使用者指南

AWS CodePipeline

API 版本 2015-07-09

AWS CodePipeline: 使用者指南

Table of Contents

什麼是 AWS CodePipeline？

AWS CodePipeline 是一種持續交付的服務，讓您能夠將發行軟體所需的步驟模型化、視覺化和自動化。您可以快速建模並設定軟體發行程序的不同階段。 CodePipeline 自動執行持續發行軟體變更所需的步驟。如需的定價資訊 CodePipeline，請參閱定價。

主題

持續交付和持續整合

CodePipeline 是一項持續交付服務，可將軟體建置、測試及部署到生產環境中自動化。

持續交付是自動化發行程序的軟體開發方法。每項軟體變更都會自動建置、測試和部署至生產環境。最終推送到生產環境之前，人員、自動化測試或商業規則可決定何時應該進行最終推送。雖然每項成功軟體變更都可以立即使用持續交付發行到生產環境，但是不需要立即發行所有變更。

持續集成是一種軟件開發實踐，其中團隊的成員使用版本控制系統，並經常將他們的工作集成到相同的位置，例如主分支。每項變更都會經過建置和驗證，盡快偵測整合錯誤。與可自動化整個軟體發行程序直到生產環境的「持續交付」相較之下，持續整合著重於自動建置和測試程式碼。

如需詳細資訊，請參閱實踐持續整合和持續交付AWS：使用 DevOps.

您可以使用 CodePipeline 主控台、AWS Command Line Interface (AWS CLI)、AWS SDK 或這些項目的任何組合來建立和管理管道。

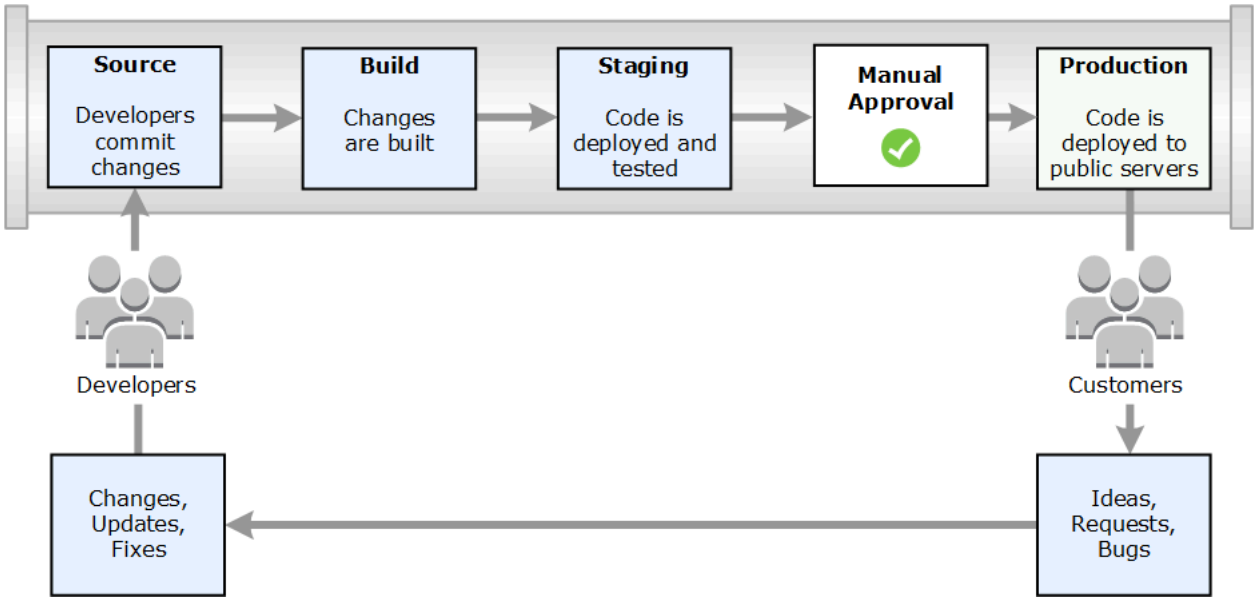
我可以用什麼來做 CodePipeline？

您可以用 CodePipeline 來協助您在雲端中自動建置、測試和部署應用程式。具體而言，您可以：

- 自動化您的發行程序：從您的來源儲存庫開始，透過建置、測試和部署，從端對端 CodePipeline 完全自動化您的發行程序。您可透過在任何階段中 (來源階段除外) 加入手動核准動作，避免透過管道移動時所造成的變更。您可依照您想要的時間、方式、所選的系統，以橫跨單一執行個體或多個執行個體的方式進行發佈。
- 建立一致的發行程序：為每個程式碼變更定義一致的步驟集。 CodePipeline 根據您的條件執行發行版本的每個階段。
- 改善品質同時加速交付：您可自動化發佈程序，以讓您的開發人員逐步進行測試和發佈程式碼，並為您的客戶加速發佈新功能。
- 使用您最愛的工具：您可採納現有來源，在管道中建置並部署工具。如需目前支援的第三AWS 服務方工具的完整清單 CodePipeline，請參閱產品與服務整合 CodePipeline。
- 一目了然地檢視進度：您可以檢視管線的即時狀態、檢查任何警示的詳細資訊、重試失敗的階段或動作、檢視每個階段中最新管道執行中使用的來源修訂版本的詳細資訊，以及手動重新執行任何管道。
- 檢視管道歷史記錄詳細資訊：您可以檢視管道執行詳細資訊，包含啟動與結束時間、執行持續時間與執行 ID。

快速瀏覽一下 CodePipeline

下圖為使用 CodePipeline 的發佈程序範例。



在此範例中，當開發人員遞交來源儲存庫變更時， CodePipeline 會自動偵測該變更。這些變更會進行建置，且若已設定任何測試，這些測試將會執行。在測試完成後，該建置程式碼會部署至開發中伺服器以進行測試。從暫存伺服器 CodePipeline 執行更多測試，例如整合或負載測試。成功完成這些測試後，並在新增至管線的手動核准動作獲得核准後， CodePipeline 將測試和核准的程式碼部署到生產執行個體。

CodePipeline 可以使用 CodeDeploy、AWS Elastic Beanstalk或將應用程式部署到 EC2 執行個體AWS OpsWorks Stacks。 CodePipeline 也可以使用 Amazon ECS 將容器型應用程式部署到服務。開發人員也可以使用隨附的整合點 CodePipeline 來插入其他工具或服務，包括建置服務、測試提供者或其他部署目標或系統。

管道複雜程度可依發佈程序需求調整。

我該如何開始使用 CodePipeline？

若要開始使用 CodePipeline：

1. 通過閱讀CodePipeline 概念 本節了解如何 CodePipeline 工作。
2. CodePipeline 按照中的步驟準備使用開始使用 CodePipeline。
3. CodePipeline 按照CodePipeline 教程自學課程中的步驟進行實驗。
4. 遵循中 CodePipeline 的步驟，在新專案或現有專案中使用在中建立管線 CodePipeline。

CodePipeline 概念

如果您瞭解 AWS CodePipeline 中使用的概念和術語，則建立自動化發行程序的模型和設定就會比較容易。下列是使用 CodePipeline 時需要知道的一些概念。

如需配 DevOps 管的範例，請參閱DevOps 管道示例。

在中使用以下術語 CodePipeline：

主題

- 管道
- 階段
- 動作
- 管道執行
- 階段執行

- 動作執行
- 動作類型
- 轉換
- 成品
- 來源修訂
- 觸發
- Variables

管道

「管道」是一個工作流程建構，說明軟體變更如何進行發行程序。每個管道由一系列的階段組成。

階段

階段是一種邏輯單元，可用來隔離環境，並限制該環境中並行變更的數目。每個階段都包含對應用程式成品執行的動作。原始程式碼就是成品的例子。階段可能是建置原始程式碼和執行測試的建置階段。也可能是程式碼部署到執行時間環境的部署階段。每個階段由一系列的序列或平行動作組成。

動作

動作是一組對應用程式程式碼執行的操作，並設定為在管道中的指定點執行動作。其中可能包括來自程式碼變更的來源動作、將應用程式部署到執行個體的動作等等。例如，部署階段可能包含部署程式碼至 Amazon EC2 或AWS Lambda運算服務的部署動作。

有效的 CodePipeline 動作類型為`source`、`build`、`test`、`deploy`、`approval`、、和`invoke`。如需動作提供者的清單，請參閱中的有效動作類型和提供者 CodePipeline 。

動作可以連續執行，也可以 parallel 執行。如需階段中序列與 parallel 動作的相關`runOrder`資訊，請參閱動作結構需求中的資訊。

管道執行

執行是一組由管道發行的變更。每個管道執行都是唯一的，並且有自己的 ID。一個執行對應於一組變更，例如合併遞交或手動發行最新遞交。兩個執行可以在不同時間發行同一組變更。

一個管道可以同時處理多個執行，但管道階段一次只處理一個執行。為了這樣做，在階段處理執行時會鎖定階段。兩個管道執行不能同時佔據同一階段。等待進入佔用階段的執行被稱為入站執行。輸入執行

仍可能會失敗、取代或手動停止。如需有關輸入執行如何運作的詳細資訊，請參閱入站執行的工作方式。

管道執行按順序周遊管道階段。管道的有效狀態為 InProgress, Stopping、Stopped、Succeeded、Superseded 和 Failed。

如需詳細資訊，請參閱PipelineExecution。

已停止的執行

您可以手動停止管道執行，讓進行中的管線執行不會透過管道繼續執行。如果手動停止，管道執行會在完全停止前顯示 Stopping 狀態。然後會顯示 Stopped 狀態。您可以重試 Stopped 管道執行。

有兩種方式可以停止管道執行：

- 停止並等待
- 停止並捨棄

如需停止執行的使用案例和這些選項的順序詳細資訊，請參閱 管道執行的停止方式。

失敗的執行

如果執行失敗，則會停止且不會完全周遊管道。狀態為 FAILED，且階段會解除鎖定。最近的執行可以趕上，並進入未鎖定的階段來鎖定它。除非失敗的執行已被取代或無法重試，否則您可以重試失敗的執行。

執行模式

若要透過管道傳遞一組最新的變更，較新的執行會通過並取代管道中較早以前的執行。發生這種情況時，較新的執行會取代較舊的執行。一個執行可以由較新的執行在某個點取代，即階段之間的點。「已取代」是預設的執行模式。

在「已取代」模式中，如果執行正在等待進入鎖定的階段，則較近期的執行可能會 catch 並取代它。較新的執行現在會等待階段解除鎖定，而被取代的執行會停止並處於 SUPERSEDED 狀態。當管道執行被取代時，執行會停止，且不會完全周遊管道。在此階段換掉被取代的執行之後，您就無法再重試被取代的執行。其他可用的執行模式為「平行」或「佇列」模式。

如需執行模式與鎖定階段的詳細資訊，請參閱在「已取代」模式下處理執行的方式。

階段執行

階段執行是完成階段內所有動作的程序。如需階段執行如何運作的資訊以及鎖定階段的相關資訊，請參閱在「已取代」模式下處理執行的方式。

階段的有效狀態為InProgressStoppingStopped、Succeeded、和Failed。除非失敗的階段無法重試，否則您可以重試失敗的階段。如需詳細資訊，請參閱StageExecution。

動作執行

「動作執行」是已設定的動作在指定成品上完成運作的過程。可能是輸入成品、輸出成品，或兩者都是。例如，建置動作可能在輸入成品上執行建置命令，例如編譯應用程式原始程式碼。動作執行詳細資訊包括動作執行 ID、相關的管道執行來源觸發，以及動作的輸入和輸出成品。

動作的有效狀態為InProgressAbandoned、Succeeded、或Failed。如需詳細資訊，請參閱ActionExecution。

動作類型

動作類型是預先設定的動作，可在中 CodePipeline選取。動作類型由其擁有者、提供者、版本和類別定義。動作類型提供用於完成管線中動作任務的自訂參數。

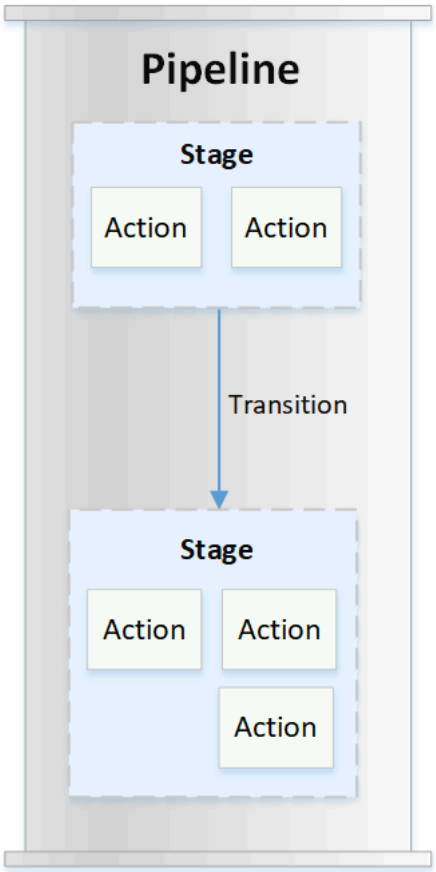
如需有關可根據動作類型整合到管道中的第三方產品和服務的資訊，請參閱與 CodePipeline 動作類型的整合。AWS 服務

如需中支援動作類型之整合模型的資訊 CodePipeline，請參閱整合模型參考。

如需協力廠商提供者如何在中設定和管理動作類型的資訊 CodePipeline，請參閱使用動作類型。

轉換

轉換是管道執行在管道中移至下一個階段的點。您可以停用階段的進入轉換，以防止執行進入該階段，然後啟用轉換以允許執行繼續。當一個以上的執行到達停用的轉換時，只有最新的執行會在啟用轉換時繼續到下一個階段。這表示在停用轉換時，較新的執行會繼續取代等待的執行，然後在啟用轉換之後，繼續的執行就是取而代之的執行。



成品

成品是由管道動作處理的資料集合，例如應用程式原始程式碼、建置的應用程式、相依性、定義檔案、範本等。成品由某些動作產生，並由其他動作取用。在管道中，成品可能是動作處理的一組檔案 (輸入成品)，或已完成的動作所更新的輸出 (輸出成品)。

動作會將輸出傳送至另一個動作，以便使用管線加工品值區進一步處理 CodePipeline 將人工因素複製到人工因素存放區，其中動作會將它們拾取。如需成品的詳細資訊，請參閱 輸入和輸出成品。

來源修訂

進行原始程式碼變更時會建立新版本。來源修訂是觸發管道執行的來源變更版本。執行會處理來源修訂。對於 GitHub 和 CodeCommit 存儲庫，這是提交。對於 S3 儲存貯體或動作，這是物件版本。

您可以使用指定的來源修訂 (例如提交) 來開始管線執行。執行將處理指定的修訂版，並覆蓋用於執行的修訂版本。如需詳細資訊，請參閱 以來源修訂版本取代啟動管線。

觸發

觸發程序是啟動管道的事件。某些觸發程序 (例如手動啟動管道) 可供管線中的所有來源動作提供者使用。某些觸發程序取決於管線的來源提供者。例如， CloudWatch 必須使用 Amazon 的事件資源來設定事件， CloudWatch 這些資源必須將管道 ARN 新增為事件規則中的目標。Amazon E CloudWatch vents 是針對具有 CodeCommit 或 S3 來源動作的管道自動變更偵測的建議觸發器。Webhook 是一種針對協力廠商儲存庫事件設定的觸發器類型。例如，WebHookv2 是一種觸發器類型，允許使用 Git 標籤來啟動與第三方來源提供者（例如 GitHub .com、 GitHub企業伺服器、 GitLab .com、 GitLab 自我管理或 Bitbucket 雲）的管道。在管線組態中，您可以指定觸發程序的篩選器，例如推送或提取要求。您可以篩選 Git 標籤、分支或檔案路徑上的程式碼推送事件。您可以在事件（已打開，更新，關閉），分支或交易路徑上搜索提取請求事件。

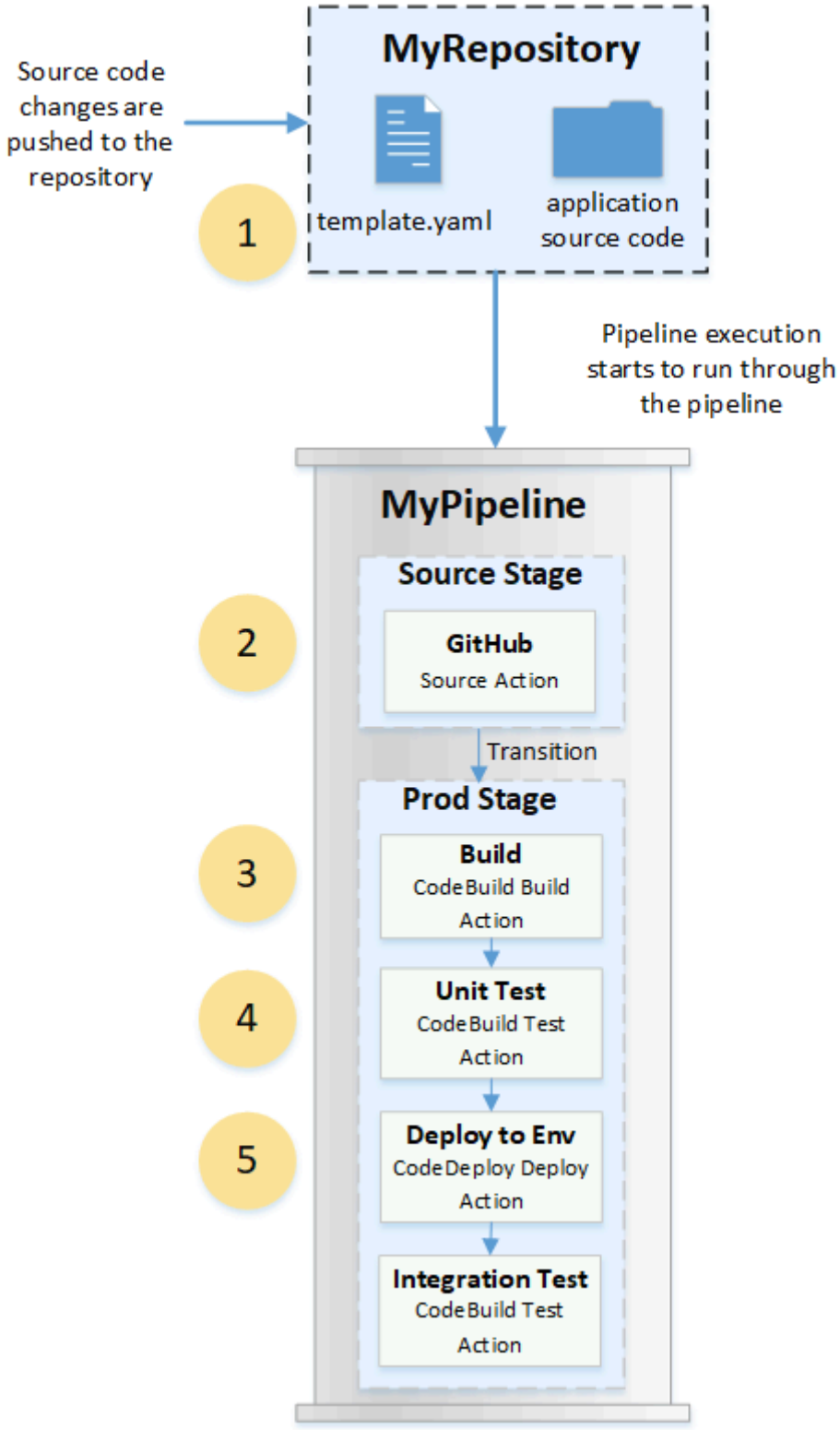
關於觸發條件的詳細資訊，請參閱 在中啟動管道 CodePipeline。如需逐步引導您使用 Git 標籤做為管線觸發程序的教學課程，請參閱教學課程：使用 Git 標籤啟動管道。

Variables

變數是可用來動態設定管線中動作的值。變數可以在管線層級宣告，也可以由管線中的動作發出。變數值會在管線執行時解析，並可在執行歷史記錄中檢視。對於在管線層級宣告的變數，您可以在配管組態中定義預設值，或針對指定的執行覆寫它們。對於動作發出的變數，在動作成功完成之後，即可使用該值。如需更多詳細資訊，請參閱 Variables。

DevOps 管道示例

作為管線的範例，兩階段 DevOps 管線可能有一個名為 Source 的來源階段，而第二個階段稱為 Prod。在此範例中，管道會以最新的變更來更新應用程式，並持續部署最新的結果。在部署最新的應用程式之前，管道會建置並測試 Web 應用程式。在此範例中，一群開發人員已經在名為的 GitHub 儲存庫中為 Web 應用程式設定了基礎結構範本和原始程式碼 MyRepository。



例如，開發人員將修正程式推送到 Web 應用程式的索引頁，並發生下列情況：

1. 應用程式原始程式碼會維護在配置為管線中的 GitHub 來源動作的儲存庫中。當開發人員將認可推送至儲存庫時， CodePipeline 偵測已推送的變更，並從「來源階段」開始執行管線。
2. 來 GitHub 源動作順利完成 (也就是說，最新的變更已下載並儲存至該執行專屬的成品值區)。GitHub 來源動作所產生的輸出成品 (來自存放庫的應用程式檔案) 會用作下一階段動作所要處理的輸入成品。
3. 管道執行從來源階段轉換到生產階段。Prod Stage 中的第一個動作會執行在管線中建立 CodeBuild 並設定為建置動作的建置專案。建置任務提取建置環境映像，並在虛擬容器中建置 Web 應用程式。
4. Prod Stage 中的下一個動作是在中建立 CodeBuild 並設定為管線中的測試動作的單元測試專案。
5. 經過單元測試的程式碼接下來由生產階段中的部署動作處理，該動作會將應用程式部署到生產環境。部署動作成功完成之後，階段中的最後一個動作是在管線中建立 CodeBuild 並設定為測試動作的整合測試專案。測試動作呼叫 Shell 指令碼，在 Web 應用程式上安裝並執行測試工具 (例如連結檢查程式)。成功完成後，輸出是一個已建置的 Web 應用程式和一組測試結果。

開發人員可以將動作新增至管道，以部署或進一步測試已建置並測試過每一項變更的應用程式。

如需詳細資訊，請參閱 管道執行的運作方式。

管道執行的運作方式

本節提供 CodePipeline 處理一組變更的方式概觀。 CodePipeline追蹤手動啟動管線或對原始程式碼進行變更時啟動的每個管線執行。 CodePipeline使用以下執行模式來處理每個執行在管道中進行的方式。

- 取代模式：較近的執行可能會超過較舊的執行。此為預設值。
- QUEED 模式：執行按照排入佇列的順序逐個處理。這需要管線類型 V2。
- 平行模式：在平行模式下，執行會同時執行，彼此獨立執行。在開始或完成之前，執行不會等待其他運行完成。這需要管線類型 V2。

管道執行的啟動方式

您可以在變更原始程式碼或手動啟動管線時開始執行。您也可以透過排程的 Amazon CloudWatch 事件規則觸發執行。例如，當原始程式碼變更推送到設定為管道來源動作的儲存庫時，管道會偵測變更並啟動執行。

> **Note**
>
> 如果管道包含多個來源動作，則會再次執行所有來源動作，即使只偵測到一個來源動作的變更也是一樣。

管道執行的停止方式

若要使用主控台來停止管道執行，您可以在管道視覺化頁面、執行歷程記錄頁面或詳細歷程記錄頁面上選擇 Stop execution (停止執行)。若要使用 CLI 來停止管道執行，請使用 `stop-pipeline-execution` 命令。如需詳細資訊，請參閱 停止管線執行 CodePipeline。

有兩種方式可以停止管道執行：

- 停止並等待：允許所有進行中的動作執行完成，且不會啟動後續動作。管道執行不會繼續進行後續階段。您無法在已處於 `Stopping` 狀態的執行上使用此選項。
- 停止並捨棄：捨棄所有進行中的動作執行且不會完成，而且不會啟動後續動作。管道執行不會繼續進行後續階段。您可以在已處於 `Stopping` 狀態的執行上使用此選項。

> **Note**
>
> 此選項可能會導致工作失敗或工作失序。

每個選項都會產生不同的管道順序和動作執行階段，如下所示。

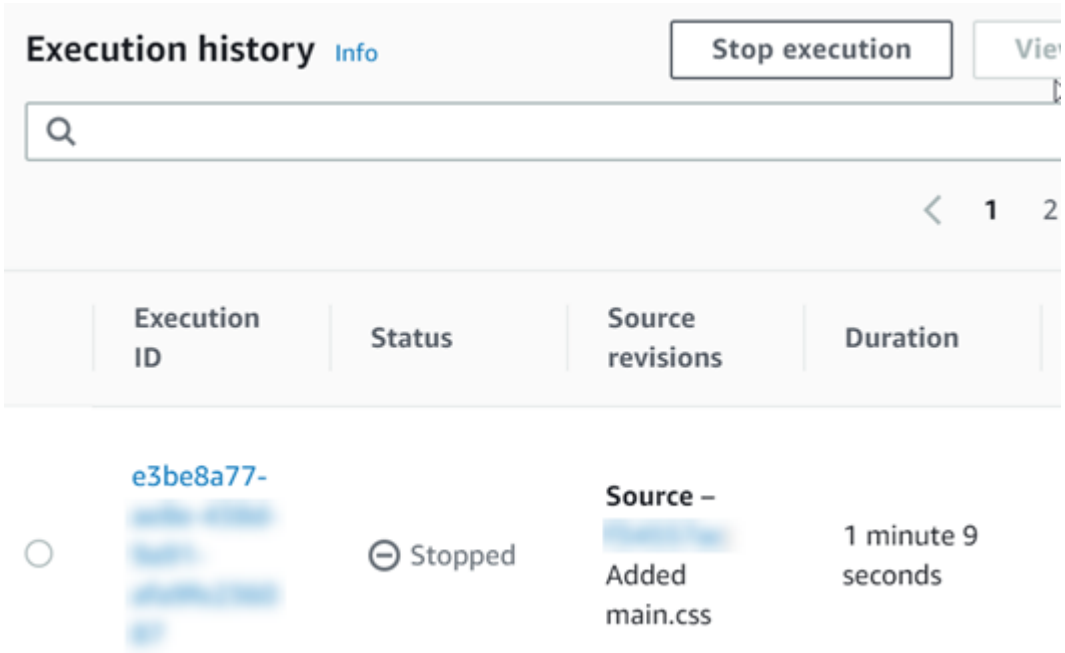
選項 1：停止並等待

當您選擇停止並等待時，選取的執行會繼續進行，直到進行中的動作完成為止。例如，下列管道執行已在建置動作正在進行時停止。

1. 在管道檢視中，會顯示成功訊息橫幅，且建置動作會繼續執行，直到完成為止。管道執行狀態為 Stopping (停止中)。

   在歷程記錄檢視中，進行中動作 (例如建置動作) 的狀態為 In progress (進行中)，直到建置動作完成為止。當動作正在進行時，管道執行狀態為 Stopping (停止中)。
2. 停止程序完成時，執行就會停止。如果成功完成建置動作，其狀態為 Succeeded (成功)，且管道執行會顯示 Stopped (已停止) 狀態。後續動作不會啟動。Retry (重試) 按鈕已啟用。

