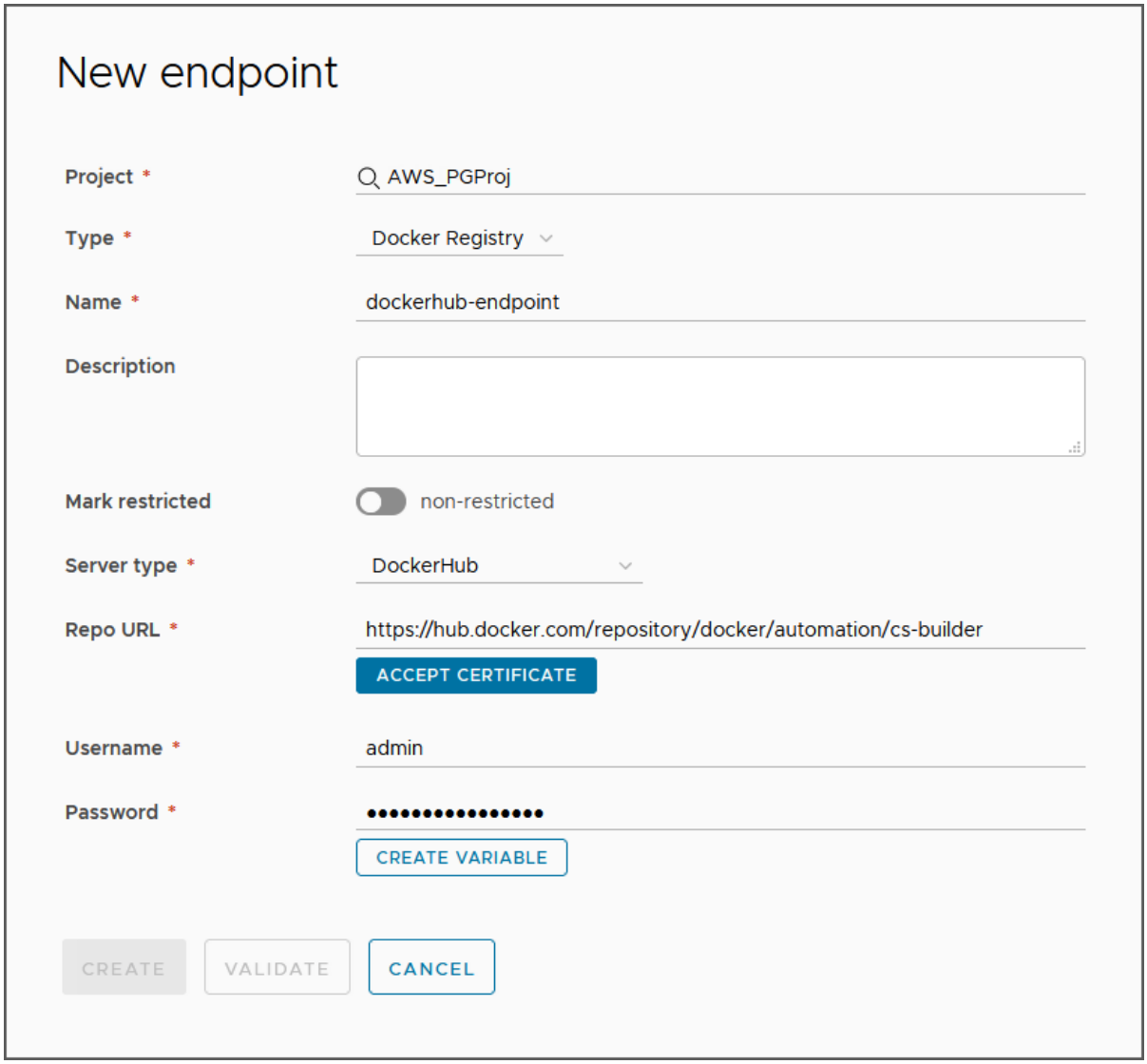
   d 為類型選取 **Docker 登錄**。

   e 輸入相關的名稱。

   f 為伺服器類型選取 **DockerHub**。

g 輸入 Docker Hub 存放庫的 URL。

h 輸入可存取存放庫的名稱和密碼。



2 在 CD 管線中，設定輸入內容以在管線執行時自動插入 Docker 參數。

3 建立 Docker webhook。

a 按一下**觸發器 > Docker**。

b 按一下**新增適用於 Docker 的 Webhook**。

c 選取專案。

d 輸入相關的名稱。

e 選取您的 Docker 登錄端點。

如果端點尚未存在，請按一下**建立端點**並建立端點。

f 選取 webhook 要觸發的插入了 Docker 參數的管線。請參閱步驟 2。

如果管線已設定使用自訂新增的輸入參數，[輸入參數] 清單會顯示參數和值。您可以輸入將透過觸發事件傳遞到管線的輸入參數的值。或者，您可以將這些值保留空白，或使用預設值 (如果已定義)。

如需有關輸入索引標籤上參數的詳細資訊，請參閱手動新增工作前在 Code Stream 中規劃 CICD 原生建置。

g 輸入 API Token。

CSP API Token 會向您驗證與 Code Stream 的外部 API 連線。取得 API Token：

1 按一下**產生 Token**。

2 輸入與使用者名稱和密碼相關聯的電子郵件地址，然後按一下**產生**。

您產生的 Token 有效期為六個月。它也稱為重新整理 Token。

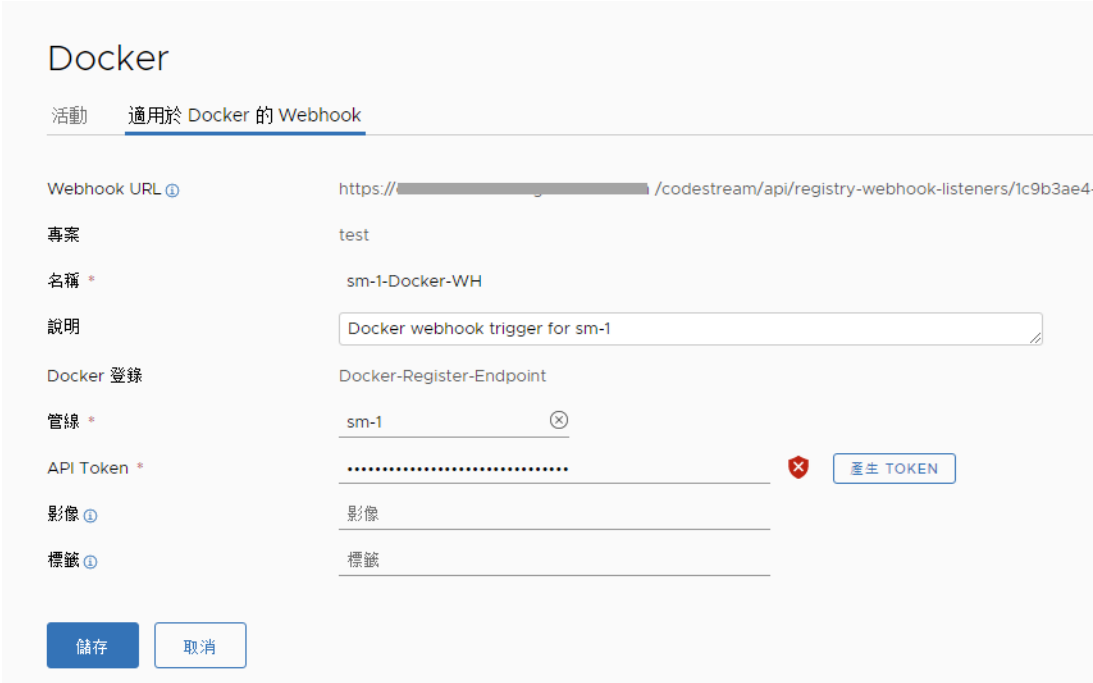
- 若要將 Token 保留為變數供日後使用，請按一下**建立變數**，輸入變數的名稱，然後按一下**儲存**。
- 若要將 Token 保留為文字值供日後使用，請按一下**複製**，然後將 Token 貼到文字檔中以在本機儲存。

您可以選擇建立變數並將 Token 儲存在文字檔中，以供日後使用。

3 按一下**關閉**。

h 輸入建置映像。

i 輸入標籤。



j 按一下**儲存**。

Webhook 卡隨即顯示並啟用 Docker webhook。如果您想要虛擬推送至 Docker Hub 存放庫，而不觸發 Docker webhook 並執行管線，請按一下**停用**。

4 在 CD 管線中，設定 Kubernetes 部署工作。

a 在 Kubernetes 工作內容中，選取您的 Kubernetes 開發叢集。

b 選取**建立**動作。

c 為裝載來源選取**本機定義**。

d 然後，選取本機 YAML 檔案。

例如，Docker Hub 可能會將下列本機 YAML 定義做為裝載發佈至 webhook 中的 URL：

```
{
"callback_url": "https://registry.hub.docker.com/u/svendowideit/testhook/hook/
2141b5bi5i5b02bec211i4eeih0242eg11000a/",
"push_data": {
"images": [
"27d47432a69bca5f2700e4dff7de0388ed65f9d3fb1ec645e2bc24c223dc1cc3",
"51a9c7c1f8bb2fa19bcd09789a34e63f35abb80044bc10196e304f6634cc582c",
"..."
],
"pushed_at": 1.417566161e+09,
"pusher": "trustedbuilder",
"tag": "latest"
},
"repository": {
"comment_count": 0,
"date_created": 1.417494799e+09,
"description": "",
"dockerfile": "#\n# BUILD\u0009\u0009docker build -t svendowideit/apt-cacher .\n#
RUN\u0009\u0009docker run -d -p 3142:3142 -name apt-cacher-run apt-cacher\n#\n# and
then you can run containers with:\n# \u0009\u0009docker run -t -i -rm -e http_proxy
http://192.168.1.2:3142/ debian
bash\n#\nFROM\u0009\u0009ubuntu\n\n\nVOLUME\u0009\u0009[\/var/cache/apt-cacher-ng\]
\nRUN\u0009\u0009apt-get update ; apt-get install -yq apt-cacher-ng\n\nEXPOSE
\u0009\u00093142\nCMD\u0009\u0009chmod 777 /var/cache/apt-cacher-ng ; /etc/init.d/apt-
cacher-ng start ; tail -f /var/log/apt-cacher-ng/*\n",
"full_description": "Docker Hub based automated build from a GitHub repo",
"is_official": false,
"is_private": true,
"is_trusted": true,
"name": "testhook",
"namespace": "svendowideit",
"owner": "svendowideit",
"repo_name": "svendowideit/testhook",
"repo_url": "https://registry.hub.docker.com/u/svendowideit/testhook/",
"star_count": 0,
"status": "Active"
}
}
```