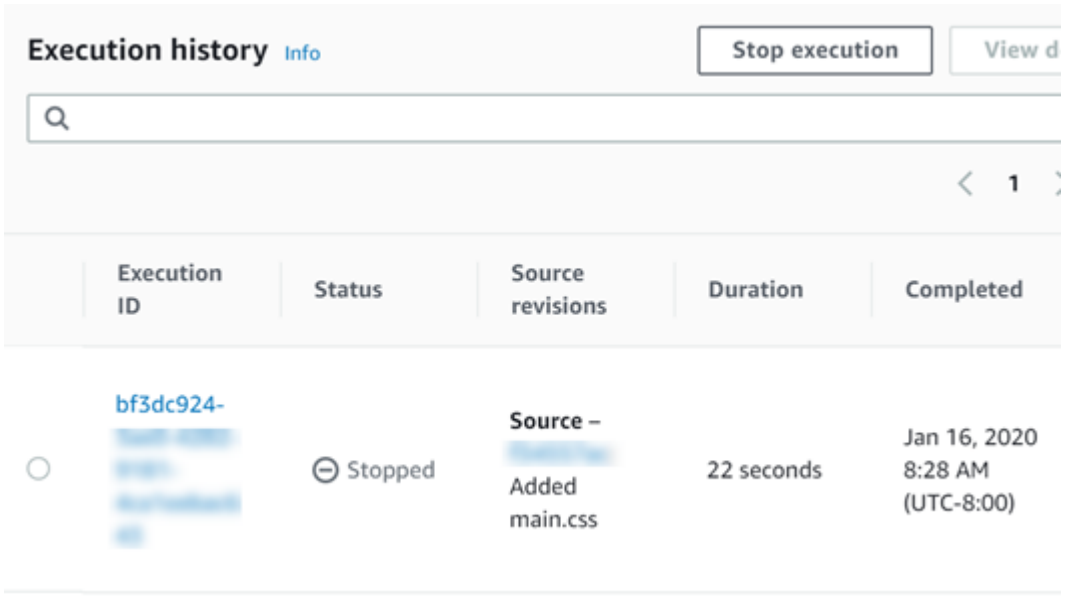
在歷程記錄檢視中，執行中的動作完成後，執行狀態為 Stopped (已停止)。



選項 2：停止並捨棄

當您選擇停止並捨棄時，選取的執行不會等待進行中的動作完成。這些行動已被捨棄。例如，下列管道執行已在建置動作正在進行時停止並捨棄。

1. 在管道檢視中，會顯示成功橫幅訊息，建置動作會顯示 In progress (進行中) 狀態，而管道執行會顯示 Stopping (停止中) 狀態。
2. 管道執行停止後，建置動作會顯示 Abandoned (已捨棄) 狀態，而管道執行會顯示 Stopped (已停止) 狀態。後續動作不會啟動。Retry (重試) 按鈕已啟用。
3. 在歷程記錄檢視中，執行狀態為 Stopped (已停止)。



停止管道執行的使用案例

我們建議您使用「停止並等待」選項來停止管道執行。此選項更安全，因為它可以避免可能的失敗或管道中的 out-of-sequence 任務。中放棄動作時 CodePipeline，動作提供者會繼續與動作相關的任何工作。在 AWS CloudFormation 動作的情況下，管道中的部署動作會被捨棄，但堆疊更新可能會繼續進行，以致更新失敗。

作為可能導致 out-of-sequence 任務的放棄動作範例，如果您透過 S3 部署動作部署大型檔案 (1GB)，而且您選擇在部署已進行時停止和放棄動作，則該動作會放棄在 Amazon S3 中 CodePipeline，但會繼續執行。Amazon S3 沒有遇到任何取消上傳的指示。接下來，如果您使用非常小型的檔案開始新的管道執行，現在有兩個部署正在進行中。由於新執行的檔案大小很小，因此新的部署會在舊的部署仍在上傳時完成。當舊的部署完成時，舊檔案會覆寫新檔案。

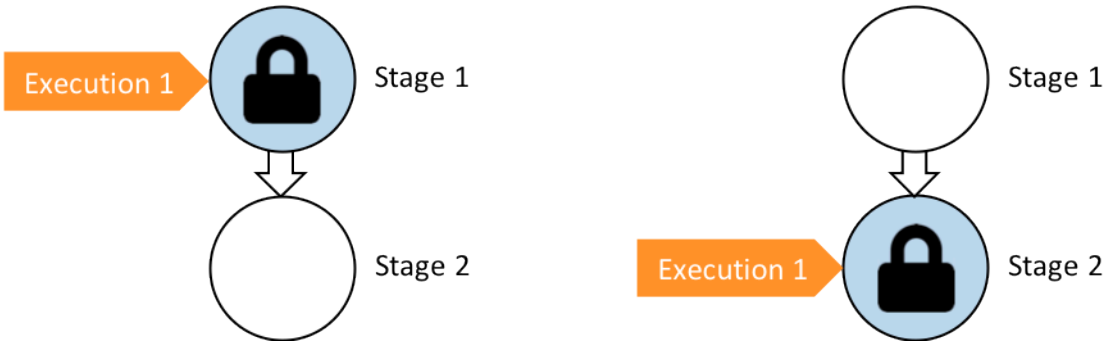
在有自訂動作的情況下，您可以使用「停止並捨棄」選項。例如，您可以放棄具有不需要完成的工作的自訂動作，然後再開始執行錯誤修正。

在「已取代」模式下處理執行的方式

處理執行的預設模式為「已取代」模式。執行是由該執行所取用和處理的一組變更組成。管道可同時處理多個執行。每個執行會分別通過管道。管道按順序處理每個執行，且可能以較晚的執行取代較早的執行。下列規則可用來處理「已取代」模式的管線中的執行。

規則 1：正在處理執行時會鎖定階段

由於每個階段一次只能處理一個執行，因此階段在進行中會鎖定。當執行完成某個階段時，就會轉換至管道的下一個階段。

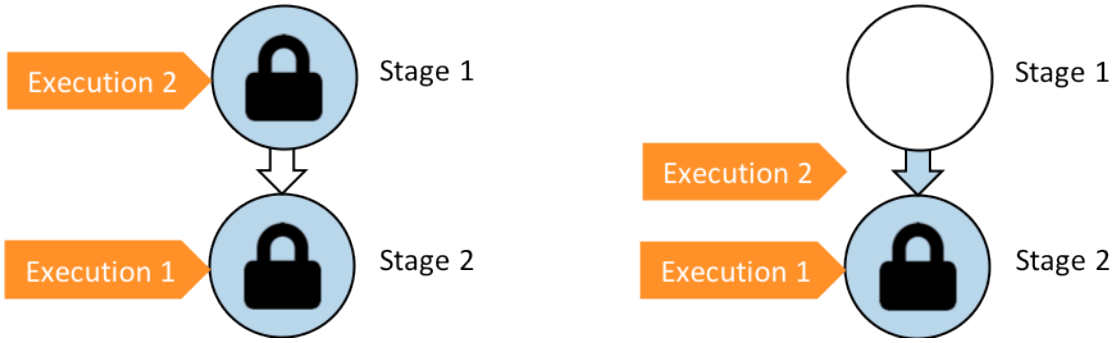


之前：Stage 1 is locked as Execution 1 enters. 之後： Stage 2 is locked as Execution 1 enters.

規則 2：後續執行等待階段解除鎖定

當階段鎖定時，等待中的執行會停留在鎖定的階段前面。必須先成功完成針對階段所設定的所有動作，再將階段視為完成。失敗會釋放階段的鎖定。當執行停止時，執行不會在階段中繼續進行，並已將階段解除鎖定。

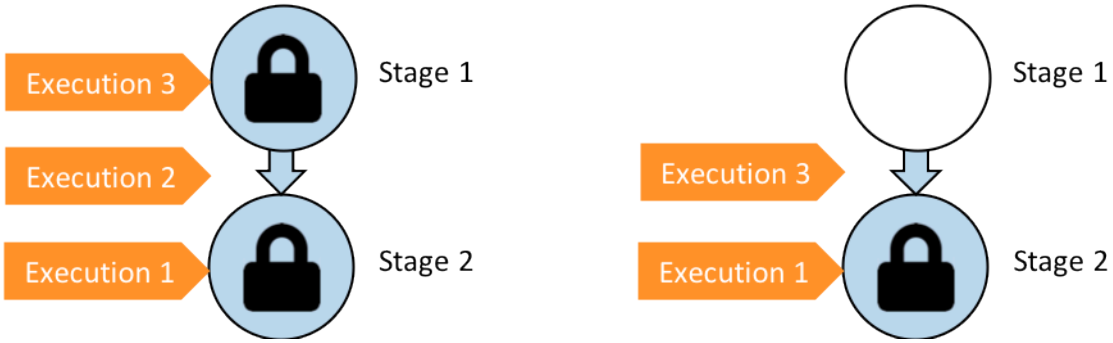
> **Note**
>
> 在您停止執行之前，我們建議您停用階段前面的轉換。如此一來，當階段由於停止執行而解除鎖定時，階段就不會接受後續的管道執行。



之前： Stage 2 is locked as Execution 1 enters. 之後： Execution 2 exits Stage 1 and waits between stages.

規則 3：等待中的執行由較新的執行取代

只會取代階段之間的執行。鎖定的階段會將一個執行扣留在階段前面，以等待階段完成。較新的執行會趕上等待中的執行，並在階段解除鎖定後立即進入下一個階段。被取代的執行不會繼續。在此範例中，「執行 2」在等待鎖定的階段時已被「執行 3」取代。「執行 3」進入下一個階段。



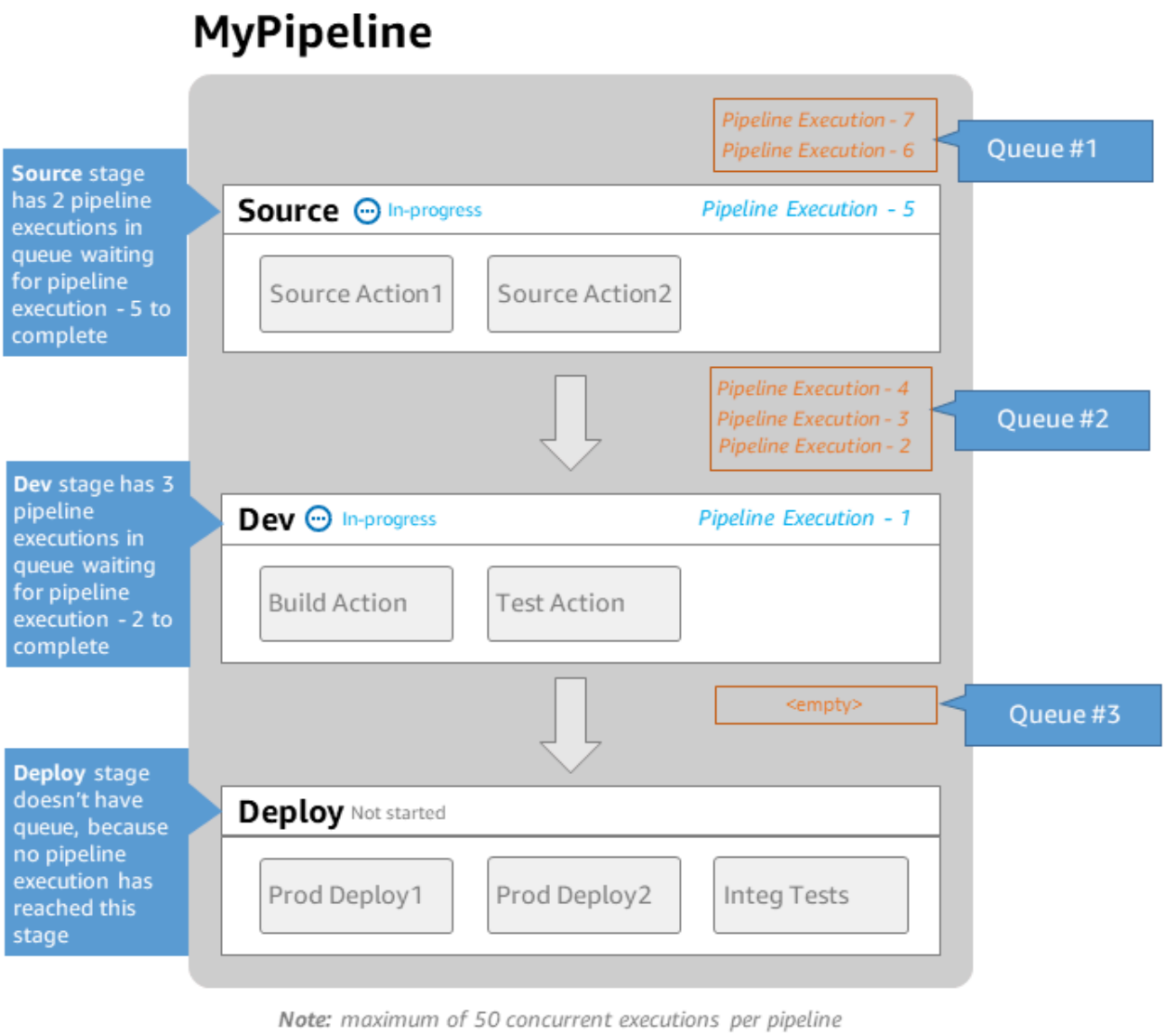
之前：執行2 在階段之間等待，而執行 3 進入階段 1。 之後：執行 3 離開階段 1。執行 3 取代了執行 2。

如何在 QUEED 模式下處理執行

對於處於 QUEUE 模式的管道，在處理執行時，階段會被鎖定；不過，等待的執行不會超過已經啟動的執行。

等待的執行按照它們到達階段的順序在入口點聚集到鎖定階段，形成了等待執行的隊列。使用 QUEUE 模式，您可以在同一管道中擁有多個佇列。當排入佇列的執行項目進入階段時，階段會被鎖定，而且無法進入其他執行項目。此行為與「已取代」模式保持不變。當執行完成階段時，階段會變成解除鎖定，並準備好進行下一次執行。

下圖顯示 QUEED 模式管線處理序執行階段的方式。例如，當「來源」階段處理程序執行 5 時，6 和 7 的執行會形成佇列 #1，然後在階段進入點等待。階段解鎖後，會處理佇列中的下一個執行。



如需執行模式配額的詳細資訊，請參閱AWS CodePipeline 中的配額。

以平行模式處理執行的方式

對於 PALLEAN 模式的管道，執行是彼此獨立的，並且在開始之前不要等待其他執行完成。沒有佇列。若要檢視管線中的 parallel 執行，請使用執行歷史記錄檢視。

在每個功能都有自己的功能分支，並部署到其他使用者未共用的目標的開發環境中使用 PALLEAR 模式。

如需執行模式配額的詳細資訊，請參閱AWS CodePipeline 中的配額。

管理管道流程

管道執行的流程可由以下方式控制：

- 轉換，控制執行進入階段的流程。可以啟用或停用轉換。停用轉移時，管線執行將無法進入階段。等待進入禁用轉換的階段的管道執行稱為入站執行。啟用轉換之後，輸入執行會移至階段並鎖定它。

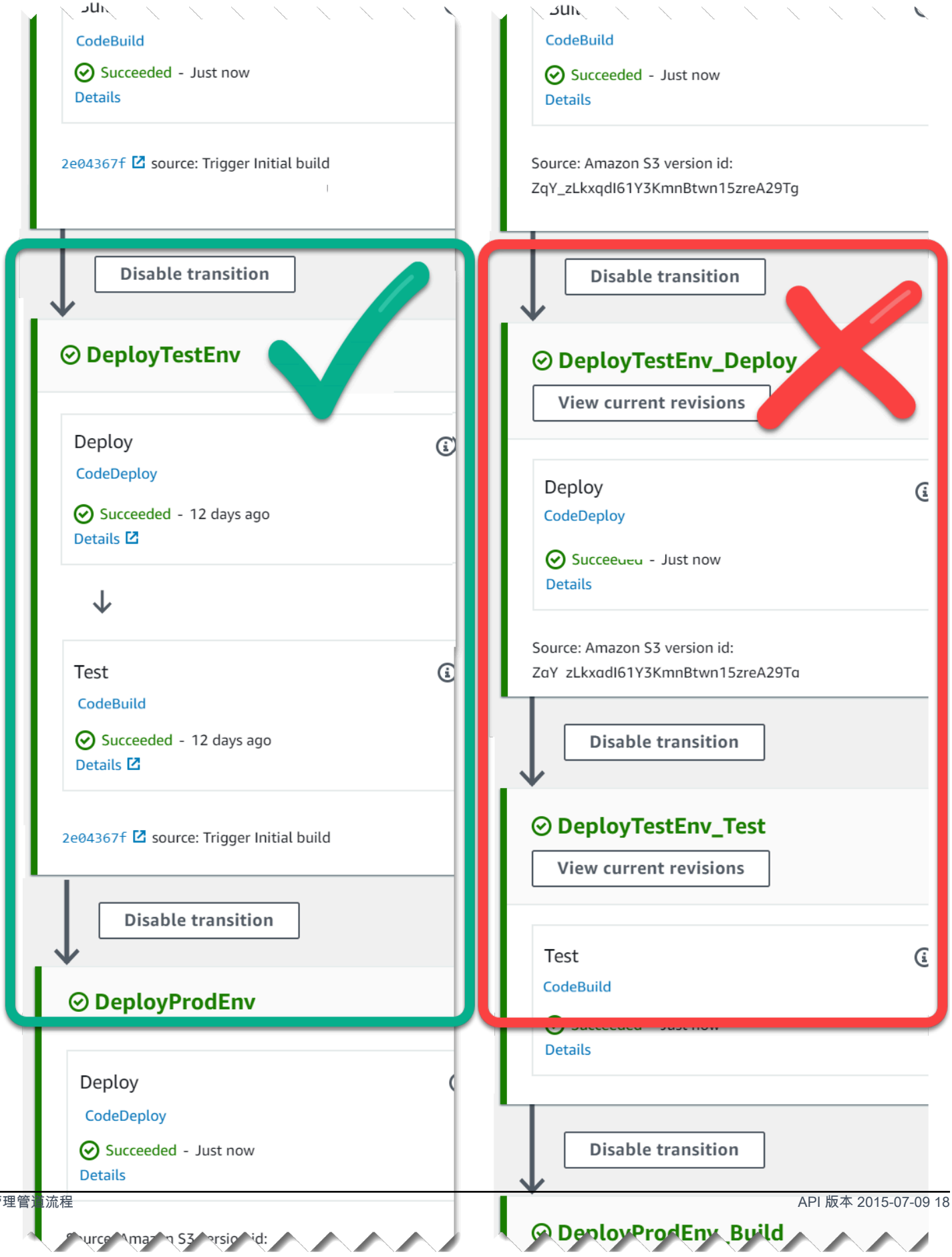
  與等待鎖定階段的執行類似，在停用轉換時，等待進入階段的執行仍可由新的執行取代。當停用的轉換重新啟用時，最新的執行 (包括停用轉換時取代較舊執行的任何執行) 會進入階段。
- 核准動作，可防止管線轉換至下一個動作，直到授與權限為止 (例如，透過授權身分的手動核准)。例如，當您想要控制管道轉換到最終生產階段的時間，您可以使用核准動作。

  > **Note**
  >
  > 具有核准動作的階段會鎖定，直到核准動作獲得核准或拒絕，或已逾時為止。逾時核准動作與失敗動作的處理方式相同。
- 失敗，當階段中的動作未成功完成時。修訂不會轉換到階段中的下一個動作，或管道中的下一個階段。可能會發生下列情況：
  - 您手動重試包含失敗動作的階段。這將恢復執行 (重試失敗的動作，如果成功，則在階段/管道中繼續)。
  - 另一個執行進入失敗的階段，並取代失敗的執行。此時，無法重試失敗的執行。

建議的管道結構

決定程式碼變更如何流經管道時，最好將相關動作聚集在一個階段內，以便階段鎖定時，動作全部處理相同的執行。您可以為每個應用程式環境 (或可用區域) 建立階段，依此類推。AWS 區域階段太多的管道 (也就是太細微) 可能允許太多並行變更，而在較大階段中有許多動作的管道 (太粗糙) 可能花太長時間來發行變更。

例如，在同一階段中，如果測試動作在部署動作之後，則保證可測試已部署的相同變更。在此範例中，變更已部署至「測試」環境而後經過測試，然後測試環境的最新變更會部署至「生產」環境。在建議的範例中，「測試」環境和「生產」環境是不同的階段。


CodeBuild
Succeeded - Just now
Details
2e04367f source: Trigger Initial build
Disable transition
DeployTestEnv
Deploy
CodeDeploy
Succeeded - 12 days ago
Details
Test
CodeBuild
Succeeded - 12 days ago
Details
2e04367f source: Trigger Initial build
Disable transition
DeployProdEnv
Deploy
CodeDeploy
Succeeded - Just now
Details
CodeBuild
Succeeded - Just now
Details
Source: Amazon S3 version id:
ZqY_zLkxqdI61Y3KmnBtwn15zreA29Tg
Disable transition
DeployTestEnv_Deploy
View current revisions
Deploy
CodeDeploy
Succeeded - Just now
Details
Source: Amazon S3 version id:
ZqY_zLkxqdI61Y3KmnBtwn15zreA29Tg
Disable transition
DeployTestEnv_Test
View current revisions
Test
CodeBuild
Details
Disable transition
DeployProdEnv_Build

左邊：相關的測試、部署和核准動作組合在一起 (建議)。右邊：相關動作分屬不同階段 (不建議)。

入站執行的工作方式

輸入執行是指等待無法使用的階段、轉移或動作在向前移動之前變為可用的執行項目。下一個階段、轉場或動作可能無法使用，因為：

- 另一個執行已經進入下一個階段並將其鎖定。
- 進入下一個階段的轉換已停用。

如果您想要控制目前的執行是否有時間在後續階段中完成，或者您想要在特定時間點停止所有動作，您可以停用轉換來保留輸入執行。若要判斷是否有輸入執行，您可以在主控台中檢視管線，或檢視get-pipeline-state命令的輸出。

輸入執行作業時需考量下列事項：

- 一旦動作、轉移或鎖定階段可用，進行中的輸入執行就會進入階段，並透過管線繼續進行。
- 當入站執行正在等待時，可以手動停止。入站執行可以具有`InProgress`、`Stopped`、或`Failed`狀態。
- 當輸入執行已停止或失敗時，無法重試，因為沒有失敗的動作可重試。當輸入執行已停止並啟用轉換時，已停止的輸入執行不會繼續進入階段。

您可以檢視或停止輸入執行。

輸入和輸出成品

CodePipeline 與開發工具整合以檢查程式碼變更，然後透過持續交付程序的所有階段進行建置和部署。人工因素是指管線中動作所處理的檔案，例如包含應用程式程式碼的檔案或資料夾、索引頁檔案、指令碼等。例如，Amazon S3 來源動作成品是一個檔案名稱 (或檔案路徑)，其中為管道來源動作提供應用程式原始程式碼檔案，而這些檔案通常以 ZIP 檔案形式提供，例如下列人工因素名稱範例：SampleApp_Windows.zip。來源動作的輸出人工因素 (應用程式原始程式碼檔案) 是來自來源動作的輸出人工因素，也是下一個動作 (例如建置動作) 的輸入人工因素。另一個範例是，建置動作可能會執行編譯應用程式原始程式碼的建置命令，輸入成品是來自來源動作的應用程式原始程式碼檔。如需有關人工因素參數 (例如動作) 的詳細資訊，請參閱特定動作的 CodeBuild 動作組態AWS CodeBuild參考頁面。

動作使用您在建立管道時選擇的 Amazon S3 成品儲存貯體中存放的輸入和輸出成品。 CodePipeline 根據階段中的動作類型，壓縮和傳輸輸入或輸出加工品的檔案。

> **Note**
>
> 加工品儲存貯體與用作所選來源動作為 S3 之管道來源檔案位置的儲存貯體不同。

例如：

1. CodePipeline 當有提交到來源儲存庫時，會觸發管道執行，並從「來源」階段提供輸出成品 (任何要建立的檔案)。
2. 上一步驟的輸出成品 (任何要建置的檔案) 會擷取成為建置階段的輸入成品。建置階段的輸出成品 (已建置的應用程式) 可能是更新的應用程式，或建置給容器的已更新 Docker 影像。
3. 上一個步驟 (建置的應用程式) 的輸出成品會作為輸入成品擷取到 Deploy 階段，例如中的測試或生產環境。AWS 雲端您可部署應用程式至部署機群，或部署容器式應用程式至 ECS 叢集中執行的任務。

當您建立或編輯動作時，您可以為動作指定一或多個輸入和輸出成品。例如，對於具有「來源」和「部署」階段的兩階段管線，您可以在 「編輯動作」 中為部署動作的輸入成品選擇來源動作的人工因素名稱。

- 當您使用主控台建立第一個管道時，請在同一個管道中建 CodePipeline 立 Amazon S3 儲存貯體，AWS 帳戶並AWS 區域為所有管道存放項目。每次使用控制台在該區域中創建另一個管道時，都 CodePipeline 會在存儲桶中為該管道創建一個文件夾。自動化發行程序執行時，它會使用該資料夾來存放管道的成品。此儲存貯體名為 codepipeline-*region*-*12345EXAMPLE*，其中 *region* 為您建立管道的 AWS 區域，而 *12345EXAMPLE* 是 12 位數的隨機數字，可確保儲存貯體名稱是唯一的。

  > **Note**
  >
  > 如果您已經有一個以 codepipeline-區域開頭的存儲桶-在您要創建管道的區域中，請 CodePipeline 使用該值區作為默認值區。它也遵循詞典順序; 例如，代碼管道-區域 abcexample 在代碼管道-區域防範例之前選擇。

  CodePipeline 截斷成品名稱，這可能會導致某些值區名稱看起來類似。即使成品名稱似乎被截斷，也會以不受名稱截斷的成品影響的方式對 CodePipeline 應至成品值區。該管道可以正常運作。這不是資料夾或成品的問題。管道名稱有 100 個字元的限制。雖然成品資料夾名稱看起來可能變短了，對管道來說仍然是唯一的。

建立或編輯管道時，管線中必須有一個人工因素儲存貯體AWS 區域，AWS 帳戶而且您計劃在其中執行動作的每個區域必須有一個人工因素儲存貯體。如果您使用主控台來建立管道或跨區域動作，CodePipeline 會在您有在其中具有動作的區域中設定預設的成品儲存貯體。

如果您使用建立管道，只AWS CLI要該儲存貯體AWS 帳戶與管道位於同一個 Amazon S3 儲存貯體，就可以將該管道的成品存放在任何 Amazon S3 儲AWS 區域存貯體中。如果您擔心超出帳戶允許的 Amazon S3 儲存貯體限制，可以這樣做。如果您使用建立或編輯管線，並新增「跨區域」動作(在「區域」中使用的提供AWS者與管線不同的動作)，則必須為計劃執行動作的每個額外「區域」提供人工因素值區。AWS CLI

- 每個動作都會有一種類型。根據類型，動作可能會有下列其中一項或兩項：
  - 輸入成品，這是它在動作執行過程所使用或處理的成品。
  - 輸出成品，這是動作的輸出。

  管道中的每個輸出成品都必須有唯一名稱。動作的每個輸入成品都必須符合管道中稍早動作的輸出成品，不論該動作緊接在階段中的動作前面，還是在數個階段之前的階段中執行。

  一個成品可以由多個動作處理。

管線類型

CodePipeline 提供以下管道類型，其特性和價格不同，因此您可以根據應用程式的需求量身打造管道功能和成本。

- V1 類型管道具有包含標準管道、階段和動作層級參數的 JSON 結構。
- V2 類型管線與 V1 類型具有相同的結構，以及用於發行安全性和觸發器組態的其他參數。

如需的定價資訊 CodePipeline，請參閱定價。

有關每種配管類型中參數的詳細資訊，請參閱此CodePipeline 配管結構參照頁面。如需有關要選擇哪種配管類型的資訊，請參閱什麼類型的管道適合我？。

什麼類型的管道適合我？

管線類型由每個配管版本支援的特性和特徵集來決定。

以下是每種管線類型可用的使用案例和特性的摘要。

	V1 類型	第二類型
特性		
使用案例	• 標準部署	• 透過在執行階段傳遞管線層級變數的組態進行部署 • 管線設定為在 Git 標籤上啟動的部署
动作级变量	支援	支援
管線層級變數	不支援	支援
來源修訂取代	不支援	支援
觸發和篩選 Git 標籤、提取要求、分支或檔案路徑	不支援	支援
平行和佇列執行模式	不支援	支援

如需的定價資訊 CodePipeline，請參閱定價。

開始使用 CodePipeline

如果您不熟悉 CodePipeline，可以按照本章中的步驟進行設置後，遵循本指南中的自學課程。

CodePipeline 主控台在可摺疊面板中包含實用資訊，您可以從資訊圖示或頁面上的任何「資訊」連結開啟這些資

訊。()。
您可以隨時關閉此面板。

主 CodePipeline 控台也提供快速搜尋資源的方法，例如儲存庫、建置專案、部署應用程式和管道。選擇 Go to resource (移至資源)，或按 / 鍵，然後輸入資源名稱。任何相符項目都會出現在清單中。搜尋不區分大小寫。您只會看到您有權檢視的資源。如需詳細資訊，請參閱在主控台檢視資源。

您必須先建立AWS 帳戶並建立第一個管理使用AWS CodePipeline者，才能第一次使用。

主題

步驟 1：建立AWS 帳戶和管理使用者

註冊 AWS 帳戶

如果您還沒有 AWS 帳戶，請完成以下步驟建立新帳戶。

註冊 AWS 帳戶

1. 開啟 https://portal.aws.amazon.com/billing/signup。
2. 請遵循線上指示進行。

   部分註冊程序需接收來電，並在電話鍵盤輸入驗證碼。

註冊 AWS 帳戶時，會建立 AWS 帳戶根使用者。根使用者有權存取該帳戶中的所有 AWS 服務和資源。作為最佳安全實務，將管理存取權指派給管理使用者，並且僅使用根使用者來執行需要根使用者存取權的任務。

註冊程序完成後，AWS 會傳送一封確認電子郵件給您。您可以隨時登錄 https://aws.amazon.com/ 並選擇 我的帳戶，以檢視您目前的帳戶活動並管理帳戶。

建立管理使用者

在您註冊 AWS 帳戶 之後，請保護您的 AWS 帳戶根使用者、啟用 AWS IAM Identity Center，以及建立管理使用者，讓您可以不使用根使用者處理日常作業。

保護您的 AWS 帳戶根使用者

1. 選擇 根使用者 並輸入您的 AWS 帳戶電子郵件地址，以帳戶擁有者身分登入 AWS Management Console。在下一頁中，輸入您的密碼。

   如需使用根使用者登入的說明，請參閱 AWS 登入使用者指南中的以根使用者身分登入。
2. 若要在您的根使用者帳戶上啟用多重要素驗證 (MFA)。

   如需指示，請參閱《IAM 使用者指南》中的為 AWS 帳戶根使用者啟用虛擬 MFA 裝置 (主控台)。

建立管理使用者

1. 啟用 IAM Identity Center。

   如需指示，請參閱AWS IAM Identity Center使用者指南中的啟用 AWS IAM Identity Center。
2. 在 IAM Identity Center 中，將管理存取權授予管理使用者。

   有關如何使用 IAM Identity Center 目錄 作為身分來源的教學課程，請參閱《AWS IAM Identity Center 使用者指南》中的以預設 IAM Identity Center 目錄 設定使用者存取權。

以管理員的身分登入

- 若要使用您的 IAM 身分中心使用者登入，請使用建立 IAM 身分中心使用者時傳送至您電子郵件地址的登入 URL。

如需有關如何使用 IAM Identity Center 使用者登入的說明，請參閱《AWS 登入 使用者指南》中的登入 AWS存取入口網站。

步驟 2：套用受管理策略以進行管理存取 CodePipeline

您必須授予與之互動的權限 CodePipeline。執行此操作的最快方法是將受管理的策略套用至系統`AWSCodePipeline_FullAccess`管理使用者。

> **Note**
>
> 該`AWSCodePipeline_FullAccess`政策包括允許控制台用戶將 IAM 角色傳遞給 CodePipeline 或其他許可的許可AWS 服務。這樣可讓該服務擔任該角色，並代表您執行動作。將政策附加至使用者、角色或群組時，便會套用 `iam:PassRole` 許可。確定政策只套用至信任的使用者。當具有這些許可的使用者使用主控台來建立或編輯管道時，可以使用以下選項：
>
> - 建立 CodePipeline 服務角色或選擇現有角色並將角色傳遞給 CodePipeline
> - 可能會選擇建立用於變更偵測的 CloudWatch 事件規則，並將 CloudWatch 事件服務角色傳遞給 CloudWatch 事件
>
> 如需詳細資訊，請參閱授與使用者將角色傳遞給 AWS 服務.

> **Note**
>
> 此`AWSCodePipeline_FullAccess`政策可讓您存取 IAM 使用者可存取的所有 CodePipeline 動作和資源，以及在管道中建立階段時所有可能的動作，例如建立包括 CodeDeploy Elastic Beanstalk 或 Amazon S3 的階段。以最佳實務而言，您應僅授予個人執行其職責所需的許可。如需如何將 IAM 使用者限制為一組有限的 CodePipeline 動作和資源的詳細資訊，請參閱從 CodePipeline 服務角色移除許可。

若要提供存取權，請新增權限至您的使用者、群組或角色：

- AWS IAM Identity Center 中的使用者和群組：

  建立權限合集。請遵循 AWS IAM Identity Center 使用者指南 的 建立權限合集 中的指示。

- 透過身分提供者在 IAM 中管理的使用者：

  建立聯合身分的角色。請遵循 IAM 使用者指南 的 為第三方身分提供者 (聯合) 建立角色 中的指示。
- IAM 使用者：
  - 建立您的使用者可擔任的角色。請遵循 IAM 使用者指南 的 為 IAM 使用者建立角色 中的指示。
  - (不建議) 將政策直接附加至使用者，或將使用者新增至使用者群組。請遵循 IAM 使用者指南 的 新增權限至使用者 (主控台) 中的指示。

步驟 3：安裝 AWS CLI

若要從本機開發機器上的 AWS CLI 呼叫 CodePipeline 命令，您必須安裝 AWS CLI。如果您打算僅使用本指南中針對 CodePipeline 主控台的步驟開始使用，則此步驟為選擇性步驟。

安裝及設定 AWS CLI

1. 在本機電腦上，下載並安裝 AWS CLI。這將使您能夠 CodePipeline 從命令行進行交互。如需詳細資訊，請參閱開始設定 AWS 命令列界面。

   > **Note**
   >
   > CodePipeline 僅適用於 1.7.38 及AWS CLI更高版本。若要判斷您可能已安裝的版本，請執行命令aws --version。AWS CLI若要將舊版的升級AWS CLI到最新版本，請遵循解除安裝中的指示AWS CLI，然後遵循安裝 AWS Command Line Interface.

2. 使用 configure 命令來設定 AWS CLI，如下所示：

   ```
   aws configure
   ```

   出現提示時，請指定要搭配AWS使用之 IAM 使用者的存取金鑰和AWS秘密存取金鑰 CodePipeline。系統提示您輸入預設區域名稱時，請指定將建立管道的區域 (例如 us-east-2)。系統提示您輸入預設輸出格式時，請指定 json。例如：

   ```
   AWS Access Key ID [None]: Type your target AWS access key ID here, and then press
    Enter
   AWS Secret Access Key [None]: Type your target AWS secret access key here, and then
    press Enter
   Default region name [None]: Type us-east-2 here, and then press Enter
   Default output format [None]: Type json here, and then press Enter
   ```

> **Note**
>
> 如需 IAM、存取金鑰和秘密金鑰的詳細資訊，請參閱管理 IAM 使用者的存取金鑰和如何取得登入資料？。
> 如需有關可用之區域和端點的詳細資訊 CodePipeline，請參閱AWS CodePipeline端點和配額。

步驟 4：打開控制台 CodePipeline

- 請登入AWS Management Console並開啟 CodePipeline 主控台，網址為 http://console.aws.amazon.com/codesuite/codepipeline/home。

後續步驟

您已完成事前準備。您可以開始使用 CodePipeline. 若要開始使用 CodePipeline，請參閱CodePipeline教程。

產品與服務整合 CodePipeline

默認情況下 AWS CodePipeline ，與許多 AWS 服務 和合作夥伴的產品和服務集成。請使用以下各節中的資訊，協助您進行設定， CodePipeline 以便與您使用的產品和服務整合。

以下相關資源可協助您使用此服務。

主題

與 CodePipeline 動作類型的整合

本主題中的整合資訊是依 CodePipeline 動作類型組織的。

主題

來源動作整合

下列資訊依 CodePipeline 動作類型組織，可協助您設定 CodePipeline 為與下列來源動作提供者整合。

主題

- GitHub （版本 1）源操作

Amazon ECR 來源動作

Amazon ECR 是 AWS 碼頭映像存儲庫服務。您可以使用 Docker push 和 pull 命令，將 Docker 影像上傳至您的儲存庫。Amazon ECS 任務定義中會使用 Amazon ECR 儲存庫 URI 和影像來參考來源影像資訊。

進一步了解：

- 若要檢視設定參數和範例 JSON/YAML 程式碼片段，請參閱 Amazon ECR
- 在中建立管線 CodePipeline
- 教學課程：使用 Amazon ECR 來源和 ECS 到部署建立管道 CodeDeploy

Amazon S3 來源動作

Amazon S3 是互聯網的存儲。您可以使用 Amazon S3 隨時從 Web 任何地方存放和擷取任意資料量。您可以設定 CodePipeline 為使用版本控制的 Amazon S3 儲存貯體做為程式碼的來源動作。

> **Note**
>
> Amazon S3 也可以作為部署動作包含在管道中。

進一步了解：

- 若要檢視設定參數和範例 JSON/YAML 程式碼片段，請參閱 Amazon S3 來源動作
- 步驟 1：為您的應用程式建立 S3 儲存貯體
- 建立管道 (CLI)
- CodePipeline 使用 Amazon EventBridge （之前的亞馬遜 CloudWatch 活動）來偵測 Amazon S3 來源儲存貯體中的變更。請參閱 一般整合 CodePipeline。

連接到 Bitbucket 雲 GitHub（版本 2），GitHub 企業服務器，GitLab.com 和 GitLab 自我管理

連線 (`CodeStarSourceConnection`動作) 是用來存取您的第三方 Bitbucket 雲端、 GitHub 企業伺服器 GitHub、 GitLab .com 或 GitLab自我管理的儲存庫。

> **Note**
>
> 亞太區域 (香港)、亞太區域 (海德拉巴)、亞太區域 (雅加達)、亞太區域 (墨爾本)、亞太區域 (大阪)、非洲 (開普敦)、中東 (巴林)、中東 (阿聯酋)、歐洲 (西班牙)、歐洲 (蘇黎世)、以色列 (特拉維夫) 或 AWS GovCloud (美國西部) 區域不提供此功能。若要參考其他可用動作，請參閱產品與服務整合 CodePipeline。如需在歐洲 (米蘭) 區域進行此動作的考量，請參閱中的附註CodeStarSourceConnection 適用於 Bitbucket 雲端 GitHub、 GitHub 企業伺服器、GitLab .com 和 GitLab 自我管理動作。

比特桶雲	您可以設定 CodePipeline 使用 Bitbucket 雲端儲存庫做為程式碼的來源。您必須先前建立了一個 Bitbucket 帳戶和至少一個 Bitbucket 雲端儲存庫。您可以透過建立管道或編輯現有管道來新增 Bitbucket Cloud 儲存庫的來源動作。 **Note** 您可以建立連至 Bitbucket Cloud 儲存庫的連線。不支援安裝式 Bitbucket 供應商類型，例如 Bitbucket 伺服器。 您可以設定稱為連線的資源，以允許管道存取第三方程式碼儲存庫。建立連線時，請將 AWS CodeStar 應用程式與第三方程式碼儲存庫安裝，然後將其與您的連線建立關聯。 如果是 Bitbucket 雲端，請使用主控台中的「Bitbucket」選項或 CLI 中的`CodestarSourceConnection` 動作。請參閱 比特桶雲連接。 您可以使用此動作的「完整複製」選項來參照儲存庫 Git 中繼資料，以便下游動作可以直接執行 Git 命令。此選項只能由 CodeBuild 下游動作使用。 進一步了解： • 若要檢視組態參數和範例 JSON/YAML 程式碼片段，請參閱。CodeStarSourceConnection 適用於 Bitbucket 雲端 GitHub、 GitHub 企業伺服器、GitLab .com 和 GitLab 自我管理動作 • 若要檢視使用 Bitbucket Cloud 來源建立管線的入門教學課程，請參閱開始使用連線。

GitHub 或 GitHub 企業雲	您可以設 CodePipeline 定使用 GitHub 儲存庫做為程式碼的來源。您必須先前建立 GitHub 帳戶和至少一個 GitHub儲存庫。您可以透過建立管道或編輯現有管道來為 GitHub 存放庫新增來源動作。 您可以設定稱為連線的資源，以允許管道存取第三方程式碼儲存庫。建立連線時，請將 AWS CodeStar 應用程式與第三方程式碼儲存庫安裝，然後將其與您的連線建立關聯。 使用主控台中的 GitHub (版本 2) 提供者選項或 CLI 中的`CodestarSourceConnection` 動作。請參閱 GitHub 連接。 您可以使用此動作的「完整複製」選項來參照儲存庫 Git 中繼資料，以便下游動作可以直接執行 Git 命令。此選項只能由 CodeBuild 下游動作使用。 進一步了解： • 若要檢視設定參數和範例 JSON/YAML 程式碼片段，請參閱 CodeStarSourceConnection 適用於 Bitbucket 雲端 GitHub、 GitHub 企業伺服器、GitLab .com 和 GitLab 自我管理動作 • 如需說明如何連線至 GitHub 儲存庫並使用「完整複製」選項的教學課程，請參閱教學課程：搭 GitHub 配管線來源使用完整複製。 • 目前的 GitHub 動作是的版本 2 來源動作 GitHub。版本 1 GitHub 動作使用 OAuth 令牌身份驗證進行管理。雖然我們不建議使用 GitHub 版本 1 動作，但具有第 1 GitHub 版動作的現有管道仍可繼續運作，而不會造成任何影響。您現在可以在管道中使用CodeStarSourceConnection 適用於 Bitbucket 雲端 GitHub、 GitHub 企業伺服器、 GitLab .com 和 GitLab 自我管理動作來源動作，透過應用 GitHub 程式管理 GitHub 來源動作。如果您有使用版本 1 GitHub 動作的管道，請參閱中更新該管道以使用第 2 版 GitHub 動作的步驟將 GitHub 版本 1 來源動作更新為 GitHub 版本 2 來源動作。
GitHub 企業伺服器	您可以設定 CodePipeline 使用 GitHub 企業伺服器存放庫做為程式碼的來源。您必須先前建立 GitHub 帳戶和至少一個 GitHub 儲存庫。您可以透過建立管道或編輯現有管道來新增 GitHub企業伺服器儲存庫的來源動作。

	您可以設定稱為連線的資源，以允許管道存取第三方程式碼儲存庫。建立連線時，請將 AWS CodeStar 應用程式與第三方程式碼儲存庫安裝，然後將其與您的連線建立關聯。 使用主控台中的「GitHub 企業伺服器提供者」選項或 CLI 中的`CodestarSourceConnection` 動作。請參閱 GitHub 企業伺服器連線。 **Important** AWS CodeStar 連線不支援 GitHub 企業伺服器 2.22.0 版，因為發行版本中存在已知問題。若要連線，請升級至 2.22.1 版或最新的可用版本。 您可以使用此動作的「完整複製」選項來參照儲存庫 Git 中繼資料，以便下游動作可以直接執行 Git 命令。此選項只能由 CodeBuild 下游動作使用。 進一步了解： • 若要檢視設定參數和範例 JSON/YAML 程式碼片段，請參閱 CodeStarSourceConnection 適用於 Bitbucket 雲端 GitHub、 GitHub 企業伺服器、GitLab .com 和 GitLab 自我管理動作 • 如需說明如何連線至 GitHub 儲存庫並使用「完整複製」選項的教學課程，請參閱教學課程：搭 GitHub 配管線來源使用完整複製。
GitLab.com	您可以設 CodePipeline 定使用 GitLab .com 存放庫做為程式碼的來源。您必須先前已建立 GitLab .com 帳戶和至少一個 GitLab .com 儲存庫。您可以透過建立管道或編輯現有管道來新增 GitLab .com 存放庫的來源動作。 使用主控台中的GitLab提供者選項，或使用 CLI 中`GitLab`提供者的`CodestarSourceConnection` 動作。請參閱 GitLab.com 連線。 進一步了解： • 若要檢視設定參數和範例 JSON/YAML 程式碼片段，請參閱 CodeStarSourceConnection 適用於 Bitbucket 雲端 GitHub、 GitHub 企業伺服器、GitLab .com 和 GitLab 自我管理動作

GitLab 自我管理	您可以設定 CodePipeline 使用 GitLab 自我管理的安裝做為程式碼的來源。您必須先前已建立 GitLab 帳戶，並擁有自我管理 GitLab (企業版或社群版) 的訂閱。您可以建立管道或編輯現有管道，為 GitLab 自我管理的存放庫新增來源動作。 您可以設定稱為連線的資源，以允許管道存取第三方程式碼儲存庫。建立連線時，請將 AWS CodeStar 應用程式與第三方程式碼儲存庫安裝，然後將其與您的連線建立關聯。 使用主控台中的GitLab 自我管理提供者選項或 CLI 中的`CodestarSourceConnection` 動作。請參閱 GitLab 自我管理的連線。 您可以使用此動作的「完整複製」選項來參照儲存庫 Git 中繼資料，以便下游動作可以直接執行 Git 命令。此選項只能由 CodeBuild 下游動作使用。 進一步了解： • 若要檢視設定參數和範例 JSON/YAML 程式碼片段，請參閱 CodeStarSourceConnection 適用於 Bitbucket 雲端 GitHub、 GitHub 企業伺服器、GitLab .com 和 GitLab 自我管理動作 • 如需使用此提供者類型建立連線的步驟，請參閱 GitLab 自我管理的連線。

CodeCommit 來源動作

CodeCommit 是版本控制服務，可讓您私下在雲端存放和管理資產 (例如，文件、來源碼和二進位檔案)。您可以配置 CodePipeline 使用 CodeCommit 存儲庫中的分支作為代碼的源代碼。建立儲存庫並將其與本機上的工作目錄相關聯。然後，您可以建立一個管道，使用分支做為階段中來源動作的一部分。您可以透過建立管線或編輯現有配管來連接至 CodeCommit 存放庫。

您可以使用此動作的「完整複製」選項來參照儲存庫 Git 中繼資料，以便下游動作可以直接執行 Git 命令。此選項只能由 CodeBuild 下游動作使用。

進一步了解：

- 若要檢視組態參數和範例 JSON/YAML 程式碼片段，請參閱。CodeCommit
- 教學：建立範本管道 (CodeCommit 儲存庫)

- CodePipeline 使用 Amazon CloudWatch 事件偵測用作管道來源的 CodeCommit 儲存庫中的變更。每個來源動作都有對應的事件規則。此事件規則將會在儲存庫發生變更時啟動管道。請參閱 一般整合 CodePipeline。

GitHub （版本 1）源操作

GitHub 版本 1 的動作是透過 OAuth 應用程式管理的。在可用區域中，您也可以在管道中使用CodeStarSourceConnection 適用於 Bitbucket 雲端 GitHub、 GitHub 企業伺服器、 GitLab .com 和 GitLab 自我管理動作來源動作，透過 GitHub 應用程式管理 GitHub 來源動作。如果您有使用 GitHub 版本 1 動作的管道，請參閱中更新該管道以使用第 2 GitHub 版動作的步驟將 GitHub 版本 1 來源動作更新為 GitHub 版本 2 來源動作。

> **Note**
>
> 雖然我們不建議使用 GitHub 版本 1 動作，但具有第 1 GitHub 版動作的現有管道仍可繼續運作，而不會造成任何影響。

進一步了解：

- 若要取得有關基於 OAuth 的存取 (與應用程式 GitHub 存取相比) 的更多資訊，請參 GitHub 閱。https://docs.github.com/en/developers/apps/differences-between-github-apps-and-oauth-apps
- 若要檢視包含第 1 版 GitHub 動作詳細資訊的附錄，請參閱附錄 A：第 1 GitHub 版來源動作。

建置動作整合

下列資訊是依 CodePipeline 動作類型組織的，可協助您設定 CodePipeline 為與下列建置動作提供者整合。

主題

- CodeBuild 建置動作
- CloudBees 建置動作
- Jenkins 建置動作
- TeamCity 建置動作

CodeBuild 建置動作

CodeBuild 是全受管組建服務，可編譯原始碼、執行單元測試，並產生可立即部署的成品。

您可以將 CodeBuild 建置動作新增至管線的建置階段。如需詳細資訊，請參閱的 CodePipeline 動作組態參考AWS CodeBuild。

> **Note**
>
> CodeBuild 也可以包含在管道中作為測試動作，無論是否具有構建輸出。

進一步了解：

- 若要檢視組態參數和範例 JSON/YAML 程式碼片段，請參閱。AWS CodeBuild
- 什麼是 CodeBuild？
- CodeBuild— 完全託管的構建服務

CloudBees 建置動作

您可以設定 CodePipeline 為使CloudBees用在管線中的一或多個動作中建置或測試程式碼。

進一步了解：

- 重要：創新 2017：雲端優先搭配 AWS

Jenkins 建置動作

您可以配置 CodePipeline 為使用 Jenkins CI 在管道中的一個或多個動作中構建或測試您的代碼。您必須先前創建了一個 Jenkins 項目，並為該項目安裝和配置了 Jenkins 的 CodePipeline 插件。您可以建立新的管道或編輯現有的管道，以連線至 Jenkins 專案。

詹金斯訪問是在每個項目的基礎上配置的。您必須在要使用的每個 Jenkins 實例上安裝詹金斯 CodePipeline 插件。 CodePipeline您還必須配置對詹金斯項目的 CodePipeline 訪問權限。您應該設定 Jenkins 專案僅接受 HTTPS/SSL 連線，以保護 Jenkins 專案。如果您的 Jenkins 專案安裝在 Amazon EC2 執行個體上，請考慮在每個執行個體 AWS CLI 上安裝以提供您的 AWS 登入資料。然後使用您要用於連線的認證，在這些執行個體上設定設定 AWS 檔。這是透過 Jenkins Web 界面新增和儲存它們的替代方案。

進一步了解：

- 存取 Jenkins
- 教學：建立四階段管道

TeamCity 建置動作

您可以設定 CodePipeline 為使TeamCity用在管線中的一或多個動作中建置和測試程式碼。

進一步了解：

- TeamCity 插件 CodePipeline

測試動作整合

下列資訊依 CodePipeline 動作類型組織，可協助您設定 CodePipeline 為與下列測試動作提供者整合。

主題

- CodeBuild 測試動作
- AWS Device Farm 測試動作
- 鬼 Inspector 測試動作
- 微焦點 StormRunner 負載測試動作

CodeBuild 測試動作

CodeBuild是雲端中完全受控的建置服務。 CodeBuild 編譯您的原始程式碼、執行單元測試，並產生準備好部署的成品。

您可以新增 CodeBuild 至管線做為測試動作。如需詳細資訊，請參閱 AWS CodeBuild 的 CodePipeline 動作組態參考。

> **Note**
>
> CodeBuild 也可以包含在管道中作為構建操作，並具有強制的構建輸出成品。

進一步了解：

- 若要檢視組態參數和範例 JSON/YAML 程式碼片段，請參閱。AWS CodeBuild
- 什麼是 CodeBuild？

AWS Device Farm 測試動作

AWS Device Farm 是一種應用程式測試服務，可用於在實際實體手機和平板電腦上測試 Android、iOS 和 Web 應用程式，並與之互動。您可以配置 CodePipeline 為用 AWS Device Farm 於在管道中的一個或多個動作中測試您的代碼。 AWS Device Farm 允許您上傳自己的測試或使用內置的無腳本兼容性測試。由於系統會平行執行測試，所以會在幾分鐘內開始多個裝置上的測試。測試完成後，會更新包含高階結果、低層級記錄、 pixel-to-pixel 螢幕擷取畫面和效能資料的測試報告。 AWS Device Farm 支持原生和混合 Android，iOS 和消防操作系統應用程序的測試，包括使用鈦 PhoneGap，Xamarin，統一和其他框架創建的應用程序。它支援遠端存取 Android 應用程式，可讓您直接與測試裝置互動。