用於在 Docker Hub 中建立 webhook 的 API 採用下列格式：`https://cloud.docker.com/v2/repositories/%3CUSERNAME%3E/%3CREPOSITORY%3E/webhook_pipeline/`

JSON 代碼本文類似於：

```
{
"name": "demo_webhook",
"webhooks": [
{
"name": "demo_webhook",
```

```
"hook_url": "http://www.google.com"
}
]
}
```

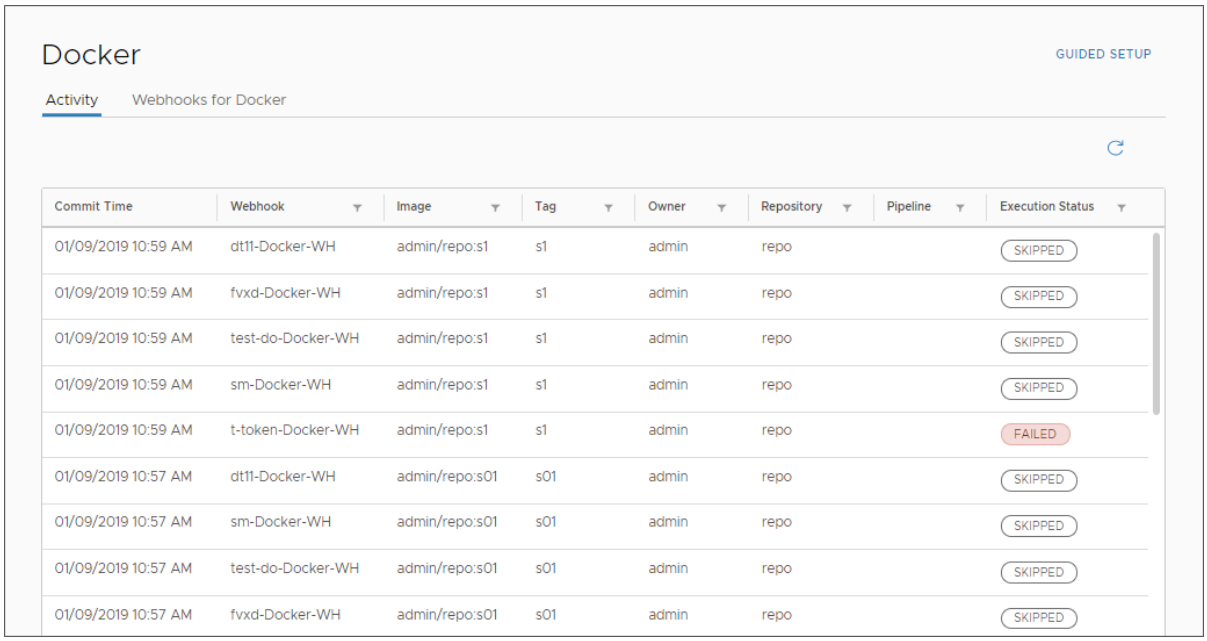
若要從 Docker Hub 伺服器接收事件，在 Code Stream 中建立的 Docker webhook 的驗證配置會將允許清單驗證機制與 webhook 的隨機字串語彙基元搭配使用。它根據安全 Token 篩選事件，您可將安全 Token 附加至 `hook_url`。

Code Stream 可以使用設定的安全 Token 驗證來自 Docker Hub 伺服器的任何請求。例如：

```
hook_url = IP:Port/pipelines/api/docker-hub-webhooks?secureToken = ""
```

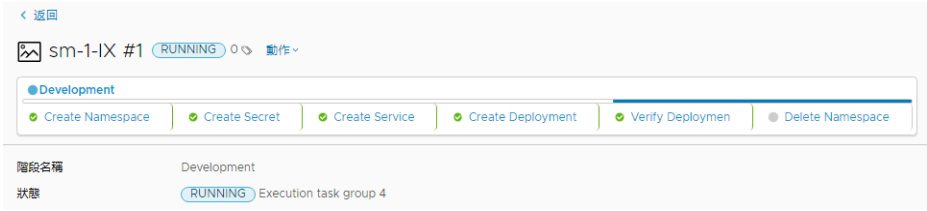
5 在 Docker Hub 存放庫中建立 Docker 構件。或者，更新現有構件。

6 若要確認是否已觸發，並查看 Docker Webhook 上的活動，請按一下**觸發器 > Docker > 活動**。



7 按一下**執行**，並在管線執行時觀察管線。

8 按一下執行階段，並在管線執行時檢視工作。



結果

恭喜您！您可以設定 Docker 觸發器以持續執行 CD 管線。您的管線現在可以將新的和更新的 Docker 構件上傳至 Docker Hub 存放庫。

後續步驟

確認新的或已更新的構件已部署至 Kubernetes 開發叢集中的 Docker 主機。

如何在 Code Stream 中使用 Git 觸發器執行管線

身為 Code Stream 管理員或開發人員，您可以使用 Git 觸發器將 Code Stream 與 Git 生命週期整合。當您在 GitHub、GitLab 或 Bitbucket Enterprise 中進行代碼變更時，事件會透過 Webhook 與 Code Stream 進行通訊並觸發管線。當 Cloud Assembly 和企業版本在同一個網路上可以連線時，Webhook 則適用於 GitLab、GitHub 和 Bitbucket 內部部署企業版本。

在 Code Stream 中新增 Git 的 webhook 時，它還會在 GitHub、GitLab 或 Bitbucket 存放庫中建立 webhook。如果稍後更新或刪除 Webhook，該動作也會更新或刪除 GitHub、GitLab 或 Bitbucket 中的 Webhook。

Webhook 定義必須包含將要監控的存放庫的分支上的 Git 端點。若要建立 Webhook， Code Stream 會使用 Git 端點。如果不存在端點，您可以在新增 webhook 時建立端點。此範例假設在 GitHub 中您有預先定義的 Git 端點。

備註 若要建立 Webhook，Git 端點必須使用私人 Token 進行驗證，不能使用密碼。

透過使用相同的 Git 端點並在 Webhook 組態頁面中為分支名稱提供不同的值，可以為不同的分支建立多個 Webhook。若要為相同 Git 存放庫中的其他分支建立其他 Webhook，無需針對多個分支複製 Git 端點多次。而是在 Webhook 中提供分支名稱，以便重複使用 Git 端點。如果 Git Webhook 中的分支與端點中的分支相同，則無需在 Git Webhook 頁面中提供分支名稱。

此範例顯示如何將 Git 觸發器與 GitHub 存放庫搭配使用，但必要條件包括使用另一個 Git 伺服器類型時所需的準備工作。

必要條件

- 確認您是 Code Stream 中的專案的成員。如果您不是管理員，請要求 Code Stream 管理員將您新增為專案成員。請參閱如何在 Code Stream 中新增專案。
- 確認要監控的 GitHub 分支上存在 Git 端點。請參閱如何整合 Code Stream 與 Git。
- 確認您具有在 Git 存放庫中建立 webhook 的權限。

- 如果在 GitLab 中設定 Webhook，請在 GitLab Enterprise 中變更預設網路設定，以啟用輸出請求並允許建立本機 Webhook。

  備註 僅 GitLab Enterprise 需要進行此變更。這些設定不適用於 GitHub 或 Bitbucket。

  a 以管理員身分登入 GitLab Enterprise 執行個體。

  b 使用 URL (例如，`http://{gitlab-server}/admin/application_settings/network`) 移至網路設定。

  c 展開**輸出請求**，然後按一下：

  - 允許從 Web hook 和服務向本機網路發出請求。
  - 允許從系統 hook 向本機網路發出請求。

- 對於您想要觸發的管線，確認您已將輸入內容設定為在管線執行時插入 Git 參數。

如需輸入參數的相關資訊，請參閱手動新增工作前在 Code Stream 中規劃 CICD 原生建置。

程序

1 在 Code Stream 中，按一下**觸發器** > **Git**。

2 依序按一下**適用於 Git 的 Webhook** 索引標籤和**新增適用於 Git 的 Webhook**。

  a 選取專案。

  b 為 webhook 輸入有意義的名稱和說明。

c 選取針對要監控的分支設定的 Git 端點。

當您建立 webhook 時，webhook 定義將包含目前端點詳細資料。

- 如果稍後在端點中變更 Git 類型、Git 伺服器類型或 Git 存放庫 URL，則 webhook 將無法再觸發管線，因為它會嘗試使用原始端點詳細資料存取 Git 存放庫。您必須刪除 webhook，然後再次建立包含該端點的 webhook。
- 如果您稍後在端點中變更驗證類型、使用者名稱或私人 Token，則 webhook 將繼續運作。
- 如果使用的是 BitBucket 存放庫，則存放庫的 URL 必須採用以下格式之一：`https://api.bitbucket.org/{user}/{repo name}` 或 `http(s)://{bitbucket-enterprise-server}/rest/api/1.0/users/{username}/repos/{repo name}`。

備註 如果先前建立的 webhook 使用的 Git 端點是使用密碼進行基本驗證，則您必須將其刪除並透過使用私人 Token 進行驗證的 Git 端點重新定義 webhook。

請參閱如何整合 Code Stream 與 Git。

d (選擇性) 輸入您想讓 webhook 監控的分支。

如果未指定分支，則 Webhook 會監控為 Git 端點設定的分支。

e (選擇性) 為 webhook 產生密碼 Token。

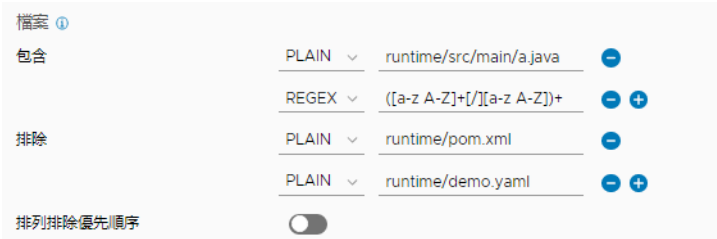
如果使用密碼 Token， Code Stream 將為 Webhook 產生隨機字串 Token。然後，當 Webhook 收到 Git 事件資料時，它會使用密碼 Token 傳送資料。 Code Stream 會使用資訊來判定呼叫是否來自預期來源，例如設定的 GitHub 執行個體、存放庫和分支。密碼 Token 提供一層額外的安全性，用來確認 Git 事件資料來自於正確的來源。

f (選擇性) 提供檔案包含或排除做為觸發器的條件。

- 檔案包含。如果認可中的任何檔案與包含路徑或 Regex 中指定的檔案相符，則認可將觸發管線。透過指定規則運算式， Code Stream 僅在變更集中的檔案名稱與提供的運算式相符時觸發管線。為單一存放庫中的多個管線設定觸發器時，規則運算式篩選器很有用。
- 檔案排除。如果認可中的所有檔案與排除路徑或 Regex 中指定的檔案相符，則不會觸發管線。
- 排除優先。開啟 [排除優先] 時，可確保即使認可中的任何檔案與排除路徑或 Regex 中指定的檔案相符，也不會觸發管線。預設設定為關閉。

如果條件同時滿足檔案包含和檔案排除，則管線不會觸發。

在下列範例中，檔案包含和檔案排除都是觸發器的條件。



- 對於檔案包含，認可對 `runtime/src/main/a.java` 或任何 Java 檔案的任何變更都將觸發事件組態中設定的管線。
- 對於檔案排除，僅認可這兩個檔案中的變更不會觸發事件組態中設定的管線。

g 對於 Git 事件，選取**推送**或**提取**請求。

h 輸入 API Token。

CSP API Token 會向您驗證與 Code Stream 的外部 API 連線。取得 API Token：

1 按一下**產生 Token**。

2 輸入與使用者名稱和密碼相關聯的電子郵件地址，然後按一下**產生**。

您產生的 Token 有效期為六個月。它也稱為重新整理 Token。

- 若要將 Token 保留為變數供日後使用，請按一下**建立變數**，輸入變數的名稱，然後按一下**儲存**。
- 若要將 Token 保留為文字值供日後使用，請按一下**複製**，然後將 Token 貼到文字檔中以在本機儲存。

您可以選擇建立變數並將 Token 儲存在文字檔中，以供日後使用。

3 按一下**關閉**。

i 選取 webhook 要觸發的管線。

如果管線包括自訂新增的輸入參數，[輸入參數] 清單會顯示參數和值。您可以輸入透過觸發事件傳遞到管線的輸入參數的值。或者，您可以將這些值保留空白，或使用預設值 (如果已定義)。