進一步了解：

- 若要檢視組態參數和範例 JSON/YAML 程式碼片段，請參閱。AWS Device Farm
- 什麼是 AWS Device Farm？
- AWS Device Farm 在測 CodePipeline 試階段中使用

鬼 Inspector 測試動作

您可以設定 CodePipeline 為使用 Ghost Inspector 在管線中的一或多個動作中測試您的程式碼。

進一步了解：

- 鬼 Inspector 文檔與服務集成 CodePipeline

微焦點 StormRunner 負載測試動作

您可以設定 CodePipeline 為在管線中的一或多個動作中使用 Micro Focus StormRunner 載入。

進一步了解：

- 用於整合的微型焦點 StormRunner 負載文件 CodePipeline

部署動作整合

下列資訊依 CodePipeline 動作類型組織，可協助您設定 CodePipeline 為與下列部署動作提供者整合。

主題

Amazon S3 部署動作

Amazon S3 是互聯網的存儲。您可以使用 Amazon S3 隨時從 Web 任何地方存放和擷取任意資料量。您可以將動作新增至使用 Amazon S3 做為部署供應商的管道。

> **Note**
>
> Amazon S3 也可以作為來源動作包含在管道中。

進一步了解：

- 在中建立管線 CodePipeline
- 教學課程：建立使用 Amazon S3 做為部署供應商的管道

AWS AppConfig 部署動作

AWS AppConfig 是建立、管理及快速部署應用程式組態的 AWS Systems Manager 功能。您可以搭 AppConfig 配託管在 EC2 執行個體、容器 AWS Lambda、行動應用程式或 IoT 裝置上的應用程式搭配使用。

進一步了解：

- CodePipeline 的動作組態參照 AWSAppConfig
- 教學：建立將 AWS AppConfig 做為部署提供者使用的管道

AWS CloudFormation 部署動作

AWS CloudFormation讓開發人員和系統管理員能夠輕鬆建立和管理相關資 AWS 源集合，並使用範本佈建和更新這些資源。您可以使用此服務的範例範本或建立自己的範本。範本會描述執行應用程式所需的 AWS 資源以及任何相依性或執行階段參數。

AWS 無伺服器應用程式模型 (AWS SAM) 延伸 AWS CloudFormation 以提供簡化的方式來定義和部署無伺服器應用程式。 AWS SAM 支援 Amazon API Gateway API、 AWS Lambda 函數和 Amazon DynamoDB 資料表。您可以使 CodePipeline 用 AWS CloudFormation 和 AWS SAM 來持續交付無伺服器應用程式。

您可以將動作新增至用 AWS CloudFormation 作部署提供者的管線。當您作 AWS CloudFormation 為部署提供者使用時，可以在管道執行過程中對 AWS CloudFormation 堆疊和變更集採取動作。 AWS CloudFormation 可以在管線執行時建立、更新、取代和刪除堆疊，以及變更集合。因此，可以根據您在 AWS CloudFormation 範本 AWS 和參數定義中提供的規格，在管線執行期間建立、佈建、更新或終止自訂資源。

進一步了解：

- CodePipeline 的動作組態參照 AWS CloudFormation
- 持續交付使用 CodePipeline — 了解如 CodePipeline 何使用建立持續交付工作流程 AWS CloudFormation。
- 自動化部署 Lambda 應用程式 — 瞭解如何使用 AWS 無伺服器應用程式模型，以及為 AWS CloudFormation 您的 Lambda 應用程式建立持續交付工作流程。

AWS CloudFormation StackSets 部署動作

AWS CloudFormation讓您可以跨多個帳戶和 AWS 區域部署資源。

> **Note**
>
> 該`CloudFormationStackSet`和`CloudFormationStackInstances`行動不適用於亞太區域 (香港)、歐洲 (蘇黎世)、歐洲 (米蘭)、非洲 (開普敦) 及中東 (巴林) 等地區。若要參考其他可用動作，請參閱產品與服務整合 CodePipeline。

您可以使用 CodePipeline 與 AWS CloudFormation 來更新堆疊集定義，並將更新部署到執行個體。

您可以將下列動作新增至管線，以用 AWS CloudFormation StackSets 作部署提供者。

- CloudFormationStackSet
- CloudFormationStackInstances

進一步了解：

- CodePipeline 的動作組態參照 AWS CloudFormation StackSets
- 教學課程：使用AWS CloudFormation StackSets 部署動作建立管道

Amazon ECS 部署動作

Amazon ECS 是可高度擴展的高效能容器管理服務，可讓您在. AWS 雲端建立管道時，您可以選取 Amazon ECS 做為部署供應商。原始碼控制儲存庫中的程式碼變更會觸發管道以建立新的 Docker 映像、將其推送到容器登錄，然後將更新的映像部署到 Amazon ECS。您也可以使用中的 ECS (藍/綠) 提供者動作， CodePipeline 將流量路由和部署到 Amazon ECS。 CodeDeploy

進一步了解：

- 什麼是 Amazon ECS？
- 教學課程：持續部署 CodePipeline
- 在中建立管線 CodePipeline
- 教學課程：使用 Amazon ECR 來源和 ECS 到部署建立管道 CodeDeploy

Elastic Beanstalk 部署動作

Elastic Beanstalk 是用於部署和擴展與 Java，淨，PHP，Node.js，Python，紅寶石，圍棋和碼頭開發的 Web 應用程序和服務的服務，如阿帕奇，Nginx 的，乘客和 IIS 熟悉的服務器上開發的 Web 應用程序和服務。您可以配置 CodePipeline 為使用 Elastic Beanstalk 部署您的代碼。您可以在建立管道之前或使用「建立管線」精靈時，建立 Elastic Beanstalk 應用程式和環境，以用於階段中的部署動作。

> **Note**
>
> 此功能不適用於亞太區域 (海德拉巴)、亞太區域 (墨爾本)、中東 (阿拉伯聯合大公國)、歐洲 (西班牙) 或歐洲 (蘇黎世) 區域。若要參考其他可用動作，請參閱產品與服務整合 CodePipeline。

進一步了解：

- 開始使用 Elastic Beanstalk
- 在中建立管線 CodePipeline

AWS OpsWorks 部署動作

AWS OpsWorks 是一項配置管理服務，可幫助您使用 Chef 配置和操作各種形狀和大小的應用程序。您可以使用 AWS OpsWorks Stacks來定義應用程式的架構和每個元件的規格，包括套件安裝、軟體組態和儲存等資源。您可以 CodePipeline 將程式碼與中的自訂 Chef 說明書和應用程式配合使用 AWS OpsWorks Stacks 來部署您的程式 AWS OpsWorks碼。

- 自定義廚師食譜- AWS OpsWorks 使用 Chef 食譜來處理諸如安裝和配置軟件包以及部署應用程序之類的任務。
- 應用程式 — AWS OpsWorks 應用程式是由您想要在應用程式伺服器上執行的程式碼所組成。應用程式程式碼存放在儲存庫中，例如 Amazon S3 儲存貯體。

在建立管線之前，請先建立 AWS OpsWorks 堆疊和圖層。您可以在建立管線之前或使用「建立管線」精靈時，建立要在階段中的部署動作中使用的 AWS OpsWorks 應用程式。

CodePipeline 目前 AWS OpsWorks 僅在美國東部 (維吉尼亞北部) 區域 (us-east-1) 提供支援。

進一步了解：

- CodePipeline 搭配使用 AWS OpsWorks Stacks
- 技術指南和配方
- AWS OpsWorks 應用程式

Service Catalog 部署動作

Service Catalog 可讓組織建立和管理已核准可在上使用的產品目錄 AWS。

> **Note**
>
> 此功能不適用於亞太區域 (海德拉巴)、亞太區域 (雅加達)、亞太區域 (墨爾本)、亞太區域 (大阪)、中東 (阿拉伯聯合大公國)、歐洲 (西班牙)、歐洲 (蘇黎世) 或以色列 (特拉維夫) 等地區。若要參考其他可用動作，請參閱產品與服務整合 CodePipeline。

您可以設定 CodePipeline 將產品範本的更新和版本部署至 Service Catalog。您可以建立要在部署動作中使用的 Service Catalog 產品，然後使用 [建立管線] 精靈建立管線。

進一步了解：

- 教學課程：建立部署至 Service Catalog 的管線
- 在中建立管線 CodePipeline

Amazon Alexa 部署動作

Amazon Alexa Skills Kit 可讓您建置和散佈雲端技能給支援 Alexa 功能的裝置的使用者。

> **Note**
>
> 此功能不適用於亞太區域 (香港) 或歐洲 (米蘭) 地區。若要使用該區域中可用的其他部署動作，請參閱部署動作整合。

您可以將動作新增至使用 Alexa Skills Kit 做為部署提供者的管道。您的管道會偵測到來源變更，然後將更新部署至 Alexa 服務中的技能。

進一步了解：

- 教學：建立部署 Amazon Alexa 技能的管道

CodeDeploy 部署動作

CodeDeploy協調 Amazon EC2 執行個體、現場部署執行個體或兩者的應用程式部署。您可以設定 CodePipeline 為用 CodeDeploy 來部署程式碼。您可以在建立管道之前或使用「建立管線」精靈時，建立要在階段中的部署動作中使用的 CodeDeploy 應用程式、部署和部署群組。

進一步了解：

- 步驟 3：在中建立應用程式 CodeDeploy
- 教學：建立範本管道 (CodeCommit 儲存庫)

XebiaLabs 部署動作

您可以配置 CodePipeline 為用XebiaLabs於在管道中的一個或多個動作中部署您的程式碼。

進一步了解：

- 搭配使用 XL 部署 CodePipeline

與 Amazon 簡易通知服務整合核准動作

Amazon SNS 是一種快速、靈活且全受管的推播通知服務，可讓您傳送個別訊息或將訊息散播給大量收件者。Amazon SNS 可讓您輕鬆且經濟實惠地傳送推播通知給行動裝置使用者、電子郵件收件者，甚至將訊息傳送至其他分散式服務。

在中建立手動核准請求時 CodePipeline，您可以選擇性地發佈到 Amazon SNS 中的某個主題，以便所有訂閱該主題的 IAM 使用者都會收到通知，告知該核准動作已準備好可供檢閱。

進一步了解：

- 什麼是 Amazon SNS？
- 將 Amazon SNS 許可授與 CodePipeline服務角色

叫用動作整合

下列資訊依 CodePipeline 動作類型組織，可協助您設定 CodePipeline 為與下列叫用動作提供者整合。

主題

- Lambda 調用動作
- 斯奈克調用行動
- Step Functions 調用操作

Lambda 調用動作

Lambda 可讓您執行程式碼，無須佈建或管理伺服器。您可以設定 CodePipeline 為使用 Lambda 函數，為管道增加彈性和功能。您可以在建立管道之前或使用「建立管道」精靈時，建立 Lambda 函數以新增為階段中的動作。

進一步了解：

- CodePipeline 的動作組態參照 AWS Lambda
- 在 CodePipeline 中於管道中呼叫 AWS Lambda 函數

斯奈克調用行動

您可以配置 CodePipeline 為使用 Snyk 通過檢測和修復安全漏洞並更新應用程序代碼和容器映像中的依賴關係來保護開放源代碼環境的安全。您也可以使用中的 Snyk 動作 CodePipeline 來自動化管道中的安全性測試控制項。

進一步了解：

- CodePipeline 的動作組態參照 Snyk 動作結構參考
- AWS CodePipeline 使用 Snyk 自動掃描中的漏洞

Step Functions 調用操作

Step Functions 可讓您建立和設定狀態機器。您可以設定 CodePipeline 為使用 Step Functions 叫用動作來觸發狀態機器執行。

> **Note**
>
> 此功能不適用於亞太區域（香港）和歐洲（米蘭）地區。若要參考其他可用動作，請參閱產品與服務整合 CodePipeline。

進一步了解：

- CodePipeline 的動作組態參照 AWS Step Functions
- 教學課程：使用AWS Step Functions叫用管道中的動作

一般整合 CodePipeline

下列 AWS 服務 整合不是以 CodePipeline 動作類型為基礎。

Amazon CloudWatch	Amazon CloudWatch 監控您的 AWS 資源。 進一步了解：

	• 什麼是 Amazon CloudWatch？
Amazon EventBridge	Amazon EventBridge 是一種 Web 服務，可 AWS 服務 根據您定義的規則偵測變更，並在發生變更 AWS 服務 時在一或多個指定的動作中叫用動作。 • 發生變更時自動啟動管道執行 — 您可以在 Amazon 中設定的規則中設定 CodePipeline 為目標 EventBridge。這會設定在另一個服務變更時自動啟動管道。 進一步了解： • 什麼是 Amazon EventBridge？ • 在中啟動管道 CodePipeline. • CodeCommit 來源動作和 EventBridge • 管道狀態變更時接收通知 — 您可以設定 EventBridge 規則來偵測管線、階段或動作的執行狀態變更並回應。 進一步了解： • 監控 CodePipeline 事件 • 教學課程：設定 CloudWatch Events 規則以接收管道狀態變更的電子郵件通知
AWS Cloud9	AWS Cloud9 是一個在線 IDE，您可以通過 Web 瀏覽器訪問它。IDE 提供豐富的程式碼編輯體驗，可支援多種程式設計語言和執行時間除錯器，以及內建終端機。在背景中，Amazon EC2 執行個體託管一個 AWS Cloud9 開發環境。如需詳細資訊，請參閱《AWS Cloud9 使用者指南》。 進一步了解： • 設定 AWS Cloud9

AWS CloudTrail	CloudTrail擷取帳戶或代表 AWS 帳戶發出的 AWS API 呼叫和相關事件，並將日誌檔交付到您指定的 Amazon S3 儲存貯體。您可以設 CloudTrail 定為從 CodePipeline主控台擷取 API 呼叫 AWS CLI、從和 CodePipeline API 擷取 CodePipeline 命令。 進一步了解： • 使用 AWS CloudTrail 記錄 CodePipeline API 呼叫
AWS CodeStar 通知	您可以設定通知，讓使用者知道重要的變更，例如管道開始執行時。如需詳細資訊，請參閱 建立通知規則。

AWS Key Management Service	AWS KMS 是一種受管服務，可讓您輕鬆地建立和控制用來加密資料的加密金鑰。依預設， CodePipeline 用於 AWS KMS 為存放在 Amazon S3 儲存貯體中的管道加密成品。 進一步了解： • 若要從一個 AWS 帳戶建立使用來源儲存貯體、成品儲存貯體和服務角色的管道，以及來自不同 AWS 帳戶的 CodeDeploy 資源，您必須建立客戶管理的 KMS 金鑰、將金鑰新增至管道，並設定帳戶原則和角色以啟用跨帳戶存取。如需詳細資訊，請參閱 在 CodePipeline 中建立使用來自另一個 AWS 帳戶資源的管道。 • 若要從將 AWS CloudFormation 堆疊部署到另一個 AWS 帳戶的帳戶建立管道，您必須建立客 AWS 戶管理的 KMS 金鑰、將金鑰新增至管道，然後設定帳戶原則和角色，以將堆疊部署到另 AWS 一個帳戶。如需詳細資訊，請參閱如 CodePipeline 何使用在不同帳戶中部署 AWS CloudFormation 堆疊？ • 若要為管道的 S3 成品儲存貯體設定伺服器端加密，您可以使用預設的 AWS 受管 KMS 金鑰或建立客戶管理的 KMS 金鑰，然後設定儲存貯體政策以使用加密金鑰。如需詳細資訊，請參閱 為 Amazon S3 中存放的成品設定伺服器端加密 CodePipeline。 對於 AWS KMS key，您可以使用金鑰識別碼、金鑰 ARN 或別名 ARN。 **Note** 別名只能在建立 KMS 金鑰的帳戶中辨識。如果是跨帳戶動作，您只可以使用金鑰 ID 或金鑰 ARN 來識別金鑰。跨帳戶動作涉及使用來自其他帳戶 (AccountB) 的角色，因此指定金鑰 ID 將使用其他帳戶 (AccountB) 中的金鑰。

來自社群的範例

下列各節提供部落格文章、文章和社群所提供範例的連結。

> **Note**
>
> 這些連結僅供參考用途，不應視為完整清單或對範例內容的背書。 AWS 對外部內容的內容或準確性概不負責。

主題

- 整合範例：部落格文章

整合範例：部落格文章

- 從第三方 Git 存儲庫跟踪 AWS CodePipeline 構建狀態

  了解如何設定資源以顯示您的管道，並在第三方儲存庫中建立動作狀態，讓開發人員無需切換內容即可輕鬆追蹤狀態。

  二零二一年三月

- 使用 AWS CodeCommit、 AWS CodeBuild、和完整的 CI/CD 光碟 AWS CodeDeployAWS CodePipeline

  了解如何設定使用 CodeCommit CodePipeline CodeBuild、和 CodeDeploy 服務在一組 Amazon EC2 Linux 執行個體上編譯、建置和安裝版本控制的 Java 應用程式的管道。

  2020 年 9 月出版

- 如何使用以 GitHub 下方式部署到 Amazon EC2 CodePipeline

  了解如何 CodePipeline 從頭開始設定，將開發、測試和 prod 分支部署到不同的部署群組。了解如何使用和設定 IAM 角色、 CodeDeploy 代理程式 CodeDeploy，以及 CodePipeline。

  2020 年 4 月發佈

- 測試和建立 Step Functions 的 CI/CD 管線 AWS

  瞭解如何設定可協調 Step Functions 狀態機器和管道的資源。

  二零二零年三月

- 實作 DevSecOps 使用 CodePipeline

了解如何在中使用 CI/CD 管道 CodePipeline 來自動化預防性和偵測安全控制。這篇文章說明如何使用管道建立簡單的安全性群組，並在來源、測試和生產階段執行安全性檢查，以改善 AWS 帳戶的安全性狀態。

發佈日期：2017 年 3 月

- 使 CodePipeline用 Amazon ECR 和 Amazon ECS 持續部署到亞馬遜 ECS CodeBuild AWS CloudFormation

  了解如何建立 Amazon Elastic Container Service (Amazon ECS) 的持續部署管道。應用程式是以碼頭容器的形式交付 CodePipeline，使用 CodeBuild、、、Amazon ECR 和. AWS CloudFormation

  - 下載範例 AWS CloudFormation 範本和說明，以便從 ECS 參考架構：上 GitHub的持續部署存放庫建立您自己的持續部署管道。

  發佈日期：2017 年 1 月

- Continuous Deployment for Serverless Applications

  了解如何使用的 AWS 服務 集合為無伺服器應用程式建立持續部署管道。您將使用無伺服器應用程式模型 (SAM) 來定義應用程式及其資源，並協調您 CodePipeline 的應用程式部署。

  - 檢視範例應用程式，其以具有 Gin 框架和 API Gateway Proxy 填充碼的 Go 撰寫。

  發佈日期：2016 年 12 月

- 使用 CodePipeline 和調整 DevOps 部署

  了解如何使用 Dynatrace 監控解決方案擴展管道 CodePipeline，在代碼提交之前自動分析測試執行，並保持最佳的前置時間。

  發佈日期：2016 年 11 月

- 建立管線以供 CodePipeline 使用 AWS Elastic Beanstalk 中 AWS CloudFormation 和 CodeCommit

  了解如何在中的應用程式在 CodePipeline 管道中實作持續交付 AWS Elastic Beanstalk。所有 AWS 資源都會透過使用 AWS CloudFormation 範本自動佈建。本逐步解說也包含 CodeCommit 和 AWS Identity and Access Management (IAM)。

  發佈日期：2016 年 5 月

- 自動 CodeCommit 化 CodePipeline 和 AWS CloudFormation

  用 AWS CloudFormation 於自動佈建資 AWS 源，以便使用 CodeCommit CodePipeline、CodeDeploy和的持續交付管道 AWS Identity and Access Management。

發佈日期：2016 年 4 月

- 在中建立跨帳戶管道 AWS CodePipeline

  了解如何使用 AWS Identity and Access Management將跨帳戶存取自動化佈建至 AWS CodePipeline 中的管道。在範 AWS CloudFormation 本中包含範例。

發佈日期：2016 年 3 月

- 探索 ASP.NET Core 第 2 部分：持續交付

  了解如何使用 CodeDeploy 和建立 ASP.NET 核心應用程式的完整持續傳遞系統 AWS CodePipeline。

發佈日期：2016 年 3 月

- 使用 AWS CodePipeline 主控台建立管線

  了解如何在逐步解說中使用 AWS CodePipeline 主控台建立兩階段管道。 AWS CodePipeline 教學：建立四階段管道

發佈日期：2016 年 3 月

- 嘲笑 AWS CodePipeline 管道 AWS Lambda

  了解如何叫用 Lambda 函數，讓您在設計 CodePipeline 軟體交付程序時視覺化該函數的動作和階段，然後再進行管道運作。在設計管線結構時，可以使用 Lambda 函數來測試管道是否成功完成。

發布日期：2016 年 2 月

- 在 CodePipeline 使用中執行 AWS Lambda 函數 AWS CloudFormation

  瞭解如何建立佈建使用者指南工作中使用之所有 AWS 資源的 AWS CloudFormation 堆疊在 CodePipeline 中於管道中呼叫 AWS Lambda 函數。

發布日期：2016 年 2 月

- 佈建自訂 CodePipeline 動作 AWS CloudFormation

  了解如何在中 AWS CloudFormation 使用佈建自訂動作 CodePipeline。

發佈日期：2016 年 1 月

- CodePipeline 使用佈建 AWS CloudFormation

  了解如何在 CodePipeline 使用中佈建基本持續交付管道 AWS CloudFormation。

發佈日期：2015 年 12 月

- 從部署 CodePipeline 到 AWS OpsWorks 使用自訂動作和 AWS Lambda

  了解如何設定管道以及要部署的 AWS Lambda 功 AWS OpsWorks 能 CodePipeline。

  發佈日期：2015 年 7 月

CodePipeline 教程

在您完成開始使用 CodePipeline中的步驟後，您可以嘗試本使用者指南中的其中一個 AWS CodePipeline 教學：

我想使用該嚮導創建一個管道，該管道用於 CodeDeploy 將示例應用程序從 Amazon S3 存儲桶部署到運行 Amazon Linux 的亞馬遜 EC2 實例。使用精靈來建立我的兩階段管道之後，我想要新增第三個階段。	請參閱 教學：建立簡易管道 (S3 儲存貯體)。
我想要建立兩階段管道，用於 CodeDeploy 將範例應用程式從 CodeCommit 儲存庫部署到執行 Amazon Linux 的 Amazon EC2 執行個體。	請參閱 教學：建立範本管道 (CodeCommit 儲存庫)。
我想要將建置階段新增至我在第一個教學課程中建立的三階段管道。新階段會使用 Jenkins 來建置我的應用程式。	請參閱 教學：建立四階段管道。
我想要設定一個 E CloudWatch vents 規則，在管線、階段或動作的執行狀態發生變更時傳送通知。	請參閱 教學課程：設定 CloudWatch Events 規則以接收管道狀態變更的電子郵件通知。
我想創建一個帶有 GitHub 源代碼的管道，該管道使用和構建和測試 Android 應用 CodeBuild 程序AWS Device Farm。	請參閱 教程：創建一個管道，用於構建和測試您的 Android 應用程序 AWS Device Farm。
我想使用 Amazon S3 源創建一個管道，用於測試 iOS 應用程序AWS Device Farm。	請參閱 教學課程：建立測試 iOS 應用程式的管道 AWS Device Farm。
我想要建立一個管道，將我的產品範本部署到 Service Catalog。	請參閱 教學課程：建立部署至 Service Catalog 的管線。
我想使用示例模板使用AWS CloudFormation控制台創建一個簡單的管道（使用 Amazon S3 或 GitHub 源代碼）。 CodeCommit	請參閱 教學：使用 AWS CloudFormation 建立管道。

我想創建一個兩階段的管道，該管道使用 CodeDeploy 和 Amazon ECS 將圖像從 Amazon ECR 存儲庫的藍/綠部署到 Amazon ECS 集群和服務。	請參閱 教學課程：使用 Amazon ECR 來源和 ECS 到部署建立管道 CodeDeploy 。
我想要建立管道，持續將無伺服器應用程式發佈到 AWS Serverless Application Repository。	請參閱 教學課程：建立將無伺服器應用程式發佈到 AWS Serverless Application Repository。

其他使用者指南中的下列教學課程提供了AWS 服務將其他功能整合至管道的指引：

- 建立使用AWS CodeBuild者指南 CodeBuild中使用的管道
- 在使 CodePipeline 用AWS OpsWorks者指南AWS OpsWorks Stacks中搭配使用
- AWS CloudFormation使用者指南 CodePipeline中的持續交付
- 開始在AWS Elastic Beanstalk開發人員指南中使用 Elastic Beanstalk
- 使用以下方式設定持續部署管線 CodePipeline

教學課程：使用 Git 標籤啟動管道

在本教學課程中，您將建立連線至 GitHub 儲存庫的管道，其中針對 Git 標籤觸發類型設定來源動作。在提交上建立 Git 標籤時，您的管道就會啟動。此範例說明如何建立管線，以允許根據標籤名稱的語法篩選標籤。如需使用 glob 模式篩選的詳細資訊，請參閱在語法中使用 glob 模式。

本自學課程透 GitHub 過`CodeStarSourceConnection`動作類型連接到。

> **Note**
>
> 亞太區域 (香港)、非洲 (開普敦)、中東 (巴林) 或歐洲 (蘇黎世) 地區不提供此功能。若要參考其他可用動作，請參閱產品與服務整合 CodePipeline。如需在歐洲 (米蘭) 區域進行此動作的考量，請參閱中的附註CodeStarSourceConnection 適用於 Bitbucket 雲端 GitHub、 GitHub 企業伺服器、 GitLab .com 和 GitLab 自我管理動作。

主題

- 必要條件
- 步驟 1：打開 CloudShell 並克隆您的存儲庫

- 第 2 步：創建一個管道以在 Git 標籤上觸發
- 步驟 3：標記您的提交以進行釋放
- 步驟 4：發布更改並查看日誌

必要條件

開始之前，您必須執行以下作業：

- 使用您的 GitHub 帳戶創建一個 GitHub 存儲庫。
- 準備好您的 GitHub 憑據。當您使用設AWS Management Console定連線時，系統會要求您使用 GitHub 認證登入。

步驟 1：打開 CloudShell 並克隆您的存儲庫

您可以使用命令行界面來克隆存儲庫，進行提交並添加標籤。本自學課程會啟動指令行介面的 CloudShell 例證。

1. 登入 AWS Management Console。
2. 在上方導覽列中，選擇AWS圖示。顯示器的主頁AWS Management Console面。
3. 在上方導覽列中，選擇AWS CloudShell 圖示。 CloudShell 打開。等待 CloudShell 環境建立完成。

   > **Note**
   >
   > 如果沒有看到 CloudShell 圖示，請確定您所在的地區是由支援的地區 CloudShell。本教學課程假設您位於美國西部 (奧勒岡) 區域。

4. 在中 GitHub，導覽至您的存放庫。選擇 [程式碼]，然後選擇 [HTTPS]。複製路徑。複製 Git 儲存庫的地址會複製到剪貼簿。
5. 執行下列命令以複製存放庫。

   ```
   git clone https://github.com/<account>/MyGitHubRepo.git
   ```

6. 輸入您的 GitHub 帳戶`Username`，並`Password`在提示時輸入。對於`Password`條目，您必須使用用戶創建的令牌而不是您的帳戶密碼。

第 2 步：創建一個管道以在 Git 標籤上觸發

在本節中，您可以採取下列動作建立管道：

- 與 GitHub 存放庫和動作連線的來源階段。
- 使用 AWS CodeBuild 建置動作的建置階段。

使用精靈建立管道

1. 請在以下位置登入 CodePipeline 主控台：https://console.aws.amazon.com/codepipeline/。
2. 在 Welcome (歡迎) 頁面、Getting started (入門) 頁面、或者 Pipelines (管道) 頁面上，選擇 Create pipeline (建立管道)。
3. 在 Step 1: Choose pipeline settings (步驟 1：選擇管道設定) 的 Pipeline name (管道名稱) 中，輸入 **`MyGitHubTagsPipeline`**。
4. 在「配管」類型中，將預設選取項保持在 V2。管道類型在特性和價格上有所不同。如需詳細資訊，請參閱管線類型。
5. 在 Service role (服務角色) 中，選擇 New service role (新服務角色)。

   > **Note**
   >
   > 如果您改為選擇使用現有的 CodePipeline 服務角色，請確定已將 `codestar-connections:UseConnection` IAM 權限新增至服務角色政策。如需 CodePipeline 服務角色的指示，請參閱將權限新增至 CodePipeline 服務角色。

6. 在進階設定底下，請保留預設值。在Artifact store (成品存放區) 中，針對您為管道所選取區域中的管道，選擇 Default location (預設位置)，即可使用預設成品存放區 (例如指定為預設值的 Amazon S3 成品儲存貯體)。

   > **Note**
   >
   > 這不是原始碼的來源儲存貯體。這是管道的成品存放區。每個管道都需要有個別成品存放區，例如 S3 儲存貯體。

   選擇 Next (下一步)。
7. 在 Step 2: Add source stage (步驟 2：新增來源階段) 頁面上，新增來源階段：

a. 在來源提供者中，選擇 GitHub (版本 2)。

b. 在「連線」下，選擇現有的連線或建立新的連線。若要建立或管理來 GitHub 源動作的連線，請參閱GitHub 連接。

c. 在 Repository name (儲存庫名稱) 中，選擇 GitHub 儲存庫的名稱。

d. 在管線觸發程序下，選擇 Git 標籤。

在「包含」欄位中，輸入`release*`。

在 D efault 分支中，選擇手動啟動管線時要指定的分支，或使用非 Git 標籤的來源事件。如果變更的來源不是觸發程序，或者是手動啟動管線執行，則使用的變更將是來自預設分支的 HEAD 認可。

> **Important**
>
> 以 Git 標籤觸發器類型開頭的管道將會針對 WebHookv2 事件設定，而且不會使用 Webhook 事件 (對所有推送事件進行變更偵測) 來啟動管線。

選擇 Next (下一步)。

8. 在 Add build stage (新增建置階段) 中，新增建置階段：

a. 在 Build provider (建置供應商) 中，選擇 AWS CodeBuild。允許 Region (區域) 預設為管道區域。

b. 選擇建立專案。

c. 在 Project name (專案名稱) 中，輸入此建置專案的名稱。

d. 在 Environment image (環境映像) 中，選擇 Managed image (受管映像)。針對 Operating system (作業系統)，選擇 Ubuntu。

e. 針對 Runtime (執行時間)，選擇 Standard (標準)。針對映像，選擇 aws/codebuild/standard:5.0。

f. 對於 Service role (服務角色)，選擇 New service role (新服務角色)。

> **Note**
>
> 請記下 CodeBuild 服務角色的名稱。在本教學課程的最後一個步驟中，您將需要角色名稱。

g. 在 BuildSpec 底下，針對 Build specifications (建置規格) 選擇 Insert build commands (插入建置命令)。選擇切換到編輯器，然後將以下內容粘貼到構建命令下。

```
version: 0.2
#env:
  #variables:
     # key: "value"
     # key: "value"
  #parameter-store:
     # key: "value"
     # key: "value"
  #git-credential-helper: yes
phases:
  install:
    #If you use the Ubuntu standard image 2.0 or later, you must specify
 runtime-versions.
    #If you specify runtime-versions and use an image other than Ubuntu
 standard image 2.0, the build fails.
    runtime-versions:
      nodejs: 12
    #commands:
      # - command
      # - command
  #pre_build:
    #commands:
      # - command
      # - command
  build:
    commands:
      -
  #post_build:
    #commands:
      # - command
      # - command
artifacts:
  files:
     - '*'
    # - location
  name: $(date +%Y-%m-%d)
  #discard-paths: yes
  #base-directory: location
#cache:
  #paths:
```

```
    # - paths
```

h. 選擇「繼續」 CodePipeline。這將返回到 CodePipeline控制台並創建一個使用構建命令進行配置的 CodeBuild 項目。組建專案會使用服務角色來管理AWS 服務權限。此步驟可能需要數分鐘。

i. 選擇下一步。

9. 在 Step 4: Add deploy stage (步驟 4：新增部署階段) 頁面上，選擇 Skip deploy stage (跳過部署階段)，然後再次選擇 Skip (跳過) 來接受警告訊息。選擇下一步。

10. 在 Step 5: Review (步驟 5：檢閱) 上，選擇 Create pipeline (建立管道)。

步驟 3：標記您的提交以進行釋放

建立管線並指定 Git 標籤之後，您可以在 GitHub儲存庫中標記提交。在這些步驟中，您將使用標籤`release-1`標記提交。Git 儲存庫中的每個提交都必須有唯一的 Git 標籤。當您選擇提交並加上標籤時，這可讓您將來自不同分支的變更併入管線部署中。請注意，標籤名稱版本不適用於中發行版本的概念 GitHub。

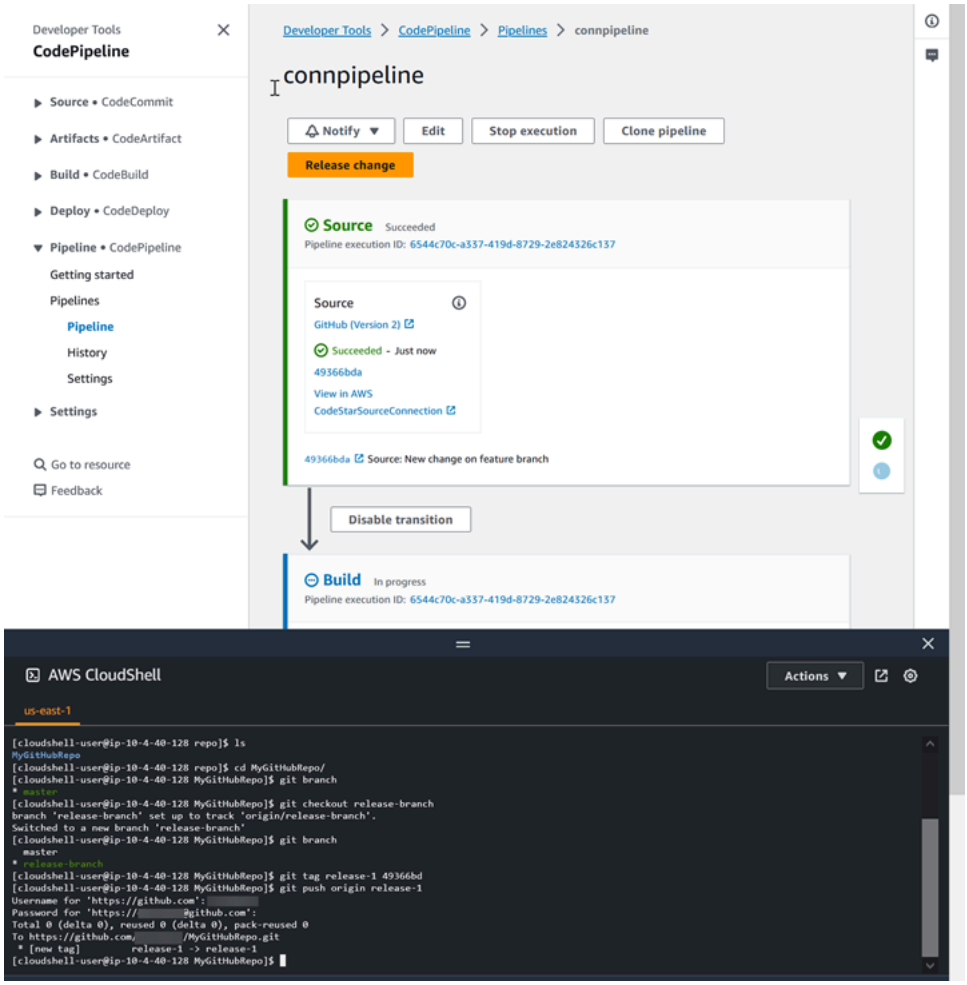
1. 參考您要標記的複製提交 ID。要查看每個分支中的提交，請在 CloudShell 終端機中輸入以下命令以捕獲要標記的提交 ID：

```
git log
```

2. 在 CloudShell 終端中，輸入命令以標記您的提交並將其推送到原點。標記提交後，您可以使用 git push 命令將標籤推送到原點。在下列範例中，輸入下列命令，將`release-1`標籤用於具有 ID 的第二次提交`49366bd`。此標籤將由管線`release*`標籤篩選器篩選，並啟動管線。

```
git tag release-1 49366bd
```

```
git push origin release-1
```



步驟 4：發布更改並查看日誌

1. 管道成功執行之後，在成功的建置階段中，選擇 [檢視記錄]。

   在 [記錄檔] 下，檢視 CodeBuild 組建輸出。這些指令會輸出所輸入變數的值。

2. 在「歷史記錄」頁面中，檢視「觸發程式」欄。查看觸發器類型 GitTag ：釋放 -1。

教學課程：篩選提取要求的分支名稱以啟動管道

在本教學課程中，您將建立一個連線至 GitHub .com 儲存庫的管道，其中將來源動作設定為使用觸發器組態來啟動您的管道，該設定會篩選提取要求。當指定的分支發生指定的提取請求事件時，您的管道會啟動。此範例說明如何建立允許篩選分支名稱的管線。如需使用觸發器的詳細資訊，請參閱在管線

JSON (CLI) 中觸發篩選。有關使用 glob 格式的正則表達式模式進行過濾的更多信息，請參閱在語法中使用 glob 模式。

本教學課程透過CodeStarSourceConnection動作類型連線至 GitHub .com。

主題

必要條件

開始之前，您必須執行以下作業：

- 使用您的 GitHub .com 帳戶創建一個 GitHub .com 存儲庫。
- 準備好您的 GitHub 憑據。當您使用設AWS Management Console定連線時，系統會要求您使用 GitHub 認證登入。

第 1 步：創建一個管道以在指定分支的拉請求上啟動

在本節中，您可以採取下列動作建立管道：

- 連接到您的 GitHub .com 存儲庫和操作的源階段。
- 使用 AWS CodeBuild 建置動作的建置階段。

使用精靈建立管道

1. 請在以下位置登入 CodePipeline 主控台 https://console.aws.amazon.com/codepipeline/。
2. 在 Welcome (歡迎) 頁面、Getting started (入門) 頁面、或者 Pipelines (管道) 頁面上，選擇 Create pipeline (建立管道)。
3. 在 Step 1: Choose pipeline settings (步驟 1：選擇管道設定) 的 Pipeline name (管道名稱) 中，輸入 **`MyFilterBranchesPipeline`**。
4. 在「配管」類型中，將預設選取項保持在 V2。管道類型的特性和價格不同。如需詳細資訊，請參閱 管線類型。
5. 在 Service role (服務角色) 中，選擇 New service role (新服務角色)。

> **Note**
>
> 如果您改為選擇使用現有的 CodePipeline 服務角色，請確定已將 `codestar-connections:UseConnection` IAM 權限新增至您的服務角色政策。如需 CodePipeline 服務角色的指示，請參閱將權限新增至 CodePipeline 服務角色。

6. 在進階設定底下，請保留預設值。在Artifact store (成品存放區) 中，針對您為管道所選取區域中的管道，選擇 Default location (預設位置)，即可使用預設成品存放區 (例如指定為預設值的 Amazon S3 成品儲存貯體)。

> **Note**
>
> 這不是原始碼的來源儲存貯體。這是管道的成品存放區。每個管道都需要有個別成品存放區，例如 S3 儲存貯體。

選擇 Next (下一步)。

7. 在 Step 2: Add source stage (步驟 2：新增來源階段) 頁面上，新增來源階段：

   a. 在來源提供者中，選擇 GitHub (版本 2)。

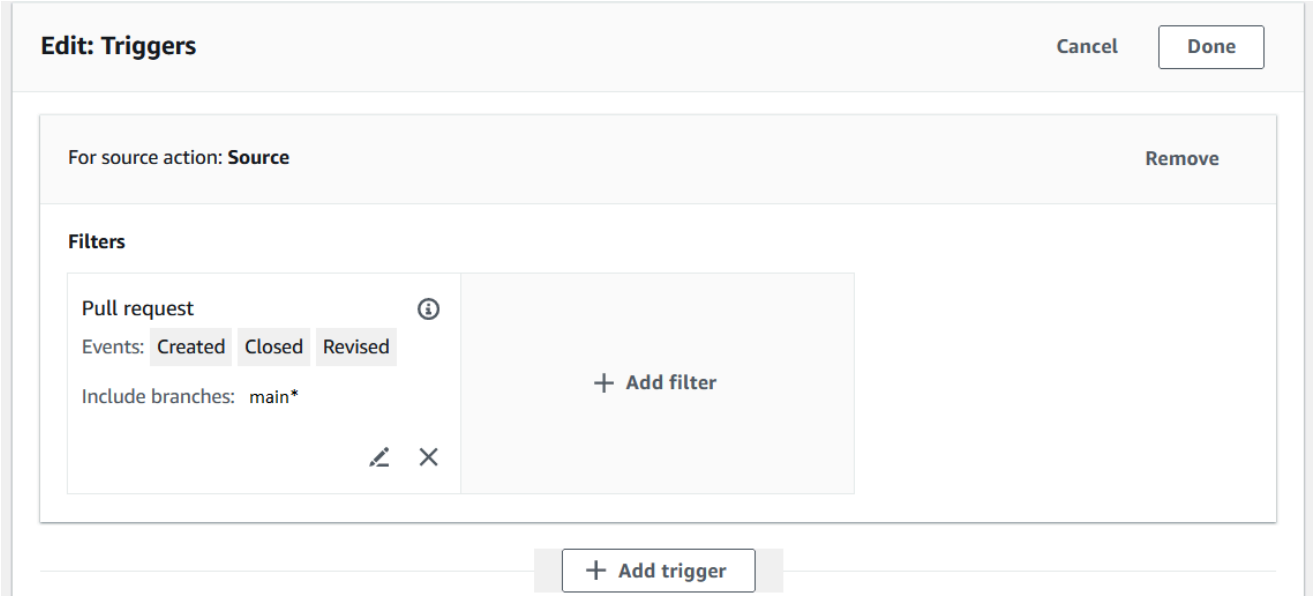
   b. 在「連線」下，選擇現有的連線或建立新的連線。若要建立或管理來 GitHub 源動作的連線，請參閱GitHub 連接。

   c. 在 [存放庫名稱] 中，選擇 GitHub .com 儲存庫的名稱。

   d. 在 [觸發類型] 下，選擇 [指定篩選]

      在事件類型下，選擇提取請求。選取提取要求下的所有事件，以便建立、更新或已關閉的提取要求發生事件。

      在「分支」下的「包含」欄位中輸入`main*`。



> **Important**
>
> 以此觸發器類型開頭的管道將會針對 WebHookv2 事件設定，且不會使用 Webhook 事件 (所有推送事件的變更偵測) 來啟動管線。

選擇下一步。

8. 在 [新增組建階段] 的 [組建提供者] 中，選擇AWS CodeBuild。允許 Region (區域) 預設為管道區域。依照中教學課程：使用 Git 標籤啟動管道的指示選擇或建立建置專案。此動作僅在本教學課程中使用，作為建立管道所需的第二個階段。
9. 在 Step 4: Add deploy stage (步驟 4：新增部署階段) 頁面上，選擇 Skip deploy stage (跳過部署階段)，然後再次選擇 Skip (跳過) 來接受警告訊息。選擇下一步。
10. 在 Step 5: Review (步驟 5：檢閱) 上，選擇 Create pipeline (建立管道)。

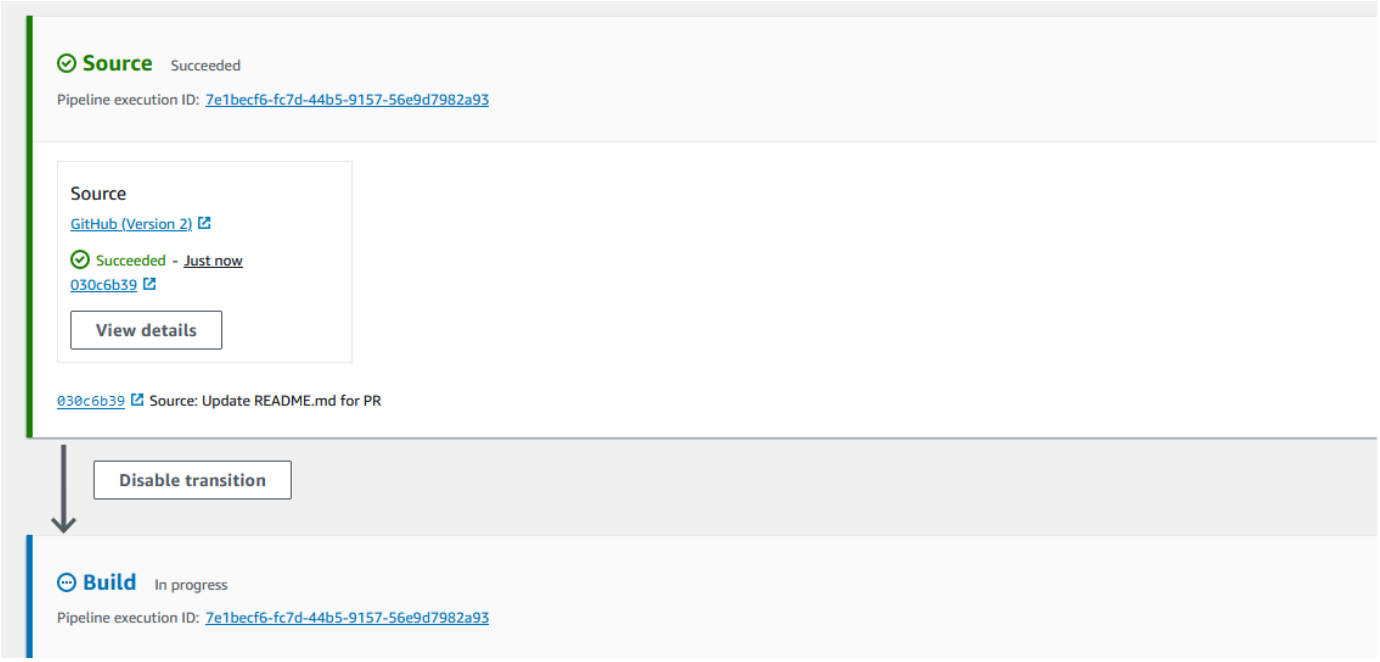
步驟 2：在 GitHub .com 中建立並合併提取要求以開始管道執行

在本節中，您會建立並合併提取要求。這將啟動您的管道，對打開的拉取請求執行一次，並針對已關閉的拉取請求執行一次執行。

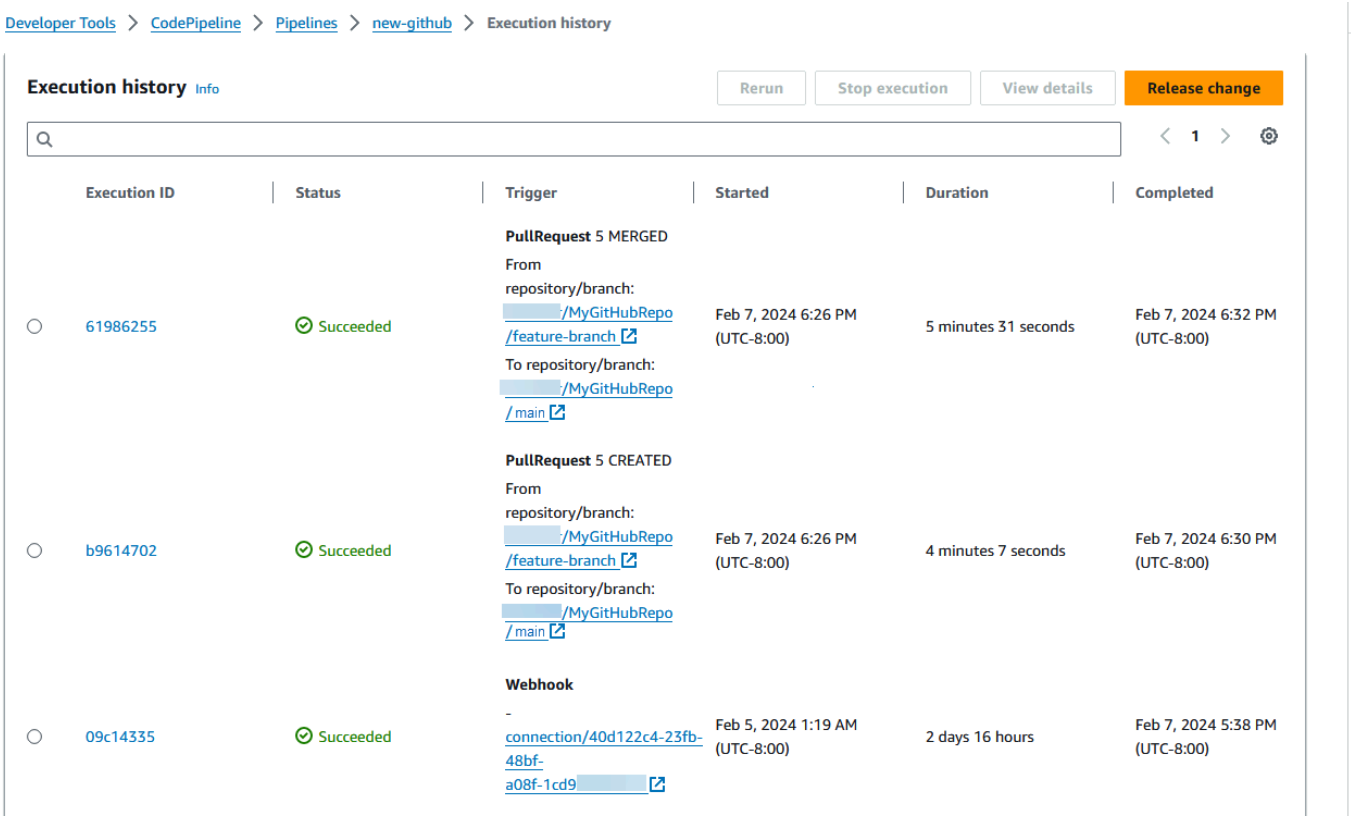
若要建立提取請求並啟動管線

1. 在 GitHub .com 中，透過對功能分支上的 Readme.md 進行變更，並向分支提取要求提取要求來建立提取請求。`main`使用類似的消息提交更改`Update README.md for PR`。

2. 管線從來源修訂開始，將提取要求的「來源」訊息顯示為「更新 Readme.md」(適用於 PR)。



3. 選擇 History (歷程記錄)。在管線執行歷程記錄中，檢視啟動管線執行的 CREATTED 和合併提取要求狀態事件。



教學課程：使用管線層級變數

在本教學課程中，您將建立管線，在其中在管線層級新增變數，並執行輸出變數值的 CodeBuild 建置動作。

主題

必要條件

開始之前，您必須執行以下作業：

- 創建一個 CodeCommit 存儲庫。
- 將 .txt 檔案新增至儲存庫。

第 1 步：創建管道並構建項目

在本節中，您可以採取下列動作建立管道：

- 與 GitHub 存放庫和動作連線的來源階段。
- 使用 AWS CodeBuild 建置動作的建置階段。

使用精靈建立管道

1. 請在以下位置登入 CodePipeline 主控台 https://console.aws.amazon.com/codepipeline/。
2. 在 Welcome (歡迎) 頁面、Getting started (入門) 頁面、或者 Pipelines (管道) 頁面上，選擇 Create pipeline (建立管道)。
3. 在 Step 1: Choose pipeline settings (步驟 1：選擇管道設定) 的 Pipeline name (管道名稱) 中，輸入 **`MyVariablesPipeline`**。
4. 在「配管」類型中，將預設選取項保持在 V2。管道類型的特性和價格不同。如需詳細資訊，請參閱管線類型。
5. 在 Service role (服務角色) 中，選擇 New service role (新服務角色)。

> **Note**
>
> 如果您改為選擇使用現有的 CodePipeline 服務角色，請確定已將 `codestar-connections:UseConnection` IAM 權限新增至您的服務角色政策。如需 CodePipeline 服務角色的指示，請參閱將權限新增至 CodePipeline 服務角色。