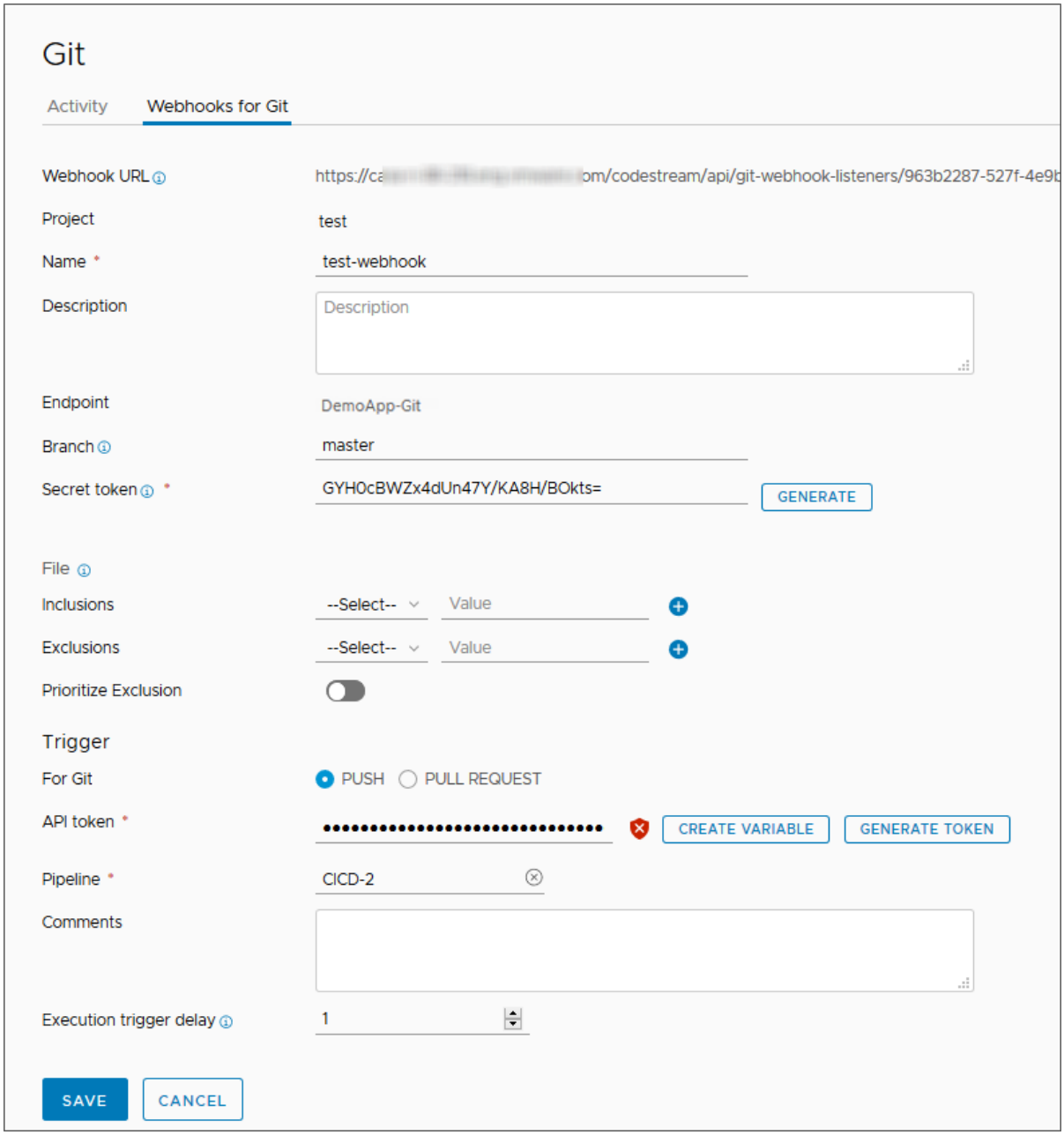
如需 Git 觸發器的自動插入輸入參數的相關資訊，請參閱必要條件。

j 按一下**建立**。

webhook 會顯示為新卡。

3 按一下 Webhook 卡。

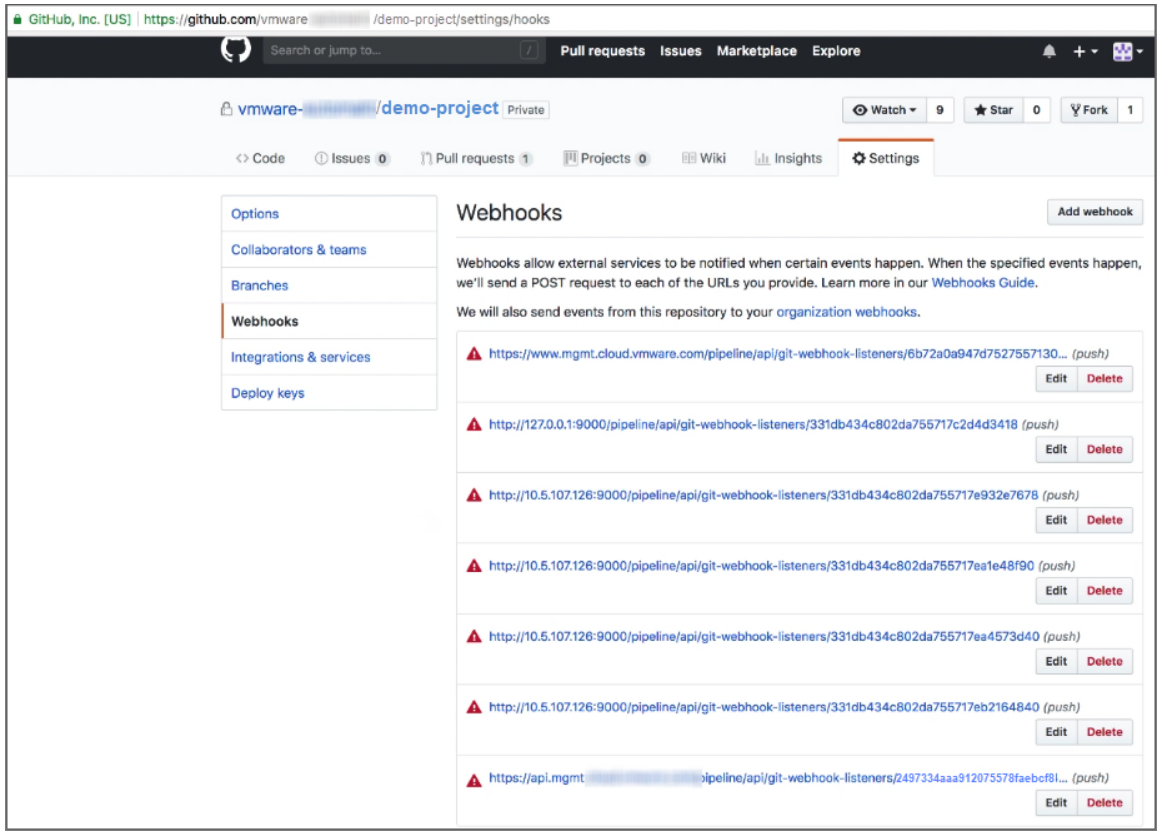
重新顯示 Webhook 資料表單時，您會看到 Webhook URL 已新增至表單的頂端。Git Webhook 透過該 Webhook URL 連線至 GitHub 存放庫。



4 在新的瀏覽器視窗中，開啟透過 Webhook 連線的 GitHub 存放庫。

a 若要查看在 Code Stream 中新增的 Webhook，請按一下**設定**索引標籤，然後選取 **Webhook**。

在 Webhook 清單的底端，您會看到相同的 Webhook URL。



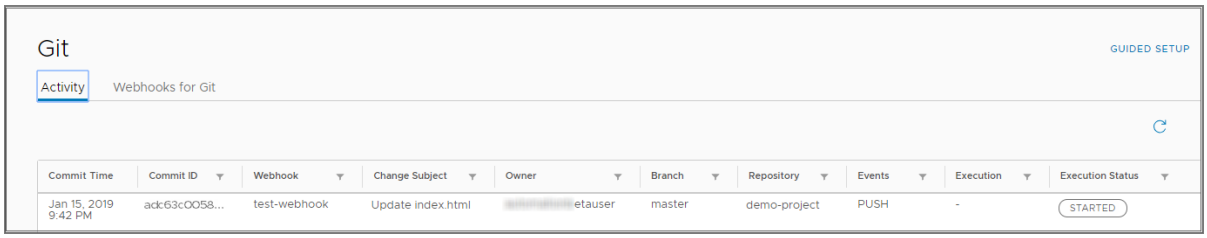
b 若要變更代碼，請按一下**代碼**索引標籤，然後在分支上選取檔案。編輯檔案後，請認可變更。

c 若要確認 Webhook URL 是否正在工作，請按一下**設定**索引標籤，然後再次選取 **Webhook**。

在 Webhook 清單的底端，Webhook URL 旁會顯示一個綠色打勾標記。

5 返回至 Code Stream 以檢視 Git Webhook 上的活動。按一下**觸發器 > Git > 活動**。

在執行狀態下，確認管線執行已啟動。



6 按一下**執行**，並在管線執行時追蹤管線。

若要觀察管線執行，可以按 [重新整理]。

結果

恭喜您！您已成功使用適用於 Git 的觸發器！

如何在 Code Stream 中使用 Gerrit 觸發器執行管線

身為 Code Stream 管理員或開發人員，您可以使用 Gerrit 觸發器將 Code Stream 與 Gerrit 代碼檢閱生命週期整合。當您在 Gerrit 專案上建立修補程式集、發佈草稿、合併程式碼變更或直接在 Git 分支上推送變更時，事件會觸發管線執行。

新增 Gerrit 觸發器時，請選取 Gerrit 接聽程式、Gerrit 伺服器上的 Gerrit 專案，並設定 Gerrit 事件。在此範例中，您先設定 Gerrit 接聽程式，然後在三個不同管線上有兩個事件的 Gerrit 觸發器中使用該接聽程式。

必要條件

- 確認您是 Code Stream 中的專案的成員。如果您不是管理員，請要求 Code Stream 管理員將您新增為專案成員。請參閱如何在 Code Stream 中新增專案。
- 確認您已在 Code Stream 中設定 Gerrit 端點。請參閱如何整合 Code Stream 與 Gerrit。
- 確認您知道 Gerrit 發行版本。
- 對於要觸發的管線，請確認已將管線的輸入內容設定為 **Gerrit**，這將允許管線在管線執行時將 Gerrit 參數作為輸入接收。

如需輸入參數的相關資訊，請參閱手動新增工作前在 Code Stream 中規劃 CICD 原生建置。

程序

1 在 Code Stream 中，按一下**觸發器 > Gerrit**。

2 (選擇性) 按一下**接聽程式**索引標籤，然後按一下**新增接聽程式**。

備註 如果已定義您打算用於 Gerrit 觸發器的 Gerrit 接聽程式，請略過此步驟。

a 選取專案。

b 輸入 Gerrit 接聽程式的名稱。

c 選取 Gerrit 端點。

d 輸入 API Token。

CSP API Token 會向您驗證與 Code Stream 的外部 API 連線。取得 API Token：

1 按一下**產生 Token**。

2 輸入與使用者名稱和密碼相關聯的電子郵件地址，然後按一下**產生**。

您產生的 Token 有效期為六個月。它也稱為重新整理 Token。

- 若要將 Token 保留為變數供日後使用，請按一下**建立變數**，輸入變數的名稱，然後按一下**儲存**。
- 若要將 Token 保留為文字值供日後使用，請按一下**複製**，然後將 Token 貼到文字檔中以在本機儲存。

您可以選擇建立變數並將 Token 儲存在文字檔中，以供日後使用。

3 按一下**關閉**。

如果您已建立變數，則 API Token 會顯示使用貨幣繫結輸入的變數名稱。如果您已複製 Token，則 API Token 會顯示已遮罩的 Token。

e 若要驗證 Token 和端點詳細資料，請按一下**驗證**。

您的 Token 將在 90 天後到期。

f 按一下**建立**。

g 在接聽程式卡上，按一下**連線**。

接聽程式會開始監控 Gerrit 伺服器上的所有活動，並接聽該伺服器上任何已啟用的觸發器。若要停止接聽該伺服器上的觸發器，請停用觸發器。

備註 若要更新連線到接聽程式的 Gerrit 端點，必須先中斷接聽程式的連線，然後再更新端點。

- 按一下**設定 > 觸發器 > Gerrit**。
- 按一下**接聽程式**索引標籤。
- 在要更新且已連線到端點的接聽程式上按一下**中斷連線**。

3 按一下**觸發器**索引標籤，然後按一下**新增觸發器**。

4 選取 Gerrit 伺服器上的專案。

5 輸入名稱。

Gerrit 觸發器名稱必須是唯一的。

6 選取已設定的 Gerrit 接聽程式。

透過使用 Gerrit 接聽程式， Code Stream 會提供伺服器上可用的 Gerrit 專案的清單。

7 選取 Gerrit 伺服器上的專案。

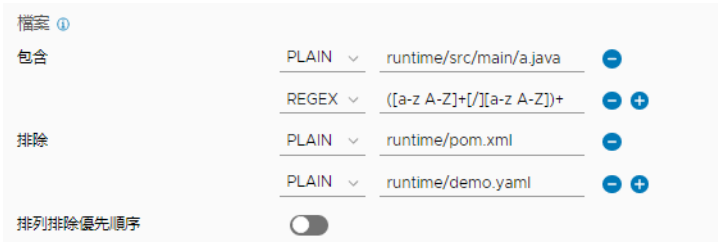
8 輸入 Gerrit 接聽程式將監控的存放庫中的分支。

9 (選擇性) 提供檔案包含或排除做為觸發器的條件。

- 提供檔案包含以觸發管線。當 commit 中的任何檔案與包含路徑或 Regex 指定的檔案相符時，會觸發管線。指定 Regex 時， Code Stream 只會觸發變更集中檔案名稱與提供的運算式相符的管線。為單一存放庫中的多個管線設定觸發器時，規則運算式篩選器很有用。
- 提供檔案排除以使管線不會觸發。當 commit 中的所有檔案與排除路徑或 Regex 中指定的檔案相符時，管線不會觸發。
- **排除優先**開啟時，可確保管線不會觸發。即使認可中的任何檔案與排除路徑或 Regex 中指定的檔案相符，管線也不會觸發。**排除優先**的預設設定已關閉。

如果條件同時滿足檔案包含和檔案排除，則管線不會觸發。

在下列範例中，檔案包含和檔案排除都是觸發器的條件。



- 對於檔案包含，認可對 `runtime/src/main/a.java` 或任何 Java 檔案的任何變更都將觸發事件組態中設定的管線。

- 對於檔案排除，僅認可這兩個檔案中的變更不會觸發事件組態中設定的管線。

10 按一下**新增組態**。

a 對於 Gerrit 事件，選取**已建立修補程式集**、**已發佈草稿**或**已合併變更**。或者，若要繞過 Gerrit 直接推送至 Git，則選取**直接 Git 推送**。