6. 在「變數」下選擇「新增變數」。在 Name (名稱) 中，輸入 `timeout`。在預設中，輸入 1000。在說明中，輸入下列描述：**`Timeout`**。

   這將創建一個變量，您可以在其中在管道執行開始時聲明該值。變數名稱必須相符，`[A-Za-z0-9@\-_]+`且可以是空字串以外的任何名稱。

7. 在進階設定底下，請保留預設值。在Artifact store (成品存放區) 中，針對您為管道所選取區域中的管道，選擇 Default location (預設位置)，即可使用預設成品存放區 (例如指定為預設值的 Amazon S3 成品儲存貯體)。

   > **Note**
   >
   > 這不是原始碼的來源儲存貯體。這是管道的成品存放區。每個管道都需要有個別成品存放區，例如 S3 儲存貯體。

   選擇 Next (下一步)。

8. 在 Step 2: Add source stage (步驟 2：新增來源階段) 頁面上，新增來源階段：

   a. 在 Source provider (來源提供者) 中選擇 AWS CodeCommit。

   b. 在存放庫名稱和分支名稱中，選擇您的存放庫和分支。

   選擇 Next (下一步)。

9. 在 Add build stage (新增建置階段) 中，新增建置階段：

   a. 在 Build provider (建置供應商) 中，選擇 AWS CodeBuild。允許 Region (區域) 預設為管道區域。

   b. 選擇建立專案。

   c. 在 Project name (專案名稱) 中，輸入此建置專案的名稱。

d. 在 Environment image (環境映像) 中，選擇 Managed image (受管映像)。針對 Operating system (作業系統)，選擇 Ubuntu。

e. 針對 Runtime (執行時間)，選擇 Standard (標準)。針對映像，選擇 aws/codebuild/standard:5.0。

f. 對於 Service role (服務角色)，選擇 New service role (新服務角色)。

> **Note**
>
> 記下 CodeBuild 服務角色的名稱。在本教學課程的最後一個步驟中，您將需要角色名稱。

g. 在 BuildSpec 底下，針對 Build specifications (建置規格) 選擇 Insert build commands (插入建置命令)。選擇切換到編輯器，然後將以下內容粘貼到構建命令下。在 buildspec 中，客戶變量$CUSTOM_VAR1將用於在構建日誌中輸出管道變量。您將在下面的步驟中創建$CUSTOM_VAR1輸出變量作為環境變量。

```
version: 0.2
#env:
  #variables:
     # key: "value"
     # key: "value"
  #parameter-store:
     # key: "value"
     # key: "value"
  #git-credential-helper: yes
phases:
  install:
    #If you use the Ubuntu standard image 2.0 or later, you must specify
 runtime-versions.
    #If you specify runtime-versions and use an image other than Ubuntu
 standard image 2.0, the build fails.
    runtime-versions:
      nodejs: 12
    #commands:
      # - command
      # - command
  #pre_build:
    #commands:
      # - command
      # - command
```

```
  build:
    commands:
      - echo $CUSTOM_VAR1
  #post_build:
    #commands:
      # - command
      # - command
artifacts:
  files:
     - '*'
    # - location
  name: $(date +%Y-%m-%d)
  #discard-paths: yes
  #base-directory: location
#cache:
  #paths:
    # - paths
```

h. 選擇繼續 CodePipeline。這將返回到 CodePipeline控制台並創建一個使用構建命令進行配置的 CodeBuild 項目。組建專案會使用服務角色來管理AWS 服務權限。此步驟可能需要數分鐘。

i. 在 [環境變數 -選用] 底下，若要建立環境變數做為將由管線層級變數解析之建置動作的輸入變數，請選擇 [新增環境變數]。這將創建在構建規格中指定的變量。$CUSTOM_VAR1在 Name (名稱) 中，輸入 CUSTOM_VAR1。在 Value (值) 中輸入 #{variables.timeout}。在類型中，選擇Plaintext。

   環境變數的#{variables.timeout}值取決於管線層級變數命名空間，以variables及在步驟 5 中為管線timeout建立的管線層級變數。

j. 選擇下一步。

10. 在 Step 4: Add deploy stage (步驟 4：新增部署階段) 頁面上，選擇 Skip deploy stage (跳過部署階段)，然後再次選擇 Skip (跳過) 來接受警告訊息。選擇下一步。

11. 在 Step 5: Review (步驟 5：檢閱) 上，選擇 Create pipeline (建立管道)。

步驟 2：發布更改並查看日誌

1. 管道成功執行之後，在成功的建置階段中，選擇 [檢視詳細資料]。

   在詳細資料頁面上，選擇記錄檔索引標籤。檢視組 CodeBuild 建輸出。這些指令會輸出所輸入變數的值。

2. 在左側導覽列中，選擇 [歷程記錄]。

   選擇最近執行的項目，然後選擇 [變數] 索引標籤。檢視管線變數的已解析值。

教學：建立簡易管道 (S3 儲存貯體)

建立管線的最簡單方法是使用AWS CodePipeline主控台中的「建立管線」精靈。

在此教學中，您將建立使用版本控制的 CodeDeploy 儲存貯體和 S3 的兩階段管道來發佈範例應用程式。

> **Note**
>
> 當 Amazon S3 是管道的來源供應商時，您可以將一或多個來源檔案壓縮為單一 .zip，然後將 .zip 上傳到來源儲存貯體。您也可以上傳單一解壓縮檔案；不過，預期 .zip 檔案的下游動作會失敗。

在您建立此範本管道後，您將新增另一個額外階段，然後停用並啟用階段間的轉換。

> **Important**
>
> 您在此程序中新增至管線的許多動作都涉及建立管線之前需要建立的AWS資源。 AWS來源動作的資源必須始終建立在您建立管道的相同AWS區域中。例如，如果您在美國東部 (俄亥俄) 區域建立管道，則您的 CodeCommit 存放庫必須位於美國東部 (俄亥俄) 區域。
> 您可以在建立管道時新增跨區域動作。 AWS跨區域作業的資源必須位於您計劃執行作業的相同「AWS區域」中。如需詳細資訊，請參閱在 CodePipeline 中新增跨區域動作。

在您開始之前，應先完成 開始使用 CodePipeline 中的事前準備。

主題

- 步驟 1：為您的應用程式建立 S3 儲存貯體
- 步驟 2：建立 Amazon EC2 Windows 執行個體並安裝 CodeDeploy 代理程式
- 步驟 3：在中建立應用程式 CodeDeploy
- 步驟 4：在中建立您的第一個管道 CodePipeline

- (選用) 步驟 5：新增另一個階段至您的管道
- (選擇性) 步驟 6：停用及啟用階段之間的轉換 CodePipeline
- 步驟 7：清除資源

步驟 1：為您的應用程式建立 S3 儲存貯體

您可以將來源檔案或應用程式存放於任何版本控制的位置。在本教學中，您會為範例應用程式檔案建立 S3 儲存貯體，並在該儲存貯體上啟用版本控制。啟用版本控制後，您會複製範本應用程式至該儲存貯體。

建立 S3 儲存貯體

1. 請在以下位置登入主控台AWS Management Console。開啟 S3 主控台。
2. 選擇建立儲存貯體。
3. 在 Bucket Name (儲存貯體名稱) 中，輸入儲存貯體的名稱 (例如 **awscodepipeline-demobucket-example-date**)。

   > **Note**
   >
   > 因為 Amazon S3 中的所有儲存貯體名稱都必須是唯一的，所以請使用您自己的名稱，而不要使用範例中顯示的名稱。您只要新增日期至範例名稱就可變更名稱。請記下此名稱，因為您在整個教學課程中都會用到。

   在 [區域] 中，選擇您要建立管道的區域 (例如美國西部 (奧勒岡)]，然後選擇 [建立值區]。
4. 建立儲存貯體後，會顯示成功橫幅。選擇 Go to bucket details (前往儲存貯體詳細資訊)。
5. 在 Properties (屬性) 標籤上，選擇 Versioning (版本控制)。選擇 Enable versioning (啟用版本控制)，然後選擇 Save (儲存)。

   啟用版本控制後，Amazon S3 會儲存貯體中每個物件的每個版本。
6. 在 Permissions (許可) 索引標籤上，保留預設值。如需 S3 儲存貯體和物件許可的相關資訊，請參閱在政策中指定許可。
7. 接著，下載範本並將其儲存至本機電腦的資料夾或目錄中。

   a. 選擇下列其中一項。如果要依照本教學中針對 Windows Server 執行個體的步驟執行，請選擇 `SampleApp_Windows.zip`。

- 如果您想要使用以下方式部署到 Amazon Linux 執行個體 CodeDeploy，請在此處下載範例應用程式：SampleApp_Linux.zip。
- 如果您想要使用部署到 Windows 伺服器執行個體 CodeDeploy，請在此處下載範例應用程式：SampleApp_Windows.zip。

範例應用程式包含下列檔案以進行部署 CodeDeploy：

- `appspec.yml`— 應用程式規格檔AppSpec案 (檔案) 是用 CodeDeploy 來管理部署的 YAML 格式檔案。若要取得有關 AppSpec 檔案的更多資訊，請參閱《AWS CodeDeploy使用指南》中的〈CodeDeploy AppSpec 檔案參考〉
- `index.html`— 索引檔案包含已部署範例應用程式的首頁。
- `LICENSE.txt`— 授權檔案包含範例應用程式的授權資訊。
- 指令碼檔案 — 範例應用程式會使用指令碼將文字檔寫入執行個體上的某個位置。每個 CodeDeploy 部署生命週期事件都會寫入一個檔案，如下所示：
  - (僅限 Linux 範例) `scripts` 資料夾 — 資料夾包含下列 shell 指令碼，可用來安裝相依性，以及啟動和停止自動化部署的範例應用程式：`install_dependencies`、`start_server`和`stop_server`。
  - (僅限 Windows 範例) `before-install.bat` — 這是`BeforeInstall`部署生命週期事件的批次指令碼，會執行以移除先前部署此範例期間寫入的舊檔案，並在執行個體上建立一個要寫入新檔案的位置。

b. 下載已壓縮的檔案。不要將檔案解壓縮。

8. 在 Amazon S3 主控台中，為您的儲存貯體上傳檔案：

a. 選擇上傳。

b. 拖放檔案或選擇 Add files (新增檔案)，並瀏覽到該檔案。

c. 選擇上傳。

步驟 2：建立 Amazon EC2 Windows 執行個體並安裝 CodeDeploy 代理程式

> **Note**
>
> 本教學提供建立 Amazon EC2 視窗執行個體的範例步驟。如需建立 Amazon EC2 Linux 執行個體的範例步驟，請參閱步驟 3：建立 Amazon EC2 Linux 執行個體並安裝 CodeDeploy 代理程式。當系統提示輸入要建立的執行個體數目時，請指定 2 個執行個體。

在此步驟中，您將建立要部署範例應用程式的 Windows 伺服器 Amazon EC2 執行個體。在此程序中，您會建立具有允許在執行個體上安裝和管理 CodeDeploy 代理程式的原則的執行個體角色。CodeDeploy 代理程式是可在 CodeDeploy 部署中使用執行個體的軟體套件。您也會附加原則，讓執行個體擷取 CodeDeploy 代理程式用來部署應用程式的檔案，並允許 SSM 管理執行個體。

建立執行個體角色

1. 在 https://console.aws.amazon.com/iam/ 開啟身分與存取權管理主控台。
2. 從主控台儀表板，選擇 Roles (角色)。
3. 選擇建立角色。
4. 在 [選取信任實體的類型] 底下，選取AWS 服務。在 Choose a use case (選擇使用案例) 下，選取 EC2，然後選擇 Next: Permissions (下一步：許可)。
5. 搜尋並選取名為的策略**AmazonEC2RoleforAWSCodeDeploy**。
6. 搜尋並選取名為的策略**AmazonSSMManagedInstanceCore**。選擇下一步：標籤。
7. 選擇下一步：檢閱。輸入角色的名稱 (例如，**EC2InstanceRole**)。

   > **Note**
   >
   > 記下您的角色名稱，以用於下一個步驟。您會在建立執行個體時選擇此角色。

   選擇建立角色。

啟動執行個體

1. 在 https://console.aws.amazon.com/ec2/ 開啟 Amazon EC2 主控台。
2. 在側邊導覽中，選擇「執行個體」，然後從頁面頂端選取「啟動執行個體」。

3. 在名稱和標籤下的名稱中，輸入**MyCodePipelineDemo**。這會指定執行個體的標籤索引鍵**Name**和標籤值**MyCodePipelineDemo**。稍後，您會建立將範例 CodeDeploy 應用程式部署到執行個體的應用程式。 CodeDeploy根據標籤選取要部署的執行個體。
4. 在「應用程式和作業系統映像 (Amazon 機器映像)」下，選擇「Windows」選項。（此 AMI 被描述為 Microsoft Windows 服務器 2019 基礎版，並標記為「符合免費方案資格」，可以在快速入門下找到.. ）
5. 在 [執行個體類型] 底下，選擇符合免費方案資格的`t2.micro`類型做為執行個體的硬體組態。
6. 在 [key pair (登入)] 下，選擇金鑰配對或建立金鑰組。

   您也可以選擇不使用 key pair 繼續。

   > **Note**
   >
   > 基於本教學的目的，您可以在不使用金鑰對的情況下繼續進行。若要使用 SSH 連接到執行個體，請建立或使用金鑰對。

7. 在 [網路設定] 底下，執行下列動作。

   在 [自動指派公用 IP] 中，確定狀態為 [啟用]。

   - 在 Assign a security group (指派安全群組) 旁，選擇 Create a new security group (建立新的安全群組)。
   - 在 [SSH] 資料列的 [來源類型] 下，選擇 [我的 IP]。
   - 選擇 [新增安全性群組]，選擇 [HTTP]，然後在 [來源類型] 下選擇 [我的 IP]。
8. 展開 Advanced Details (進階詳細資訊)。在 IAM 執行個體設定檔中，選擇您在上一個程序中建立的 IAM 角色 (例如**EC2InstanceRole**)。
9. 在摘要之下的例證數目下，輸入2..
10. 選擇啟動執行個體。
11. 選擇 View all instances (檢視所有執行個體)，以關閉確認頁面並返回主控台。
12. 您可以在 Instances (執行個體) 頁面上檢視啟動狀態。當您啟動執行個體時，其初始狀態是`pending`。在執行個體啟動後，其狀態會變更為`running`，並得到公有的 DNS 名稱。(如果 Public DNS (公有 DNS) 欄未顯示，請選擇 Show/Hide (顯示/隱藏) 圖示，然後選擇 Public DNS (公有 DNS)。）
13. 執行個體可能需要幾分鐘的時間才能準備就緒讓您連線。請確認您的執行個體已通過狀態檢查。您可以在 Status Checks (狀態檢查) 欄位查看此資訊。

步驟 3：在中建立應用程式 CodeDeploy

在中 CodeDeploy，應用程式是您要部署之程式碼的識別碼，以名稱的形式呈現。 CodeDeploy 使用此名稱來確保在部署期間參考正確的修訂版本、部署規劃和部署群組組合。稍後在本教學課程中建立管線時，您可以選取在此步驟中建立的 CodeDeploy 應用程式名稱。

您首先建立 CodeDeploy 要使用的服務角色。如果您已經建立了服務角色，則不需要再建立一個服務角色。

若要建立 CodeDeploy 服務角色

1. 在 https://console.aws.amazon.com/iam/ 開啟身分與存取權管理主控台。
2. 從主控台儀表板，選擇 Roles (角色)。
3. 選擇建立角色。
4. 在 [選取信任的實體] 下，選擇AWS 服務。在 Use case (使用案例) 中，選擇 CodeDeploy。CodeDeploy從列出的選項中進行選擇。選擇下一步。`AWSCodeDeployRole` 受管政策已連接至角色。
5. 選擇下一步。
6. 輸入角色的名稱 (例如 **`CodeDeployRole`**)，然後選擇 Create role (建立角色)。

若要在中建立應用程式 CodeDeploy

1. 請在以下位置開啟 CodeDeploy 主控台。 https://console.aws.amazon.com/codedeploy
2. 若 Applications (應用程式) 頁面未出現，請在 AWS CodeDeploy 功能表上選擇 Applications (應用程式)。
3. 選擇 建立應用程式。
4. 在 Application name (應用程式名稱) 中，輸入 `MyDemoApplication`。
5. 在 Compute Platform (運算平台) 中，選擇 EC2/On-premises (EC2/ 現場部署)。
6. 選擇 建立應用程式。

若要在中建立部署群組 CodeDeploy

1. 在顯示您應用程式的頁面上，選擇 Create deployment group (建立部署群組)。
2. 在 Deployment group name (部署群組名稱) 中，輸入 **`MyDemoDeploymentGroup`**。

3. 在 [服務角色] 中，選擇您先前建立的服務角色。您必須使用信任至少AWS CodeDeploy 具有建立服務角色中所述的信任和權限的服務角色 CodeDeploy。若要取得服務角色 ARN，請參閱取得服務角色 ARN (主控台)。
4. 在 Deployment type (部署類型) 下，選擇 In-place (就地進行)。
5. 在 Environment configuration (環境組態) 下，選擇 Amazon EC2 Instances (Amazon EC2 執行個體)。在「機碼」欄位中選擇「名稱」，然後在「值」欄位中輸入**MyCodePipelineDemo**。

   > **Important**
   >
   > 在您建立 EC2 執行個體時，您必須在此為 Name (名稱) 金鑰選擇為執行個體指派的相同數值。如果您使用 **MyCodePipelineDemo** 以外的程式碼來標記執行個體，請務必在此使用該程式碼。

6. 在 [使用 AWS Systems Manager 程式組態] 下，選擇 [立即] 並排 這會在執行個體上安裝代理程式。Windows 執行個體已使用 SSM 代理程式設定，現在將以代理 CodeDeploy 程式進行更新。
7. 在「部署設定」 下，選擇`CodeDeployDefault.OneAtaTime`。
8. 在「Load Balancer」下，確定未選取「啟用負載平衡」方塊。您不需要為此範例設定負載平衡器或選擇目標群組。取消選取核取方塊後，不會顯示負載平衡器選項。
9. 在 Advanced (進階) 區段中，保留預設值。
10. 選擇 Create deployment group (建立部署群組)。

步驟 4：在中建立您的第一個管道 CodePipeline

在教學課程的此部分中，您會建立管道。該範本會自動透過管道執行。

若要建立 CodePipeline 自動發行程序

1. 請登入AWS Management Console並開啟 CodePipeline 主控台，網址為 http://console.aws.amazon.com/codesuite/codepipeline/home。
2. 在 Welcome (歡迎) 頁面、Getting started (入門) 頁面、或者 Pipelines (管道) 頁面上，選擇 Create pipeline (建立管道)。
3. 在 Step 1: Choose pipeline settings (步驟 1：選擇管道設定) 的 Pipeline name (管道名稱) 中，輸入 **MyFirstPipeline**。

> **Note**
>
> 如果您選擇了另一個管道名稱，請務必在此整個教學中皆使用該名稱 (而非 **MyFirstPipeline**)。在您建立管道後，便無法更改其名稱。管道名稱會受到一些限制。如需詳細資訊，請參閱AWS CodePipeline 中的配額。

4. 在管線類型中，針對本教學課程的目的選擇 V1。您也可以選擇 V2；但是請注意，管道類型在特性和價格上有所不同。如需詳細資訊，請參閱管線類型。
5. 在 Service role (服務角色) 中，執行下列其中一項作業：
   - 選擇 [新增服務角色] CodePipeline 以允許在 IAM 中建立新的服務角色。
   - 選擇 Existing service role (現有服務角色) 以使用已在 IAM 中建立的服務角色。在 Role name (角色名稱) 中，從清單選擇您的服務角色。
6. 將 Advanced settings (進階設定) 下的設定保留為預設值，然後選擇 Next (下一步)。
7. 在 Step 2: Add source stage (步驟 2：新增來源階段) 的 Source provider (來源提供者) 中，選擇 Amazon S3。在 Bucket (儲存貯體) 中，輸入您在 步驟 1：為您的應用程式建立 S3 儲存貯體 中建立的 S3 儲存貯體名稱。在 S3 物件金鑰中，輸入有或沒有檔案路徑的物件金鑰，記得也要加入副檔名。例如，對於 `SampleApp_Windows.zip`，輸入範例檔案名稱，如下列範例所示：

   ```
   SampleApp_Windows.zip
   ```

   選擇 Next step (下一個步驟)。

   在 Change detection options (變更刪除選項) 下，保持預設設定。這可讓您 CodePipeline 使用 Amazon CloudWatch 事件偵測來源儲存貯體中的變更。

   選擇下一步。
8. 在 Step 3: Add build stage (步驟 3：新增建置階段) 中，選擇 Skip build stage (跳過建置階段)，然後再次選擇 Skip (跳過) 來接受警告訊息。選擇下一步。
9. 在步驟 4：新增部署階段的部署提供者中，選擇CodeDeploy 。「區域」(Region) 欄位預設AWS區域為與您的管道相同。在 Application name (應用程式名稱) 中，輸入 `MyDemoApplication`，或選擇 Refresh (重新整理) 按鈕，然後從清單中選擇應用程式名稱。在 Deployment group (部署群組) 中，輸入 **MyDemoDeploymentGroup**，或從清單中選擇，然後選擇 Next (下一步)。

> **Note**
>
> 名稱「Deploy」(部署)，是預設指定給在 Step 4: Add deploy stage (步驟 4：新增部署階段) 步驟中建立的階段名稱，如同「Source」(來源) 是管道的第一階段所指定的名稱。

10. 在 Step 5: Review (步驟 5：檢閱) 中，檢閱資訊，然後選擇 Create pipeline (建立管道)。
11. 管道開始執行。當 CodePipeline 範例將網頁部署到 CodeDeploy部署中的每個 Amazon EC2 執行個體時，您可以檢視進度、成功和失敗訊息。

恭喜您！您剛剛建立了一個簡單的管道 CodePipeline。管道有兩個階段：

- 名為 Source (來源) 的來源階段，可偵測存放於 S3 儲存貯體中的版本控制範例應用程式變更，並將那些變更提取至管道中。
- 使 CodeDeploy用部署這些變更至 EC2 執行個體的部署階段。

現在，請驗證結果。

驗證您的管道是否成功執行

1. 檢視管道初始進度。每一個階段狀態會從 No executions yet (尚未執行) 變更為 In Progress (進行中)，然後顯示 Succeeded (成功) 或 Failed (失敗)。該管道應會在幾分鐘內完成初次執行。
2. 在該動作狀態顯示 Succeeded (成功) 後，在 Deploy (部署) 階段，選擇 Details (詳細資訊)。這將打開控 CodeDeploy 制台。
3. 在 Deployment group (部署群組) 索引標籤的 Deployment lifecycle events (部署生命週期事件) 下，選擇執行個體 ID。這會開啟 EC2 主控台。
4. 在 Description (敘述) 標籤的 Public DNS (公有 DNS) 中複製地址，然後貼上至您 Web 瀏覽器的地址列中。檢視索引頁面以了解您上傳至 S3 儲存貯體的範例應用程式。

   網頁會針對您上傳到 S3 儲存貯體的範例應用程式顯示。

如需關於階段、動作以及管道如何運作的更多資訊，請參閱 CodePipeline 概念 。

(選用) 步驟 5：新增另一個階段至您的管道

現在請於管道中新增另一個階段，以從預備伺服器部署至使用 CodeDeploy 的生產伺服器。首先，您可以在中建立另一個部署群組 CodeDeploy。 CodePipelineDemoApplication 然後您將新增一個階段，

該階段包括了使用此部署群組的動作。若要新增其他階段，您可以使用 CodePipeline 主控台或擷取並手動編輯 JSON 檔案中管線的結構，然後執行update-pipeline命令以使用您的變更更新管線。AWS CLI

主題

在中建立第二個部署群組 CodeDeploy

> **Note**
>
> 在本教學的這一部分中，您會建立第二個部署群組，但部署到與以前相同的 Amazon EC2 執行個體。這僅用於示範用途。它的設計目的是無法向您展示錯誤的顯示方式 CodePipeline。

若要在中建立第二個部署群組 CodeDeploy

1. 請在以下位置開啟 CodeDeploy 主控台。 https://console.aws.amazon.com/codedeploy
2. 選擇 Applications (應用程式)，然後在應用程式清單中選擇 `MyDemoApplication`。
3. 選擇 Deployment groups (部署群組) 索引標籤，然後選擇 Create deployment group (建立部署群組)。
4. 在 Create deployment group (建立部署群組) 頁面上的 Deployment group name (部署群組名稱) 中，輸入第二個部署群組的名稱 (例如 **`CodePipelineProductionFleet`**)。
5. 在服務角色中，選擇您用於初始部署的相同 CodeDeploy 服務角色 (而非 CodePipeline 服務角色)。
6. 在 Deployment type (部署類型) 下，選擇 In-place (就地進行)。
7. 在 Environment configuration (環境組態) 下，選擇 Amazon EC2 Instances (Amazon EC2 執行個體)。在「關鍵字」方塊中選擇「名稱」，然後在「值」方塊中，`MyCodePipelineDemo`從清單中選擇。將 Deployment settings (部署設定) 保留為預設組態。
8. 在 Deployment configuration (部署組態) 下，選擇 `CodeDeployDefault.OneAtaTime`。
9. 在 Load Balancer (負載平衡器) 下，清除 Enable load balancing (啟用負載平衡)。
10. 選擇 Create deployment group (建立部署群組)。