備註 從 Gerrit 發行版本 2.15 開始，不再支援草稿變更和草稿變更集。因此，如果執行的是 Gerrit 發行版本 2.15 或更新版本，則**已發佈草稿**不是受支援的事件。

b 選取將觸發的管線。

如果管線包括自訂新增的輸入參數，[輸入參數] 清單會顯示參數和值。您可以輸入將透過觸發事件傳遞到管線的輸入參數的值。或者，您可以將這些值保留空白，或使用預設值。

備註 如果已定義預設值：

- 針對輸入參數輸入的任何值都將覆寫管線模型中定義的預設值。
- 如果管線模型中的參數值發生變更，觸發器組態中的預設值不會變更。

如需 Gerrit 觸發器的自動插入輸入參數的相關資訊，請參閱必要條件。

c 對於**已建立修補程式集**、**已發佈草稿**或**已合併變更**，某些動作預設會與標籤一起出現。您可以變更標籤或新增註解。然後，當管線執行時，標籤或註解會在**活動**索引標籤中針對管線顯示為**已採取動作**。

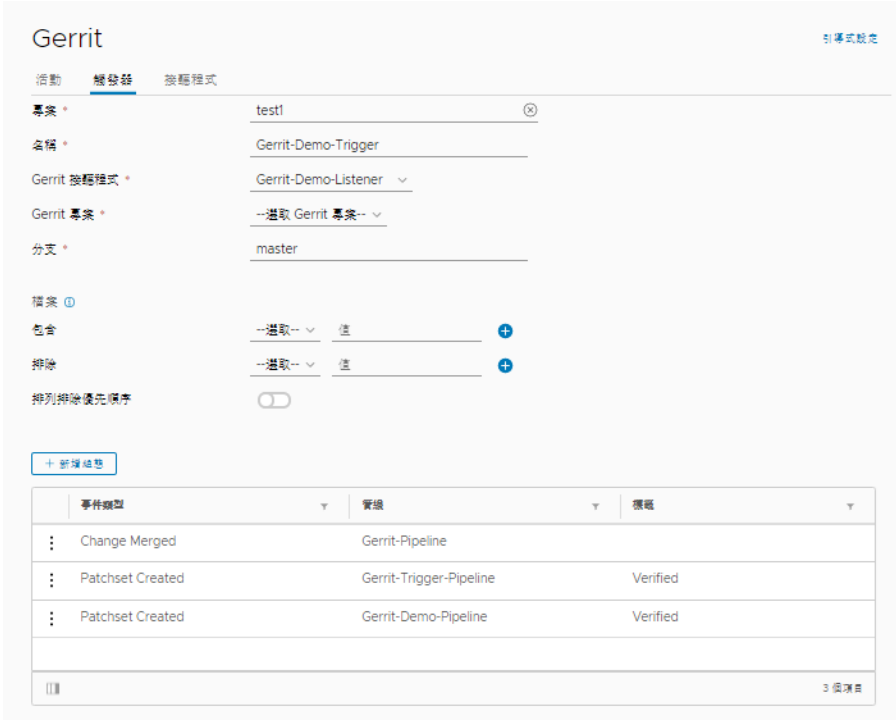
Gerrit 事件組態可讓您使用針對成功註解或失敗註解的變數輸入註解。例如：`${var.success}` 和 `${var.failure}`。

d 按一下**儲存**。

若要在多個管線上新增多個觸發事件，請再次按一下**新增組態**。

在下列範例中，您會看到三個管線的事件：

- 如果 Gerrit 專案中發生**已合併變更**事件，會觸發名為 **Gerrit-Pipeline** 的管線。
- 如果 Gerrit 專案中發生**已建立修補程式集**事件，會觸發名為 **Gerrit-Trigger-Pipeline** 和 **Gerrit-Demo-Pipeline** 的管線。



11 按一下**建立**。

Gerrit 觸發器在**觸發器**索引標籤上顯示為新卡，且預設設定為**已停用**。

12 在觸發器卡上，按一下**啟用**。

啟用觸發器後，它會使用 Gerrit 接聽程式，該接聽程式將開始監控 Gerrit 專案分支上發生的事件。

若要建立具有相同檔案包含條件或檔案排除條件但存放庫不同於建立觸發器時所包括的存放庫的觸發器，請在觸發器卡上按一下**動作 > 複製**。然後，在複製的觸發器上，按一下**開啟**，然後變更參數。

結果

恭喜您！您已成功設定在三個不同管線上有兩個事件的 Gerrit 觸發器。

後續步驟

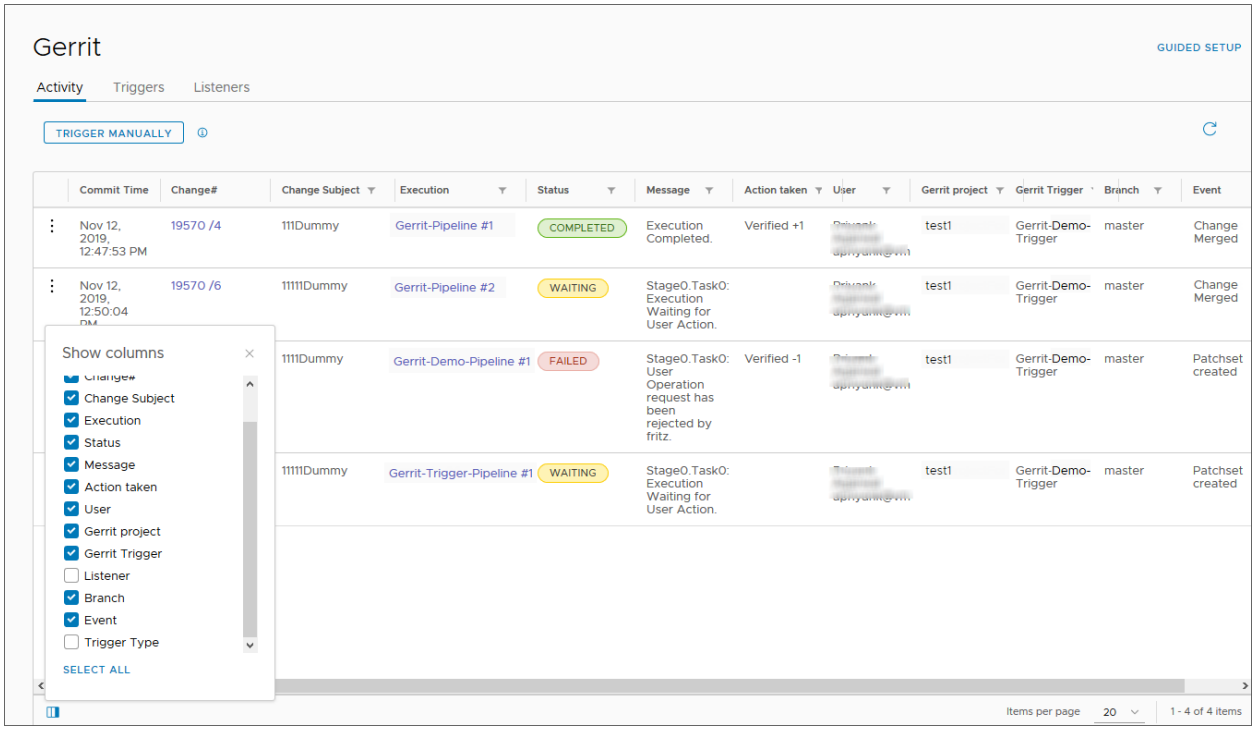
認可 Gerrit 專案中的代碼變更之後，在 Code Stream 的**活動**索引標籤中觀察 Gerrit 事件。請確認活動清單是否包含與觸發器組態中的每個管線執行對應的項目。

發生事件時，僅 Gerrit 觸發器中與特定事件類型相關的管線可以執行。在此範例中，如果已建立修補程式集，僅會執行 **Gerrit-Trigger-Pipeline** 和 **Gerrit-Demo-Pipeline**。

活動索引標籤上的資料行中的資訊說明了每個 Gerrit 觸發器事件。可以按一下資料表下方顯示的資料行圖示來選取顯示的資料行。

- 如果觸發器是直接 Git 推送，則**變更主題**和**執行**資料行為空白。
- **Gerrit 觸發器**資料行顯示已建立事件的觸發器。
- **接聽程式**資料行預設為關閉狀態。選取該資料行時，資料行將顯示接收事件的 Gerrit 接聽程式。單一接聽程式可能會顯示為與多個觸發器相關聯。

- **觸發器類型**資料行預設為關閉狀態。選取該資料行時，資料行將觸發器類型顯示為 [自動] 或 [手動]。
- 其他資料行包括**認可時間**、**變更編號**、**狀態**、**訊息**、**已採取動作**、**使用者**、**Gerrit 專案**、**分支**和**事件**。



若要控制完成或失敗的管線執行的活動，請按一下 [活動] 畫面上任何項目左側的三個點。

- 如果此管線因管線模型錯誤或其他問題而無法執行，請更正錯誤，然後選取**重新執行**以再次執行管線。
- 如果此管線因網路連線問題或其他問題而無法執行，請選取**繼續**以重新啟動相同的管線執行並節省執行時間。
- 使用**檢視執行**以開啟管線執行視圖。請參閱如何執行管線和查看結果。
- 使用**刪除**從 [活動] 畫面中刪除項目。

如果 Gerrit 事件無法觸發管線，可以按一下**手動觸發**，然後選取 Gerrit 觸發器，輸入 Change-Id，並按一下**執行**。

在 Code Stream 中監控管線

8

身為 Code Stream 管理員或開發人員，您必須深入瞭解 Code Stream 中管線的效能。您需要瞭解管線從開發到測試再到生產，如何有效地發行程式碼。

若要深入瞭解，您可以使用 Code Stream 儀表板監控管線執行的趨勢和結果。您可以使用預設管線儀表板監控單一管線，也可以建立自訂儀表板以監控多個管線。

- 管線度量包括平均時間等統計資料，可在管線儀表板上檢視。
- 若要查看多個管線的度量，請使用自訂儀表板。

本章節討論下列主題：

- Code Stream 中的管線儀表板顯示的內容
- 如何在 Code Stream 中使用自訂儀表板追蹤管線的關鍵效能指標

Code Stream 中的管線儀表板顯示的內容

管線儀表板是已執行的特定管線的結果視圖，例如趨勢、主要失敗和成功變更。當您建立管線時， Code Stream 會建立管線儀表板。

儀表板包含顯示管線執行結果的 Widget。

管線 [執行狀態計數] Widget

可以檢視一段時間內管線的執行總數 (依狀態分組)：已完成、失敗或已取消。若要瞭解管線執行狀態在較長或較短的一段時間內的變更情況，請變更顯示中的持續時間。

管線 [執行統計資料] Widget

管線執行統計資料包括一段時間內管線復原、交付或失敗的平均時間。

下列狀態適用於所有管線執行：

- 已完成
- 失敗
- 等待中
- 執行中
- 已取消

- 已排入佇列
- 未啟動
- 正在復原
- 復原已完成
- 復原失敗
- 已暫停

表 8-1. 測量平均時間

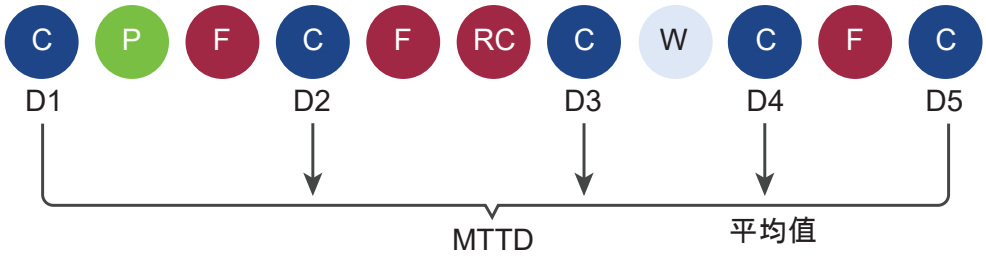
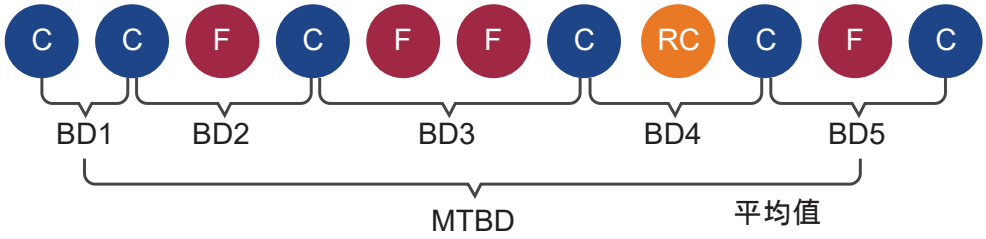
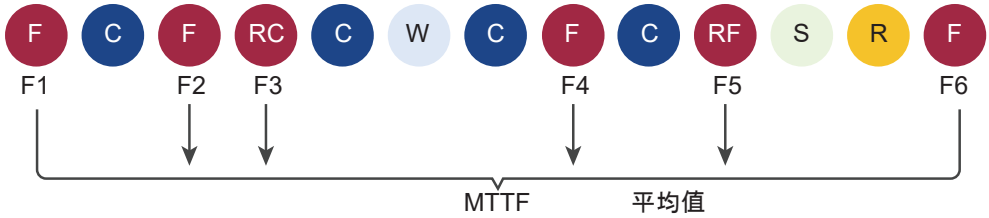
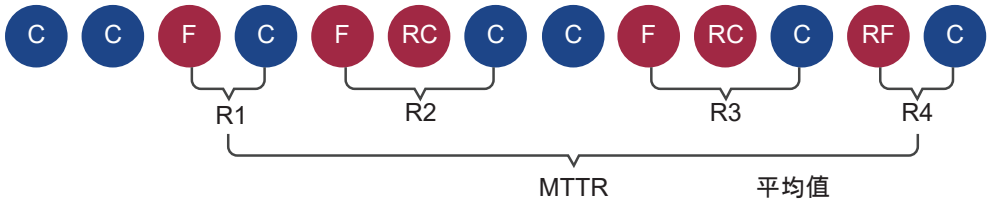
測量什麼...	它表示什麼...
平均 CI	連續整合階段所花費的平均時間，按 CI 工作類型中的時間測量。
平均交付時間 (MTTD)	一段時間內所有已完成執行的平均持續時間。D1、D2 等是交付每個已完成執行的時間量。 
平均交付間隔時間 (MTBD)	一段時間內成功交付的平均間隔時間。兩次連續已完成執行之間經過的時間即為成功交付的間隔時間，例如 BD1、BD2 等。MTBD 會指出生產環境的更新頻率。 

表 8-1. 測量平均時間 (續)

測量什麼...	它表示什麼...
平均故障間隔 (MTTF)	一段時間內以 [失敗]、[復原已完成] 或 [復原失敗] 狀態結束的執行的平均持續時間。F1、F2 等是指執行以 [故障]、[復原已完成] 或 [復原失敗] 狀態結束的時間量。 
平均復原時間 (MTTR)	一段時間內從故障復原的平均時間。從故障復原的時間，即為最終狀態為 [失敗]、[復原已完成] 或 [復原失敗] 的執行與狀態為 [已完成] 的下次立即成功執行之間經過的時間。R1、R2 等是指每次 [失敗] 或 [復原失敗] 執行後復原所需的時間量。 

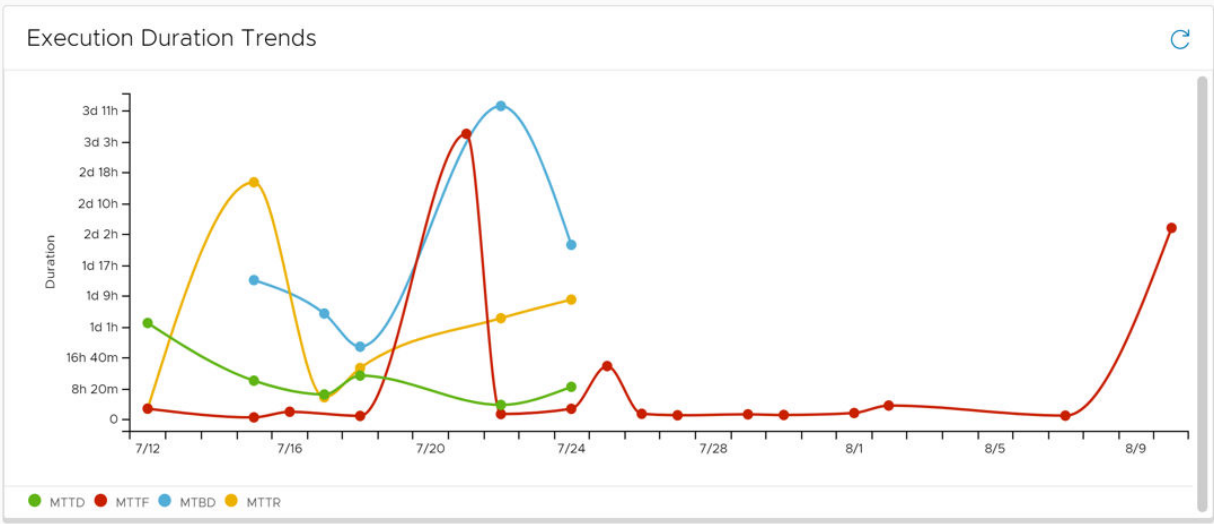
[前幾個失敗的階段] 和 [前幾項失敗的工作] Widget

這兩個 Widget 顯示了管線中前幾個失敗的階段和工作。每個測量會針對每個管線和專案報告開發環境和開發後環境的失敗次數和百分比，每週或每月平均值。您可以檢視前幾個失敗，以便對發行自動化程序中的問題進行疑難排解。

例如，您可以設定特定持續時間 (例如過去 7 天) 的顯示，並記錄此期間內的前幾個失敗的工作。如果您在環境或管線中進行了變更，並再次執行管線，則檢查更長持續時間 (例如過去 14 天) 的前幾個失敗的工作，前幾個失敗的工作可能已變更。根據此結果，您將瞭解在發行自動化程序中所做的變更已改善管線執行的成功率。