在您的管道中新增部署群組以做為另一個階段

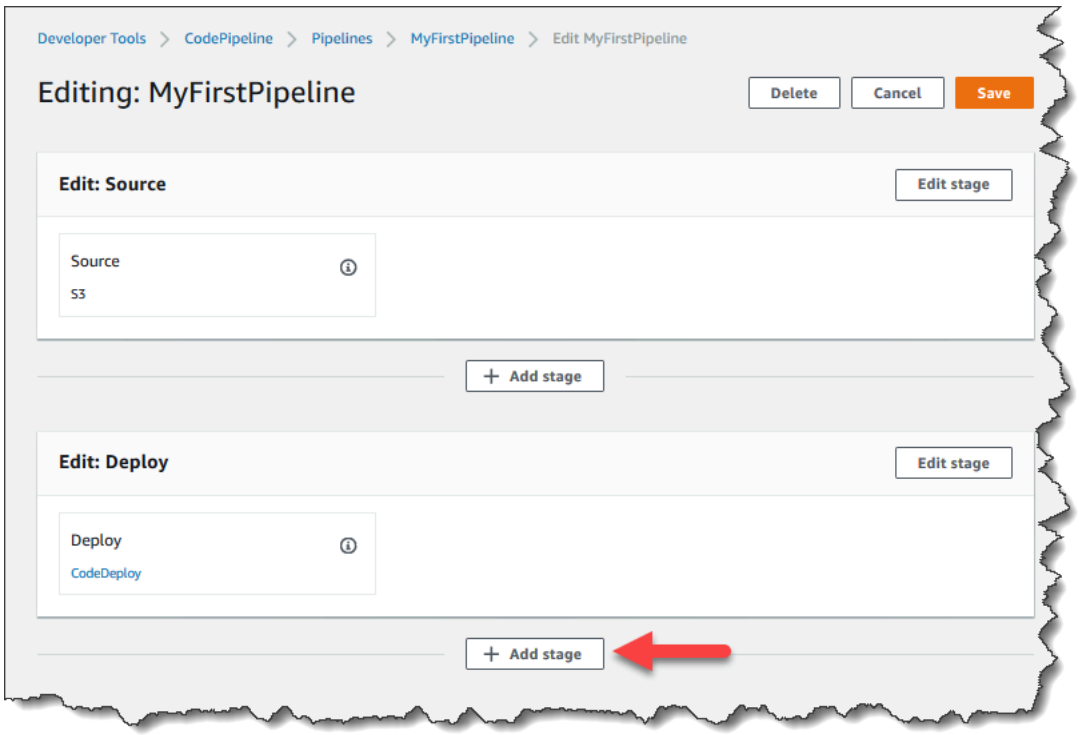
現在您已擁有另一個部署群組，您可新增使用此部署群組的階段，以部署到您在稍早使用的相同 EC2 執行個體。您可以使用 CodePipeline 主控台或 AWS CLI 來新增此階段。

主題

建立第三個階段 (主控台)

您可以使用 CodePipeline 主控台來新增使用新部署群組的新階段。由於此部署群組是部署在您已使用中的 EC2 執行個體，故在此階段的部署動作失敗。

1. 請登入AWS Management Console並開啟 CodePipeline 主控台，網址為 http://console.aws.amazon.com/codesuite/codepipeline/home。
2. 在 Name (名稱) 中，選擇您建立的管道名稱， MyFirstPipeline。
3. 在管道詳細資訊頁面上，選擇 Edit (編輯)。
4. 在 Edit (編輯) 頁面上，選擇 + Add stage (+ 新增階段) 以在 Deploy (部署) 階段後立即新增一個階段。



5. 在 Add stage (新增階段) 的 Stage name (階段名稱) 中，輸入 **Production**。選擇 Add stage (新增階段)。
6. 在新階段中，選擇 + Add action group (+ 新增動作群組)。
7. 在 Edit action (編輯動作) 的 Action name (動作名稱) 中，輸入 **Deploy-Second-Deployment**。在 [動作提供者] 的 [部署] 下，選擇CodeDeploy。
8. 在「應用程式名稱」 CodeDeploy 區段中，MyDemoApplication從下拉式清單中選擇，就像建立管線時所做的一樣。在 Deployment group (部署群組) 中，選擇您剛才建立的部署群組 **CodePipelineProductionFleet**。在 Input artifacts (輸入成品) 中，從來源動作中選擇輸入成品。選擇儲存。
9. 在 Edit (編輯) 頁面中，選擇 Save (儲存)。在 Save pipeline changes (儲存管道變更) 中，選擇 Save (儲存)。
10. 雖然新階段已新增至您的管道，但由於沒有發生觸發另一個管道執行的變更，所以會顯示 No executions yet (尚未執行)。您必須手動重新執行最新的修訂版本，來查看已編輯管道的執行情形。在配管詳細資訊頁面上，選擇「發行變更」，然後在出現提示時選擇「釋放」。這將會在管道的來源動作所指定的各個來源位置中執行最新的可用版本。

    或者，若要從本機 Linux、macOS 或 Unix 機器上的終端機或本機 Windows 電腦上的命令提示字元使用來重新執行管線，請執行start-pipeline-execution命令並指定管線的名稱。AWS CLI這會再次透過管道，在您的來源儲存貯體中執行應用程式。

    ```
    aws codepipeline start-pipeline-execution --name MyFirstPipeline
    ```

    此命令會傳回 pipelineExecutionId 物件。
11. 返回主 CodePipeline 控台，並在管線清單中選擇MyFirstPipeline開啟檢視頁面。

    該管道顯示了三個階段，以及透過這三個階段執行的成品狀態。管道可能需要五分鐘以執行所有階段。和之前一樣，您會看到該部署在前兩個階段已成功，但 Production (生產) 階段會顯示 Deploy-Second-Deployment (部署-第二-部署) 動作失敗。
12. 在 Deploy-Second-Deployment (部署-第二-部署) 動作中，選擇 Details (詳細資訊)。您將會重新導向至頁面以進行 CodeDeploy 部署。在此案例中，該失敗是部署至所有 EC2 執行個體之第一個執行個體群組的結果，造成第二個部署群組沒有任何執行個體。

    > **Note**
    >
    > 此失敗是刻意設計的，以說明在管道階段中發生失敗時的情況。

建立第三個階段 (CLI)

雖然使用 AWS CLI 新增階段至管道比使用主控台更為複雜，但也提供了更多的管道結構可見度。

為管道建立第三階段

1. 在本機 Linux、macOS 或 Unix 電腦上開啟終端機工作階段，或在本機 Windows 電腦上開啟命令提示字元，然後執行get-pipeline命令以顯示剛建立的管線結構。對於 **MyFirstPipeline**，您可能輸入下列命令：

   ```
   aws codepipeline get-pipeline --name "MyFirstPipeline"
   ```

   此指令會傳回的結構 MyFirstPipeline。該輸出的第一部分應如下所示：

   ```
   {
       "pipeline": {
           "roleArn": "arn:aws:iam::80398EXAMPLE:role/AWS-CodePipeline-Service",
           "stages": [
       ...
   ```

   該輸出的最後一個部分包含了管道中繼資料，應如下所示：

   ```
       ...
           ],
           "artifactStore": {
               "type": "S3"
               "location": "codepipeline-us-east-2-250656481468",
           },
           "name": "MyFirstPipeline",
           "version": 4
       },
       "metadata": {
           "pipelineArn": "arn:aws:codepipeline:us-
   east-2:80398EXAMPLE:MyFirstPipeline",
           "updated": 1501626591.112,
           "created": 1501626591.112
       }
   }
   ```

2. 複製並貼上此結構至純文字編輯器中，然後將檔案儲存為 **pipeline.json**。為方便起見，請將此檔案儲存在您執行 aws codepipeline 命令的相同目錄中。

> **Note**
>
> 您可透過 get-pipeline 命令直接將 JSON 輸送到檔案，如下所示：
>
> ```
> aws codepipeline get-pipeline --name MyFirstPipeline >pipeline.json
> ```

3. 複製 Deploy (部署) 階段區段，並貼在前兩個階段的後面。由於它是一個部署階段 (如同 Deploy (部署) 階段)，故您可使用它做為第三階段的範本。
4. 變更階段的名稱和部署群組的詳細資訊。

   下列範例顯示您在部署階段之後新增至管線 .json 檔案的 JSON。使用新數值編輯強調元素。請記得包含逗號，以分隔 Deploy (部署) 和 Production (生產) 階段定義。

```
,
{
    "name": "Production",
     "actions": [
        {
         "inputArtifacts": [
             {
              "name": "MyApp"
             }
          ],
          "name": "Deploy-Second-Deployment",
          "actionTypeId": {
              "category": "Deploy",
              "owner": "AWS",
              "version": "1",
              "provider": "CodeDeploy"
              },
          "outputArtifacts": [],
          "configuration": {
               "ApplicationName": "CodePipelineDemoApplication",
               "DeploymentGroupName": "CodePipelineProductionFleet"
                },
          "runOrder": 1
        }
    ]
}
```

5. 如果您使用的是使用 get-pipeline 命令擷取的管道結構，則必須從 JSON 檔案中移除 `metadata` 行。否則，update-pipeline 命令無法使用它。移除 `"metadata": { }` 行，以及 `"created"`、`"pipelineARN"` 和 `"updated"` 欄位。

   例如，從結構中移除下列幾行：

   ```
   "metadata": {
     "pipelineArn": "arn:aws:codepipeline:region:account-ID:pipeline-name",
     "created": "date",
     "updated": "date"
     }
   ```

   儲存檔案。

6. 執行 update-pipeline 命令，指定管道 JSON 檔案，如以下所示：

   ```
   aws codepipeline update-pipeline --cli-input-json file://pipeline.json
   ```

   此命令會傳回已更新管道的整個結構。

   > **Important**
   >
   > 請確認在檔案名稱之前包含 `file://`。這是此命令必要項目。

7. 執行 start-pipeline-execution 命令，指定管道名稱。這會再次透過管道，在您的來源儲存貯體中執行應用程式。

   ```
   aws codepipeline start-pipeline-execution --name MyFirstPipeline
   ```

   此命令會傳回 `pipelineExecutionId` 物件。

8. 開啟主 CodePipeline 控台，然後MyFirstPipeline從管線清單中選擇。

   該管道顯示了三個階段，以及透過這三個階段執行的成品狀態。管道可能需要五分鐘以執行所有階段。和之前一樣，雖然該部署在前兩個階段已成功，Production (生產) 階段仍會顯示 Deploy-Second-Deployment (部署-第二-部署) 動作失敗。

9. 在 Deploy-Second-Deployment (部署-第二-部署) 動作中，選擇 Details (詳細資訊) 以查看該失敗的詳細資訊。系統會將您重新導向至 CodeDeploy 部署的詳細資料頁面。在此案例中，該失敗是部署至所有 EC2 執行個體之第一個執行個體群組的結果，造成第二個部署群組沒有任何執行個體。

> **Note**
>
> 此失敗是刻意設計的，以說明在管道階段中發生失敗時的情況。

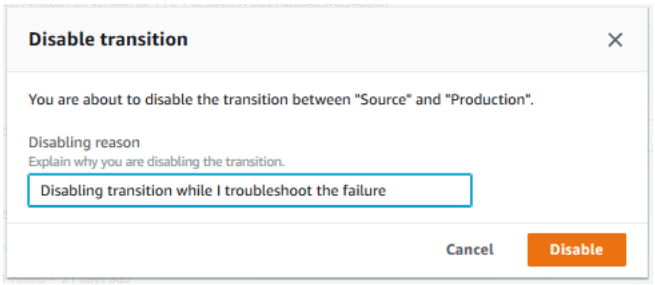
(選擇性) 步驟 6：停用及啟用階段之間的轉換 CodePipeline

您可在管道中的階段之間啟用或停用轉換。在階段間停用轉換，可讓您手動控制階段與階段之間的轉換。例如，您可能想執行管道的前兩個階段，但不希望在準備好部署至生產前轉換至第三個階段，或您正在針對該階段的問題或失敗進行故障排除。

若要停用和啟用 CodePipeline 管線中階段之間的轉移

1. 開啟主 CodePipeline 控台，然後MyFirstPipeline從管線清單中選擇。
2. 在管線的詳細資訊頁面上，選擇第二個階段 (部署) 和您在上一節 (生產) 中新增的第三個階段之間的 [停用轉換] 按鈕。
3. 在 Disable transition (停用轉換) 中，輸入在階段間停用轉換的原因，然後選擇 Disable (停用)。

   階段之間的箭號會顯示圖示與顏色變更，並會顯示 Enable transition (啟用轉換) 按鈕。



4. 再次將您的範例上傳至 S3 儲存貯體。由於該儲存貯體是版本控制的，此變更會啟動該管道。
5. 返回您管道的詳細資訊頁面並查看階段狀態。管道檢視將改變，以顯示前兩個階段的進度與成功狀態，但第三個階段上不會發生變更。此程序可能需要幾分鐘的時間。
6. 選擇兩階段之間的 Enable transition (啟用轉換) 按鈕以啟用轉換。在 Enable transition (啟用轉換) 對話方塊中，選擇 Enable (啟用)。該階段會在幾分鐘內開始執行，並嘗試處理已透過管道前兩個階段執行的成品。

> **Note**
>
> 如果您希望第三個階段成功，請在啟用轉換之前編輯 CodePipelineProductionFleet 部署群組，並指定部署應用程式的不同 EC2 執行個體集。有關如何執行此操作的詳細資訊，請參閱變更部署群組設定。如果您建立更多 EC2 執行個體，您可能需要支付額外費用。

步驟 7：清除資源

您可以將本教學課程中建立的一些資源應用在 教學：建立四階段管道 中。例如，您可以重複使用 CodeDeploy 應用程式和部署。您可以使用提供者設定建置動作，例如 CodeBuild，這是雲端中完全受控的建置服務。您也可以透過組件伺服器或系統來設定使用供應商的組件動作，例如 Jenkins。

不過，在您完成這些教學課程之後，應該刪除管道以及其所使用的資源，才不會因為持續使用那些資源而付費。首先，刪除管道，然後刪除 CodeDeploy 應用程式及其關聯的 Amazon EC2 執行個體，最後刪除 S3 儲存貯體。

清除此教學中使用的資源

1. 若要清理資 CodePipeline 源，請遵循中刪除管道中的指示AWS CodePipeline。
2. 若要清理資 CodeDeploy 源，請依照清理資源 (主控台) 中的指示進行。
3. 若要刪除 S3 儲存貯體，請按照刪除或清空儲存貯體中的說明進行。如果您不想建立更多管道，請刪除用以存放管道成品所建立的 S3 儲存貯體。如需此儲存貯體的詳細資訊，請參閱 CodePipeline 概念 。

教學：建立範本管道 (CodeCommit 儲存庫)

在本教學中，您可 CodePipeline 以使用將 CodeCommit 儲存庫中維護的程式碼部署到單一 Amazon EC2 執行個體。當您將變更推送至 CodeCommit 儲存庫時，就會觸發管道。管道會使用 CodeDeploy 做為部署服務將您的變更部署到 Amazon EC2 執行個體。

管道有兩個階段：

- 來源動作的來源階段 (來 CodeCommit 源)。
- 部署動作的部署階段 (CodeDeploy 部署)。

開始使用最簡單的方法AWS CodePipeline是使用 CodePipeline 主控台中的「建立管線」精靈。

> **Note**
>
> 在開始之前，請確定您已設定要使用的 Git 用戶端 CodeCommit。如需說明，請參閱設定 CodeCommit。

步驟 1：建立 CodeCommit 儲存庫

首先，您可以在中建立一個存放庫 CodeCommit。您的管道會在執行時從此儲存庫獲得原始程式碼。您還可以創建一個本地存儲庫，在將代碼推送到 CodeCommit 存儲庫之前維護和更新代碼。

若要建立存 CodeCommit 放庫

1. 請在以下位置開啟 CodeCommit 主控台。 https://console.aws.amazon.com/codecommit/
2. 在「區域」選取器中，選擇您AWS 區域要建立存放庫和管道的位置。如需詳細資訊，請參閱 AWS 區域 和端點。
3. 請在 Repositories (儲存庫) 頁面上，選擇 Create repository (建立儲存庫)。
4. 在 Create repository (建立儲存庫) 頁面的 Repository name (儲存庫名稱) 中，輸入儲存庫的名稱 (例如 **MyDemoRepo**)。
5. 選擇建立。

> **Note**
>
> 本教學課程中的其餘步驟用**MyDemoRepo**於 CodeCommit 儲存庫的名稱。如果您選擇不同名稱，請在此教學課程中都使用此名稱。

設定本機儲存庫

在此步驟中，您會設定本機儲存庫，以連接至遠端 CodeCommit 儲存庫。

> **Note**
>
> 您不需要設定本機存放庫。您也可以使用控制台上傳檔案，如中所述步驟 2：將示例代碼添加到存 CodeCommit 儲庫。

1. 隨著新儲存庫在主控台開啟後，選擇頁面右上角的 Clone URL (複製 URL)，然後選擇 Clone SSH (複製 SSH)。複製 Git 儲存庫的地址會複製到剪貼簿。
2. 在終端機或命令列，導覽至您要存放本機儲存庫的本機目錄。我們在此教學中使用 `/tmp`。
3. 執行以下命令來複製儲存庫，將 SSH 地址取代為您在上一個步驟中複製的地址。此命令會建立名為 `MyDemoRepo` 的目錄。您將範例應用程式複製到此目錄。

```
git clone ssh://git-codecommit.us-west-2.amazonaws.com/v1/repos/MyDemoRepo
```

步驟 2：將示例代碼添加到存 CodeCommit 儲庫

在此步驟中，您會下載為範例逐步解說所建立的範 CodeDeploy例應用程式的程式碼，並將其新增至您的 CodeCommit 存放庫。

1. 接著，下載範本並將其儲存至本機電腦的資料夾或目錄中。

   a. 選擇下列其中一項。選擇是`SampleApp_Linux.zip`否要遵循本教學課程中針對 Linux 執行個體的步驟進行操作。

   - 如果您想要使用以下方式部署到 Amazon Linux 執行個體 CodeDeploy，請在此處下載範例應用程式：SampleApp_Linux.zip。
   - 如果您想要使用部署到 Windows 伺服器執行個體 CodeDeploy，請在此處下載範例應用程式：SampleApp_Windows.zip。

   範例應用程式包含下列檔案以進行部署 CodeDeploy：

   - `appspec.yml`— 應用程式規格檔AppSpec案 (檔案) 是用 CodeDeploy 來管理部署的 YAML 格式檔案。若要取得有關 AppSpec 檔案的更多資訊，請參閱《AWS CodeDeploy使用指南》中的〈CodeDeploy AppSpec 檔案參考〉
   - `index.html`— 索引檔案包含已部署範例應用程式的首頁。
   - `LICENSE.txt`— 授權檔案包含範例應用程式的授權資訊。
   - 指令碼檔案 — 範例應用程式會使用指令碼將文字檔寫入執行個體上的某個位置。每個 CodeDeploy 部署生命週期事件都會寫入一個檔案，如下所示：
     - (僅限 Linux 範例) `scripts` 資料夾 — 資料夾包含下列 shell 指令碼，可用來安裝相依性，以及啟動和停止自動化部署的範例應用程式：`install_dependencies`、`start_server`和`stop_server`。

- (僅限 Windows 範例) `before-install.bat` — 這是`BeforeInstall`部署生命週期事件的批次指令碼，會執行以移除先前部署此範例期間寫入的舊檔案，並在執行個體上建立一個要寫入新檔案的位置。

b. 下載已壓縮的檔案。

2. 將檔案從 SampleApp_Linux.zip 解壓縮到您先前建立的本機目錄中 (例如，`/tmp/MyDemoRepo`或`c:\temp\MyDemoRepo`)。

   請務必直接將檔案放入您的本機儲存庫。不要包含 `SampleApp_Linux` 資料夾。例如，在您的本機 Linux、macOS 或 Unix 機器上，您的目錄和檔案階層應如下所示：

```
/tmp
   #-- MyDemoRepo
       #-- appspec.yml
       #-- index.html
       #-- LICENSE.txt
       #-- scripts
           #-- install_dependencies
           #-- start_server
           #-- stop_server
```

3. 若要將檔案上傳至儲存庫，請使用下列其中一種方法。

   a. 若要使用 CodeCommit 主控台上傳檔案：

      i. 開啟主 CodeCommit 控台，然後從「儲存庫」清單中選擇您的存放庫。

      ii. 選擇 Add file (新增檔案)，然後選擇 Upload file (上傳檔案)。

      iii. 選取 Choose file (選擇檔案)，然後瀏覽您的檔案。若要在資料夾下新增檔案，請選擇 [建立檔案]，然後輸入具有檔案名稱的資料夾名稱，例如`scripts/install_dependencies`。將文件內容粘貼到新文件中。

      輸入您的使用者名稱和電子郵件地址來確定變更。

      選擇 Commit changes (遞交變更)。

      iv. 針對每個檔案重複此步驟。

      您的儲存庫內容應如下所示：

```
       #-- appspec.yml
       #-- index.html
```

```
        #-- LICENSE.txt
        #-- scripts
            #-- install_dependencies
            #-- start_server
            #-- stop_server
```

b. 要使用 git 命令上傳文件：

i. 將目錄變更為您的本機儲存庫：

```
(For Linux, macOS, or Unix) cd /tmp/MyDemoRepo
(For Windows) cd c:\temp\MyDemoRepo
```

ii. 請執行下列命令來同時將所有檔案放入階段：

```
git add -A
```

iii. 請執行下列命令來確認檔案並附加確認訊息：

```
git commit -m "Add sample application files"
```

iv. 執行下列命令來從本機儲存庫推送檔案到您的 CodeCommit 儲存庫：

```
git push
```

4. 您下載並添加到本地存儲庫的文件現在已添加到 CodeCommit `MyDemoRepo`存儲庫中的`main`分支中，並準備好包含在管道中。

步驟 3：建立 Amazon EC2 Linux 執行個體並安裝 CodeDeploy 代理程式

在此步驟中，您會建立 Amazon EC2 執行個體，在其中部署範例應用程式。在此程序中，建立允許在執行個體上安裝和管理 CodeDeploy 代理程式的執行個體角色。 CodeDeploy 代理程式是可在 CodeDeploy 部署中使用執行個體的軟體套件。您也會附加原則，讓執行個體擷取 CodeDeploy 代理程式用來部署應用程式的檔案，並允許 SSM 管理執行個體。

建立執行個體角色

1. 在 https://console.aws.amazon.com/iam/ 開啟身分與存取權管理主控台。
2. 從主控台儀表板，選擇 Roles (角色)。
3. 選擇建立角色。

4. 在 [選取信任實體的類型] 底下，選取AWS 服務。在 [選擇使用案例] 下，選取 [EC2]。在 Select your use case (選取您的使用案例) 下，選擇 EC2。選擇 Next: Permissions (下一步：許可)。
5. 搜尋並選取名為的策略**AmazonEC2RoleforAWSCodeDeploy**。
6. 搜尋並選取名為的策略**AmazonSSMManagedInstanceCore**。選擇下一步：標籤。
7. 選擇下一步：檢閱。輸入角色的名稱 (例如，**EC2InstanceRole**)。

> **Note**
>
> 記下您的角色名稱，以用於下一個步驟。您會在建立執行個體時選擇此角色。

選擇建立角色。

啟動執行個體

1. 在 https://console.aws.amazon.com/ec2/ 開啟 Amazon EC2 主控台。
2. 在側邊導覽中，選擇「執行個體」，然後從頁面頂端選取「啟動執行個體」。
3. 在 Name (名稱) 中，輸入 **MyCodePipelineDemo**。這會指定執行個體的標籤 Key **Name** 和標籤值**MyCodePipelineDemo**。稍後，您會建立將範例 CodeDeploy 應用程式部署到此執行個體的應用程式。 CodeDeploy根據標籤選取要部署的執行個體。
4. 在「應用程式和作業系統映像 (Amazon 機器映像)」下，找到帶有AWS標誌的 Amazon Linux AMI 選項，並確定已選取該選項。（這個 AMI 被描述為 Amazon Linux 2 AMI（HVM），並被標記為「符合免費方案的資格」。）
5. 在「執行個體類型」下，選擇免費方案合格`t2.micro`類型做為執行個體的硬體組態。
6. 在 [key pair (登入)] 下，選擇金鑰配對或建立金鑰組。

   您也可以選擇不使用 key pair 繼續。

> **Note**
>
> 基於本教學的目的，您可以在不使用金鑰對的情況下繼續進行。若要使用 SSH 連接到執行個體，請建立或使用金鑰對。

7. 在 [網路設定] 底下，執行下列動作。

   在 [自動指派公用 IP] 中，確定狀態為 [啟用]。

- 在 Assign a security group (指派安全群組) 旁，選擇 Create a new security group (建立新的安全群組)。
- 在 [SSH] 資料列的 [來源類型] 下，選擇 [我的 IP]。
- 選擇 [新增安全性群組]，選擇 [HTTP]，然後在 [來源類型] 下選擇 [我的 IP]。

8. 展開 Advanced Details (進階詳細資訊)。在 IAM 執行個體設定檔中，選擇您在上一個程序中建立的 IAM 角色 (例如**EC2InstanceRole**)。
9. 在摘要之下的例證數目下，輸入1..
10. 選擇啟動執行個體。
11. 您可以在 Instances (執行個體) 頁面上檢視啟動狀態。當您啟動執行個體時，其初始狀態是 pending。在執行個體啟動後，其狀態會變更為 running，並得到公有的 DNS 名稱。(如果 Public DNS (公有 DNS) 欄未顯示，請選擇 Show/Hide (顯示/隱藏) 圖示，然後選擇 Public DNS (公有 DNS)。)

步驟 4：在 CodeDeploy 中建立應用程式

在中 CodeDeploy，應用程式是包含您要部署之軟體應用程式的資源。稍後，將此應用程式搭配使用，CodePipeline 將範例應用程式部署到 Amazon EC2 執行個體的自動化。

首先，您 CodeDeploy 要建立允許執行部署的角色。接著建立 CodeDeploy 應用程式。

若要建立 CodeDeploy 服務角色

1. 在 https://console.aws.amazon.com/iam/ 開啟身分與存取權管理主控台。
2. 從主控台儀表板，選擇 Roles (角色)。
3. 選擇建立角色。
4. 在 [選取信任的實體] 下，選擇AWS 服務。在 Use case (使用案例) 中，選擇 CodeDeploy。CodeDeploy從列出的選項中進行選擇。選擇下一步。AWSCodeDeployRole 受管政策已連接至角色。
5. 選擇下一步。
6. 輸入角色的名稱 (例如 **CodeDeployRole**)，然後選擇 Create role (建立角色)。

若要在中建立應用程式 CodeDeploy

1. 請在以下位置開啟 CodeDeploy 主控台。 https://console.aws.amazon.com/codedeploy

2. 如果未顯示 [應用程式] 頁面，請在功能表上選擇 [應用程式]。
3. 選擇 建立應用程式。
4. 在 Application name (應用程式名稱) 中，輸入 **MyDemoApplication**。
5. 在 Compute Platform (運算平台) 中，選擇 EC2/On-premises (EC2/ 現場部署)。
6. 選擇 建立應用程式。