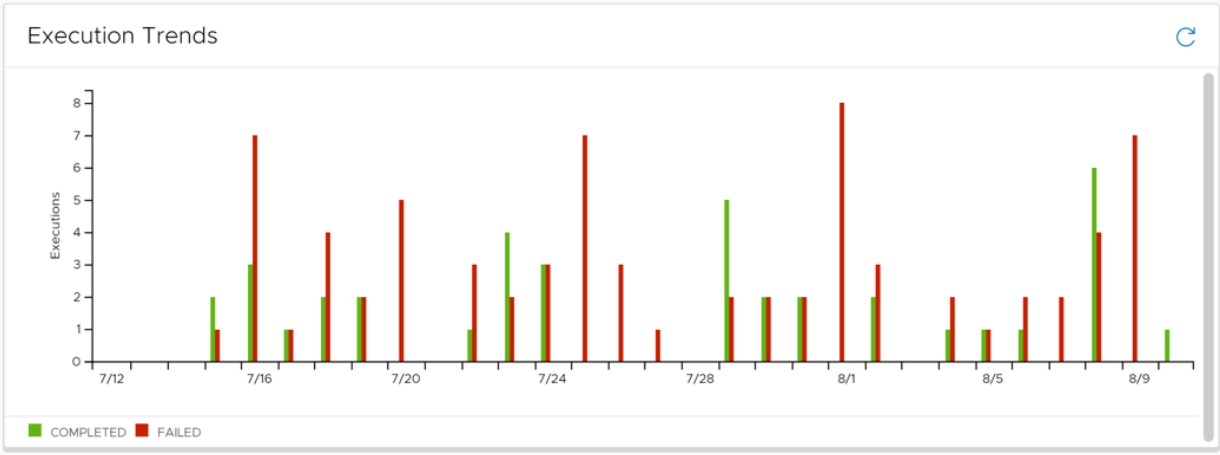
管線 [執行持續時間趨勢] Widget

管線執行持續時間趨勢顯示一段時間內的 MTTD、MTTF、MTBD 和 MTTR。



管線 [執行趨勢] Widget

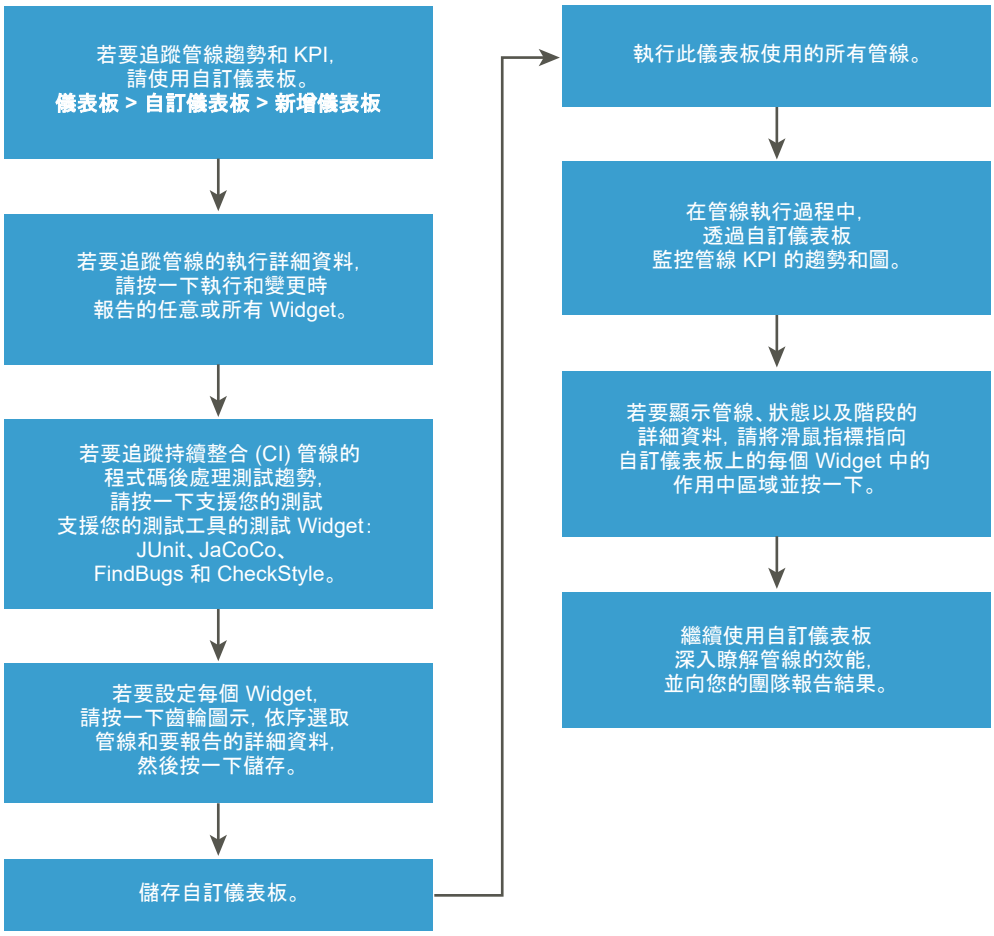
管線執行趨勢顯示一段時間內管線的每日執行總計 (依狀態分組)。除當天之外，大多數每日彙總計數僅顯示已完成執行和失敗執行。



如何在 Code Stream 中使用自訂儀表板追蹤管線的關鍵效能指標

身為 Code Stream 管理員或開發人員，您可以建立自訂儀表板，以顯示要查看的一或多個已執行管線的結果。例如，您可以建立一個適用於整個專案的儀表板，其中包含從多個管線中收集的 KPI 和度量。如果系統報告執行警告或故障，您可以使用儀表板對故障進行疑難排解。

若要使用自訂儀表板追蹤管線的趨勢和關鍵效能指標，您可以將 Widget 新增到儀表板，並將其設定為對管線進行報告。



必要條件

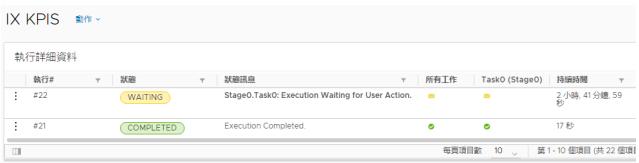
- 確認存在一或多個管線。在使用者介面中，按一下**管線**。
- 對於您想要監控的管線，請確認它們已成功執行。按一下**執行**。

程序

1 若要建立自訂儀表板，按一下**儀表板 > 自訂儀表板 > 新增儀表板**。

2 若要自訂儀表板以便對管線的特定趨勢和關鍵效能指標進行報告，請按一下 Widget。

例如，若要顯示有關管線狀態、階段、工作、執行時間長度和執行者的詳細資料，按一下**執行詳細資料** Widget。或者，對於持續整合 (CI) 管線，您可以透過使用 Widget JUnit、JaCoCo、FindBugs 和 CheckStyle 的 Widget 來追蹤後續處理趨勢。



3 設定您新增的每個 Widget。

a 在 Widget 上，按一下齒輪圖示。

b 選取管線，設定可用的選項，然後選取要顯示的資料行。

c 若要儲存 Widget 組態，請按一下**儲存**。

d 若要儲存自訂儀表板，請按一下**儲存**，然後按一下**關閉**。

4 若要顯示有關管線的詳細資訊，請按一下 Widget 上的作用中區域。

例如，在**執行詳細資料** Widget 中，按一下 [狀態] 資料行中的項目，以顯示有關管線執行的詳細資訊。或者，在**最新成功變更** Widget 上，若要顯示管線階段和工作的摘要，請按一下作用中連結。

結果

恭喜您！您已建立用於監控管線趨勢和 KPI 的自訂儀表板。

後續步驟

繼續監控管線在 Code Stream 中的效能，並與您的管理員和團隊共用結果以繼續改進應用程式發佈程序。

進一步瞭解 Code Stream

9

Code Stream 管理員和開發人員有多種方法可以進一步瞭解 Code Stream 及其功能。

您可以使用本說明文件瞭解有關管線及其執行、如何新增端點、如何新增專案等的詳細資訊。

瞭解角色提供的權限。瞭解如何使用受限制的資源和申請管線核准。請參閱如何在 Code Stream 中管理使用者存取和核准。

瞭解在探索特定工作或元件在管線、執行或端點中的位置時的搜尋值。

本章節討論下列主題：

- 什麼是 Code Stream 中的搜尋
- 供 Code Stream 管理員和開發人員使用的更多資源

什麼是 Code Stream 中的搜尋

您可以使用搜尋來尋找特定項目或其他元件所在的位置。例如，您可能想要搜尋已啟用或已停用的管線，因為管線停用後將無法執行。

我可以搜尋哪些項目

您可以在以下位置搜尋：

- 專案
- 端點
- 管線
- 執行
- 管線儀表板、自訂儀表板
- Gerrit 觸發器和伺服器
- Git Webhook
- Docker Webhook

您可以在以下位置執行以資料行為基礎的篩選器搜尋：

- 使用者作業
- 變數

- Gerrit、Git 和 Docker 觸發器活動

可以在每個觸發器的**活動**頁面中執行以網格為基礎的篩選器搜尋。

搜尋的運作方式

搜尋準則因所在的頁面而有所不同。每個頁面具有不同的搜尋準則。

搜尋位置	用於搜尋的準則
管線儀表板	專案、名稱、說明、標籤和連結
自訂儀表板	專案、名稱、說明、連結 (儀表板上項目的 UUID)
執行	名稱、註解、原因、標籤、索引、狀態、專案、顯示、執行人員、由我執行、連結 (執行的 UUID)，以及輸入參數、輸出參數或使用下列格式的狀態訊息：`<key>:<value>`
管線	名稱、說明、狀態、標籤、建立者、由我建立、更新者、由我更新、專案
專案	名稱、說明
端點	名稱、說明、類型、更新者和專案
Gerrit 觸發器	名稱、狀態、專案
Gerrit 伺服器	名稱、伺服器 URL、專案
Git Webhook	名稱、伺服器類型、存放庫、分支、專案

其中：

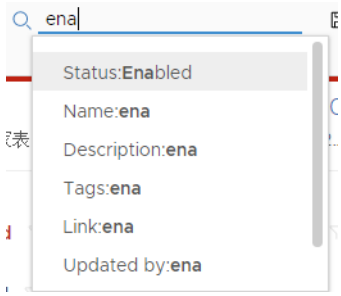
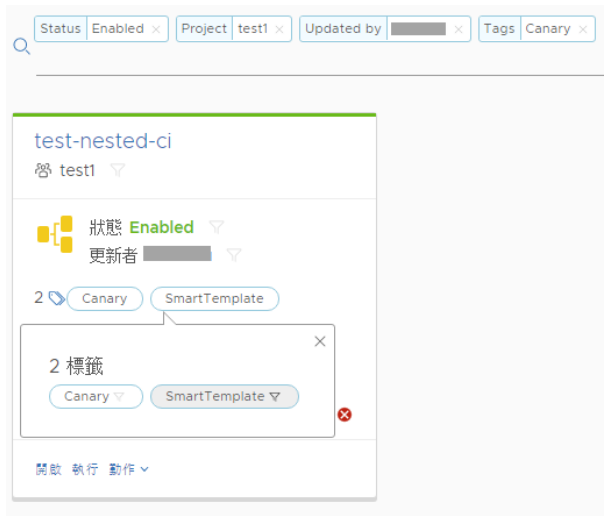
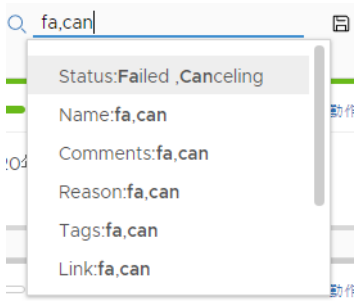
- 連結是管線、執行或儀表板上的 Widget 的 UUID。
- 輸入參數、輸出參數和狀態訊息標記法及範例包括：
  - 標記法：`input.<inputKey>:<inputValue>`

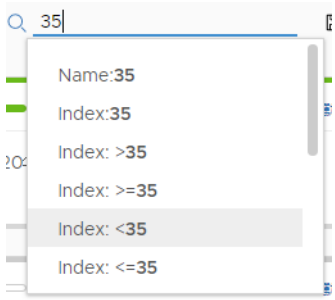
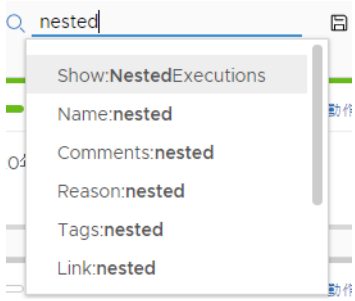
    範例：**input.GERRIT_CHANGE_OWNER_EMAIL:joe_user**
  - 標記法：`output.<outputKey>:<outputValue>`

    範例：**output.BuildNo:29**
  - 標記法：`statusMessage:<value>`

    範例：**statusMessage:Execution failed**
- 狀態取決於搜尋頁面。
  - 對於執行，可能的值包括：已完成、失敗、復原失敗或已取消。
  - 對於管線，可能的狀態值包括：已啟用、已停用或已發佈。
  - 對於觸發器，可能的狀態值包括：已啟用或已停用。
- 由我執行、由我建立或由我更新中的我是指已登入的使用者。

搜尋顯示在每個有效頁面的右上方。開始在搜尋方塊中輸入時， Code Stream 會知道頁面的內容，並建議搜尋選項。

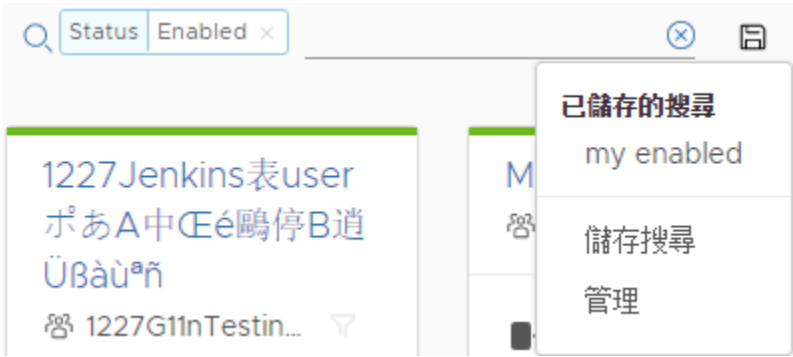
您可以使用的搜尋方法	如何輸入
輸入您的搜尋參數的一部分。 例如，若要新增一個狀態篩選器，其中會列出所有已啟用的管線，請輸入 **ena**。	
若要減少找到的項目數，請新增篩選器。 例如，輸入 **Tes** 可新增名稱篩選器。篩選器做為 AND 條件與現有的 **Status:disabled** 篩選器搭配使用，以僅顯示已停用且名稱包含 **Tes** 的管線。 新增其他篩選器時，會顯示其餘選項：**名稱**、**說明**、**標籤**、**連結**、**專案**和**更新者**。	
若要減少顯示的項目數，請按一下管線或管線執行的內容上的篩選器圖示。 ■ 對於管線，**狀態**、**標籤**、**專案**和**更新者**分別有一個篩選器圖示。 ■ 對於執行，**標籤**、**執行者**和**狀態訊息**分別有一個篩選器圖示。 例如，在管線卡中，按一下圖示以新增 SmartTemplate 標籤的篩選器至現有篩選器：**Status:Enabled**、**Project:test**、**Updated by:user** 和 **Tags:Canary**。	
使用逗號分隔符號以包含處於兩種執行狀態的所有項目。 例如，輸入 **fa,can** 可以建立做為 OR 條件工作的狀態篩選器，以列出所有失敗或已取消的執行。	

您可以使用的搜尋方法	如何輸入
輸入一個數字，以包含索引範圍內的所有項目。 例如，輸入 **35** 並選取 **<** 可以列出索引編號小於 35 的所有執行。	
建立模型為工作的管線將成為巢狀執行，依預設不會從所有執行列出。 若要顯示巢狀執行，請輸入 **nested** 並選取 **Show** 篩選器。	

如何儲存我的最愛搜尋

透過按一下搜尋區域旁的磁碟圖示，您可以儲存我的最愛搜尋，以在每個頁面上使用。

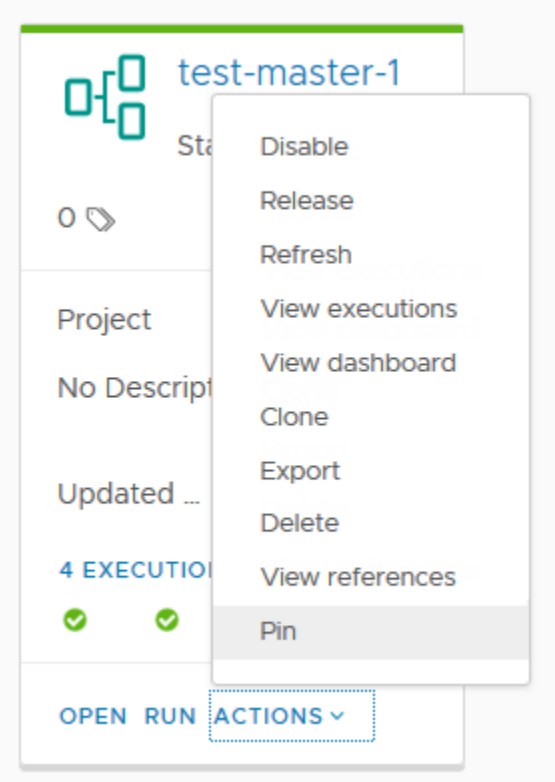
- 藉由輸入搜尋參數，然後按一下圖示為搜尋指定名稱 (例如 **my enabled**)，儲存搜尋。
- 儲存搜尋後，按一下圖示即可存取該搜尋。您也可以選取**管理**來重新命名和刪除搜尋，或在已儲存的搜尋清單中移動搜尋。



搜尋與使用者名稱相關聯，並且僅在搜尋適用的頁面中顯示。例如，如果您在管線頁面中儲存了適用於 **Status:enabled** 且名為 **my enabled** 的搜尋，則即使 **Status:enabled** 搜尋對觸發器有效，**my enabled** 搜尋在 Gerrit 觸發器頁面中也無法使用。

是否可以儲存我的最愛管線

如果您有最愛的管線或儀表板，則可以將其釘選，使其永遠會顯示在管線頁面或儀表板頁面的頂端。在管線卡上，按一下**動作 > 釘選**。



供 Code Stream 管理員和開發人員使用的更多資源

身為 Code Stream 管理員或開發人員，您可以進一步瞭解 Code Stream。

表 9-1. 供管理員使用的更多資源

若要瞭解...	查看下列資源...
管理員可以使用 Code Stream 的其他方式： ■ 設定管線以自動測試和發佈雲端原生應用程式。 ■ 從測試到生產自動執行和測試開發人員原始程式碼。 ■ 為開發人員設定管線，以便在認可對主要分支的變更之前對變更進行測試。 ■ 追蹤關鍵管線度量。	Code Stream ■ vRealize Automation 說明文件 ■ vRealize Automation 產品網站 VMware 實作 ■ 使用 vRealize Automation 社群。 ■ 使用 VMware 學習區域。 ■ 搜尋 VMware 部落格。 ■ 嘗試 VMware 實作實驗室。

表 9-2. 供開發人員使用的更多資源

若要瞭解...	查看下列資源...
開發人員可以使用 Code Stream 的其他方式： ■ 使用公用和私人登錄映像來建置適合新應用程式或服務的環境。 ■ 設定開發環境，以便您可以透過最新的穩定建置建立分支。 ■ 使用最新的程式碼變更和構件更新開發環境。 ■ 對照其他相依服務的最新穩定建置測試未認可的程式碼變更。 ■ 當對主要 CICD 管線認可的變更中斷其他服務時，會收到通知。	Code Stream ■ vRealize Automation 說明文件 ■ vRealize Automation 產品網站 VMware 實作 ■ 使用 vRealize Automation 社群。 ■ 使用 VMware 學習區域。 ■ 搜尋 VMware 部落格。 ■ 嘗試 VMware 實作實驗室。