若要在中建立部署群組 CodeDeploy

部署群組是一種資源，可定義部署相關設定，例如要部署哪些執行個體以及部署這些執行個體的速度。

1. 在顯示您應用程式的頁面上，選擇 Create deployment group (建立部署群組)。
2. 在 Deployment group name (部署群組名稱) 中，輸入 **MyDemoDeploymentGroup**。
3. 在 [服務角色] 中，選擇您先前建立之服務角色的 ARN (例如**arn:aws:iam::*account_ID*:role/CodeDeployRole**)。
4. 在 Deployment type (部署類型) 下，選擇 In-place (就地進行)。
5. 在 Environment configuration (環境組態) 下，選擇 Amazon EC2 Instances (Amazon EC2 執行個體)。在「機碼」欄位中，輸入**Name**。在「值」欄位中，輸入您用來標記例證的名稱 (例如，**MyCodePipelineDemo**)。
6. 在 [使用 AWS Systems Manager 程式組態] 下，選擇 [立即] 並排 這會在執行個體上安裝代理程式。Linux 執行個體已使用 SSM 代理程式設定，現在將以代理 CodeDeploy 程式進行更新。
7. 在 Deployment configuration (部署組態) 下，選擇 `CodeDeployDefault.OneAtaTime`。
8. 在「Load Balancer」下，確定未選取「啟用負載平衡」。您不需要為此範例設定負載平衡器或選擇目標群組。
9. 選擇 Create deployment group (建立部署群組)。

步驟 5：在 CodePipeline 中建立您的第一個管道

您現在已準備好建立和執行您的第一個管道。在此步驟中，您會建立在程式碼推送至 CodeCommit 儲存庫時自動執行的管道。

建立配 CodePipeline 管

1. 請登入AWS Management Console並開啟 CodePipeline 主控台，網址為 http://console.aws.amazon.com/codesuite/codepipeline/home。

請在以下位置開啟 CodePipeline 主控台。 https://console.aws.amazon.com/codepipeline/

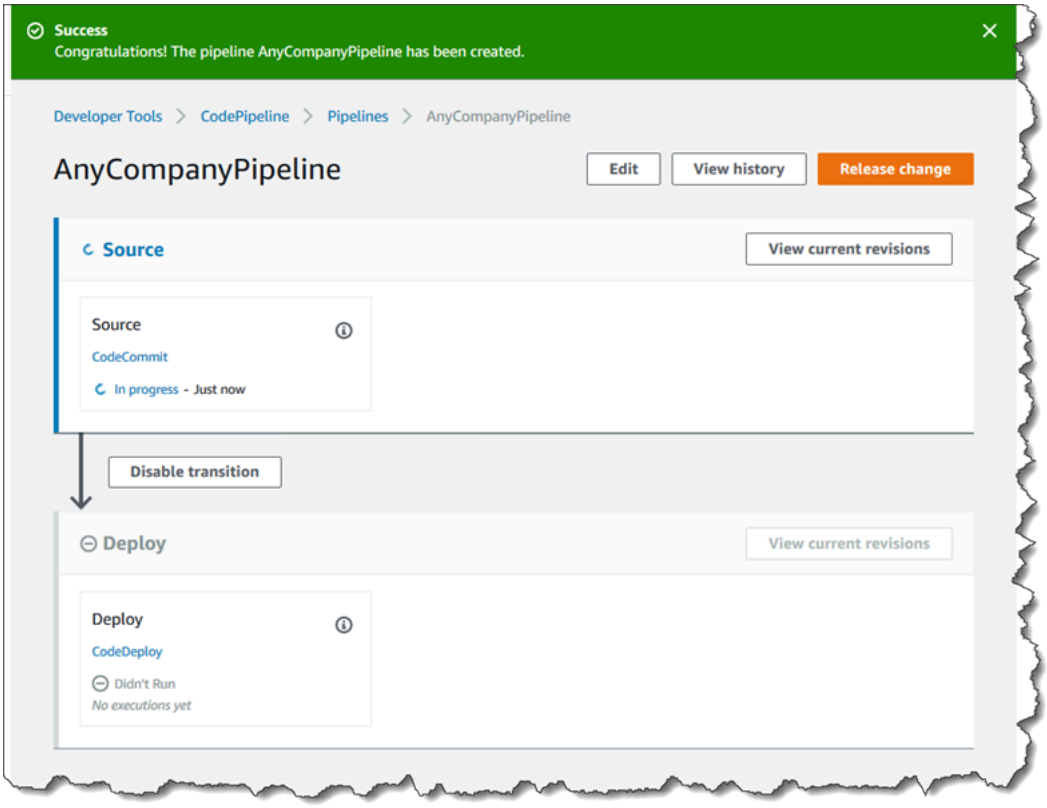
2. 選擇 Create pipeline (建立管道)。
3. 在 Step 1: Choose pipeline settings (步驟 1：選擇管道設定) 的 Pipeline name (管道名稱) 中，輸入 **MyFirstPipeline**。
4. 在管線類型中，針對本教學課程的目的選擇 V1。您也可以選擇 V2；但是請注意，管道類型在特性和價格上有所不同。如需詳細資訊，請參閱管線類型。
5. 在 [服務角色] 中，選擇 [新增服務角色] CodePipeline 以允許在 IAM 中建立服務角色。
6. 將 Advanced settings (進階設定) 下的設定保留為預設值，然後選擇 Next (下一步)。
7. 在 Step 2: Add source stage (步驟 2：新增來源階段) 的 Source provider (來源提供者) 中，選擇 CodeCommit。在存放庫名稱中，選擇您在其中建立的 CodeCommit 存放庫名稱步驟 1：建立 CodeCommit 儲存庫。在 Branch name (分支名稱) 中，選擇 `main`，然後選擇 Next step (下一個步驟)。

   選取儲存庫名稱和分支後，會出現一則訊息，顯示要為此管道建立的 Amazon E CloudWatch vents 規則。

   在 Change detection options (變更刪除選項) 下，保持預設設定。這可讓您 CodePipeline 使用 Amazon CloudWatch 事件偵測來源儲存庫中的變更。

   選擇下一步。
8. 在 Step 3: Add build stage (步驟 3：新增建置階段) 中，選擇 Skip build stage (跳過建置階段)，然後再次選擇 Skip (跳過) 來接受警告訊息。選擇下一步。

   > **Note**
   >
   > 在此教學中，您將部署無須建置服務的程式碼，因此可以略過此步驟。不過，如果在將來源碼部署到執行個體前需要先建置該來源碼，您可以在此步驟中設定 CodeBuild。
9. 在步驟 4：新增部署階段的部署提供者中，選擇CodeDeploy。在 Application name (應用程式名稱) 中，選擇 **MyDemoApplication**。在 Deployment group (部署群組) 中，選擇 **MyDemoDeploymentGroup**，然後選擇 Next step (下一個步驟)。
10. 在 Step 5: Review (步驟 5：檢閱) 中，檢閱資訊，然後選擇 Create pipeline (建立管道)。
11. 管道會在建立後開始執行。它會從您的 CodeCommit儲存庫下載程式碼，並建立 EC2 執行個體的 CodeDeploy 部署。當 CodePipeline 範例將網頁部署到 CodeDeploy 部署中的 Amazon EC2 執行個體時，您可以檢視進度、成功和失敗訊息。



恭喜您！您剛剛建立了一個簡單的管道 CodePipeline。

下一步，您會驗證結果。

驗證您的管道是否成功執行

1. 檢視管道初始進度。每一個階段狀態會從 No executions yet (尚未執行) 變更為 In Progress (進行中)，然後顯示 Succeeded (成功) 或 Failed (失敗)。該管道應會在幾分鐘內完成初次執行。
2. 顯示管線狀態的「成功」之後，請在「部署」階段的狀態區域中選擇CodeDeploy。這將打開控 CodeDeploy 制台。如果未顯示 Succeeded (成功)，請參閱 疑難排 CodePipeline。
3. 在 Deployments (部署) 標籤上，選擇部署 ID。在部署的頁面上的 Deployment lifecycle events (部署生命週期事件) 下，選擇執行個體 ID。這會開啟 EC2 主控台。
4. 在 Description (敘述) 標籤的 Public DNS (公有 DNS) 中複製地址 (例如，`ec2-192-0-2-1.us-west-2.compute.amazonaws.com`)，然後貼上至 Web 瀏覽器的地址列中。

   會顯示您下載並推送至 CodeCommit 儲存庫的範例應用程式的網頁。

如需關於階段、動作以及管道如何運作的更多資訊，請參閱 CodePipeline 概念 。

步驟 6：修改 CodeCommit 存儲庫中的代碼

您的管道已設定為隨 CodeCommit 儲存庫程式碼變更時執行。在此步驟中，您會變更屬於 CodeCommit 存放庫中範例 CodeDeploy應用程式一部分的 HTML 檔案。推送這些變更時，您的管道會再次執行，並會在您稍早存取的 Web 地址顯示您所做的變更。

1. 將目錄變更為您的本機儲存庫：

```
(For Linux, macOS, or Unix) cd /tmp/MyDemoRepo
(For Windows) cd c:\temp\MyDemoRepo
```

2. 使用文字編輯器修改 index.html 檔案：

```
(For Linux or Unix)gedit index.html
(For OS X)open -e index.html
(For Windows)notepad index.html
```

3. 修訂 index.html 檔案的內容，變更網頁的背景顏色和一些文字，然後儲存檔案。

```
<!DOCTYPE html>
<html>
<head>
  <title>Updated Sample Deployment</title>
  <style>
    body {
      color: #000000;
      background-color: #CCFFCC;
      font-family: Arial, sans-serif;
      font-size:14px;
    }

    h1 {
      font-size: 250%;
      font-weight: normal;
      margin-bottom: 0;
    }

    h2 {
      font-size: 175%;
      font-weight: normal;
      margin-bottom: 0;
    }
```

```
  </style>
</head>
<body>
  <div align="center"><h1>Updated Sample Deployment</h1></div>
  <div align="center"><h2>This application was updated using CodePipeline,
 CodeCommit, and CodeDeploy.</h2></div>
  <div align="center">
    <p>Learn more:</p>
    <p><a href="https://docs.aws.amazon.com/codepipeline/latest/
userguide/">CodePipeline User Guide</a></p>
    <p><a href="https://docs.aws.amazon.com/codecommit/latest/
userguide/">CodeCommit User Guide</a></p>
    <p><a href="https://docs.aws.amazon.com/codedeploy/latest/
userguide/">CodeDeploy User Guide</a></p>
  </div>
</body>
</html>
```

4. 一次執行下列指令，將變更提交並推送至 CodeCommit 儲存庫：

```
git commit -am "Updated sample application files"
```

```
git push
```

驗證您的管道是否成功執行

1. 檢視管道初始進度。每一個階段狀態會從 No executions yet (尚未執行) 變更為 In Progress (進行中)，然後顯示 Succeeded (成功) 或 Failed (失敗)。管道的執行應會在幾分鐘內完成。
2. 動作狀態顯示 Succeeded (成功) 後，請重新整理您先前在瀏覽器中存取的示範頁面。

   更新後的網頁隨即顯示。

步驟 7：清除資源

您可以將在此教學中建立的一些資源用在本指南的其他教學。例如，您可以重複使用 CodeDeploy 應用程式和部署。不過，在您完成這些教學課程之後，應該刪除管道以及其所使用的資源，才不會因為持續使用那些資源而付費。首先，刪除管道，然後刪除 CodeDeploy 應用程式及其關聯的 Amazon EC2 執行個體，最後刪除 CodeCommit存放庫。

清除此教學中使用的資源

1. 若要清理資 CodePipeline 源，請遵循中刪除管道中的指示AWS CodePipeline。
2. 若要清理資 CodeDeploy 源，請遵循清理部署逐步解說資源中的指示。
3. 若要刪除存 CodeCommit 放庫，請遵循刪除 CodeCommit存放庫中的指示。

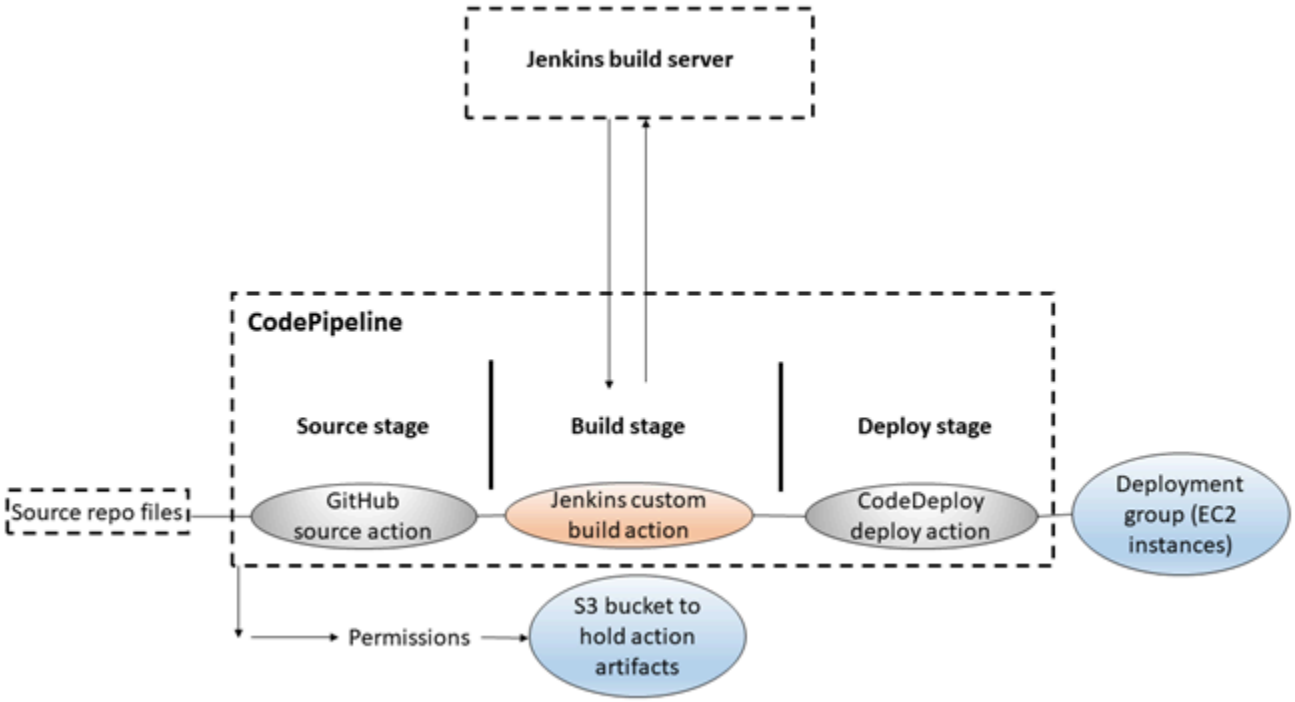
步驟 8：深入閱讀

了解有關如何 CodePipeline 工作的更多信息

- 如需關於階段、動作以及管道如何運作的更多資訊，請參閱 CodePipeline 概念 。
- 如需有關可使用執行之動作的資訊 CodePipeline，請參閱與 CodePipeline 動作類型的整合。
- 嘗試這個更進階的教學：教學：建立四階段管道。這個教學會建立多階段管道，其中包含在部署程式碼前的程式碼建置步驟。

教學：建立四階段管道

現在您已在 教學：建立簡易管道 (S3 儲存貯體)或 教學：建立範本管道 (CodeCommit 儲存庫)中建立您的第一個管道，您可以開始建立更複雜的管道。本教學課程將逐步引導您建立四階段管道，該管線使用 GitHub儲存庫作為原始碼、Jenkins 建置伺服器來建置專案，以及將建置程 CodeDeploy 式碼部署到測試伺服器的應用程式。下圖顯示了初始的三階段管道。



在建立管道之後，您將會使用測試動作來編輯管道以將之新增到階段中，藉以測試程式碼，同時也使用 Jenkins。

在建立此管道前，您必須設定必要資源。例如，如果您想要為原始程式碼使用 GitHub 儲存庫，則必須先建立儲存庫，然後才能將其新增至管線。在設定的環節中，本教學將會帶您演練如何在 EC2 執行個體上設定 Jenkins，以進行示範。

> **Important**
>
> 您在此程序中新增至管線的許多動作都涉及建立管線之前需要建立的AWS資源。 AWS來源動作的資源必須始終建立在您建立管道的相同AWS區域中。例如，如果您在美國東部 (俄亥俄) 區域建立管道，則您的 CodeCommit 存放庫必須位於美國東部 (俄亥俄) 區域。
> 您可以在建立管道時新增跨區域動作。 AWS跨區域作業的資源必須位於您計劃執行作業的相同AWS區域中。如需詳細資訊，請參閱在 CodePipeline 中新增跨區域動作。

在您開始此教學前，應已完成 開始使用 CodePipeline 中的一般先決條件。

主題

- 步驟 1：完成事前準備
- 步驟 2：在 CodePipeline 中建立管道

步驟 1：完成事前準備

要與詹金斯集成，AWS CodePipeline要求您在要使用的詹金斯的任何實例上安裝 CodePipeline 插件詹金斯。 CodePipeline您也應該設定專用的 IAM 使用者或角色，以用於 Jenkins 專案與 CodePipeline. 整合 Jenkins 的最簡單方法 CodePipeline 是在 EC2 執行個體上安裝 Jenkins，該執行個體使用您為 Jenkins 整合建立的 IAM 執行個體角色。為讓在 Jenkins 動作管道內的連結能夠成功連線，您必須在伺服器或 EC2 執行個體上進行代理與防火牆設定，以允許您的 Jenkins 專案使用送入至連接埠的連線。請確定您在允許對那些連接埠 (例如若您限制 Jenkins 只使用 HTTPS 連線則為 443 與 8443，或者若您允許 HTTPS 連線則是 80 與 8080) 的連線前，已設定 Jenkins 來驗證使用者身分並強化存取控制。如需詳細資訊，請參閱保護 Jenkins 安全。

> **Note**
>
> 此教學使用程式碼範例並設定將範本自 Haml 轉換為 HTML 的組建步驟。您可以按照中的步驟從存放 GitHub 庫下載開放原始碼範例程式碼將範例複製或複製到 GitHub 儲存庫。您將需要 GitHub 存放庫中的整個樣本，而不僅僅是 .zip 文件。
> 此教學也假設：
>
> - 您對於安裝及管理 Jenkins 還有建立 Jenkins 專案的操作非常熟悉。
> - 您已在代管您的 Jenkins 專案的相同電腦或執行個體上安裝適用於 Ruby 的 Rake 與 Haml Gem。
> - 您已設定必要的系統環境變數，讓 Rake 命令可從終端機或命令列執行 (例如在 Windows 系統上，修改 PATH 變數以加入您安裝了 Rake 的目錄)。

主題

將範例複製或複製到 GitHub 儲存庫

若要複製範例並推送至 GitHub 儲存庫

1. 從 GitHub 存放庫下載範例程式碼，或將存放庫複製到您的本機電腦。有兩種範本套件：
   - 如果您要將範例部署到 Amazon Linux、RHEL 或 Ubuntu 伺服器執行個體，請選擇 codepipeline-jenkins-aws-codedeploy_linux.zip。
   - 如果您要將範例部署到 Windows 伺服器執行個體，請選擇 [CodePipeline詹金斯-AWSCodeDeploy_Windows .zip]。
2. 從儲存庫中，選擇 Fork (延伸) 來複製範本儲存庫到 Github 帳戶中的儲存庫。如需詳細資訊，請參閱 GitHub 文件。

建立 IAM 角色以用於詹金斯整合

最佳做法是考慮啟動 EC2 執行個體來託管 Jenkins 伺服器，並使用 IAM 角色授與 CodePipeline執行個體互動所需的權限。

1. 登入 AWS Management Console，並開啟位於 https://console.aws.amazon.com/iam/ 的 IAM 主控台。
2. 在 IAM 主控台的導覽面板中，選擇角色，然後選擇建立角色。
3. 在 Select type of trusted entity (選取可信任執行個體類型) 下，選擇 AWS 服務。在 Choose the service that will use this role (選擇將使用此角色的服務) 下，選擇 EC2。在 Select your use case (選取您的使用案例) 下，選擇 EC2。
4. 選擇 Next: Permissions (下一步：許可)。在 Attach permissions policies (附加許可政策) 頁面上，選取 `AWSCodePipelineCustomActionAccess` 受管政策，然後選擇 Next: Tags (下一步：標籤)。選擇 下一步：檢閱。
5. 在 Review (檢閱) 頁面的 Role name (角色名稱) 中，輸入特別為 Jenkins 整合建立的角色名稱 (例如 *`JenkinsAccess`*)，然後選擇 Create role (建立角色)。

當您建立將會安裝 Jenkins 的 EC2 執行個體時，請在 Step 3: Configure Instance Details (步驟 3：設定執行個體詳細資訊) 中確認您選擇了執行個體角色 (例如 *`JenkinsAccess`*)。

如需執行個體角色和 Amazon EC2 的詳細資訊，請參閱 Amazon EC2 的 IAM 角色、使用 IAM 角色將許可授予在 Amazon EC2 執行個體上執行的應用程式，以及建立角色以將許可委派給 AWS 服務.

安裝和配置詹金斯和 CodePipeline 插件詹金斯

要安裝詹金斯和 CodePipeline 插件詹金斯

1. 建立您將會安裝 Jenkins 的 EC2 執行個體，並在 Step 3: Configure Instance Details (步驟 3：設定執行個體詳細資訊) 中確認您選擇了所建立的執行個體角色 (例如 *JenkinsAccess*)。如需建立 EC2 執行個體的詳細資訊，請參閱 Amazon EC2 使用者指南中的啟動 Amazon EC2 執行個體。

   > **Note**
   >
   > 如果您已經擁有要使用的 Jenkins 資源，則可以這樣做，但必須建立特殊的 IAM 使用者，將AWSCodePipelineCustomActionAccess受管政策套用至該使用者，然後在 Jenkins 資源上設定和使用該使用者的存取登入資料。如果您想要使用 Jenkins UI 來供應登入資料，請設定 Jenkins 為僅允許使用 HTTPS。如需詳細資訊，請參閱疑難排 CodePipeline。

2. 在 EC2 執行個體上安裝 Jenkins。如需更多資訊，請參閱說明安裝 Jenkins 與啟動並存取 Jenkins 的 Jenkins 文件，以及 產品與服務整合 CodePipeline 中的 details of integration with Jenkins。
3. 啟動 Jenkins，然後在首頁上選擇 Manage Jenkins (管理 Jenkins)。
4. 在 Manage Jenkins (管理 Jenkins) 頁面上，選擇 Manage Plugins (管理外掛程式)。
5. 選擇 Available (可用) 標籤，並在 Filter (篩選條件) 搜尋方塊中輸入 **`AWS CodePipeline`**。從列表中選擇 Jenkins 的CodePipeline 插件，然後選擇立即下載並在重新啟動後安裝。
6. 在 Installing Plugins/Upgrades (安裝外掛程式/升級) 頁面上，選擇 Restart Jenkins when installation is complete and no jobs are running (安裝完成且沒有正在執行的工作時重新啟動 Jenkins)。
7. 選擇 Back to Dashboard (返回儀表板)。
8. 在主要頁面上，選擇 New Item (新項目)。
9. 在項目名稱中，輸入 Jenkins 專案的名稱 (例如，*MyDemoProject*)。選擇 Freestyle project (自由形式專案)，然後選擇 OK (確定)。

   > **Note**
   >
   > 請確認您的專案名稱符合 CodePipeline 的要求。如需詳細資訊，請參閱AWS CodePipeline 中的配額。

10. 在專案的組態頁面上，勾選 Execute concurrent builds if necessary (若需要請執行同時進行的組建) 核取方塊。 在 Source Code Management (原始程式碼管理) 中，選擇 AWS CodePipeline。如果您已在 EC2 執行個體上安裝 Jenkins，並使用您為整合而建立的 IAM 使用者設定檔 CodePipeline 和 Jenkins，請將所有其他欄位保留空白。AWS CLI
11. 選擇 [進階]，然後在 [提供者] 中輸入動作提供者的名稱 CodePipeline (例如 ***MyJenkinsProviderName***)。請確認此名稱為唯一且易記。您將會在此教學後面部分中將組建動作新增到管道，然後會在新增測試動作時在次重複此操作。

    > **Note**
    >
    > 此動作名稱必須符合在 CodePipeline 中的動作命名要求。如需詳細資訊，請參閱AWS CodePipeline 中的配額。

12. 在 Build Triggers (組建觸發) 中，清除任何核取方塊，然後選擇 Poll SCM (輪詢 SCM)。在 Schedule (排程) 中，輸入五個星號，並以空格間隔，如下所示：

    ```
    * * * * *
    ```

    這次民意調查 CodePipeline 每分鐘。
13. 在 Build (組建) 中，選擇 Add build step (新增組建步驟)。選擇執行命令介面 (Amazon Linux、RHEL 或 Ubuntu 伺服器) 執行批次命令 (視窗伺服器)，然後輸入以下內容：

    ```
    rake
    ```

    > **Note**
    >
    > 請確認您的環境使用執行 Rake 所需的變數及設定來配置；否則組建將會失敗。

14. 選擇 [新增建置後動作]，然後選擇 [發行AWS CodePipeline 者]。選擇 Add (新增)，然後在 Build Output Locations (組建輸出位置) 中，將位置保留為空白。此組態為預設。將會在組建程結束時建立壓縮檔。
15. 選擇 Save (儲存) 來儲存您的 Jenkins 專案。

步驟 2：在 CodePipeline 中建立管道

在此部分的教學中，您將會使用 Create Pipeline (建立管道) 精靈來建立管道。

建立 CodePipeline 自動發行程序

1. 請登入AWS Management Console並開啟 CodePipeline 主控台，網址為 http://console.aws.amazon.com/codesuite/codepipeline/home。
2. 如有需要，可使用區域選擇器，將區域變更為您的管道資源所在的區域。例如，如果您已在中建立上一個教學課程的資源`us-east-2`，請確定 [區域] 選取器已設定為美國東部 (俄亥俄州)。

   如需有關可用之區域和端點的詳細資訊 CodePipeline，請參閱AWS CodePipeline端點和配額。
3. 在 Welcome (歡迎) 頁面、Getting started (入門) 頁面、或者 Pipelines (管道) 頁面上，選擇 Create pipeline (建立管道)。
4. 在 Step 1: Choose pipeline settings (步驟 1：選擇管道設定) 頁面的 Pipeline name (管道名稱) 中輸入管道的名稱。
5. 在管線類型中，針對本教學課程的目的選擇 V1。您也可以選擇 V2；但是請注意，管道類型在特性和價格上有所不同。如需詳細資訊，請參閱管線類型。
6. 在 [服務角色] 中，選擇 [新增服務角色] CodePipeline 以允許在 IAM 中建立服務角色。
7. 將 Advanced settings (進階設定) 下的設定保留為預設值，然後選擇 Next (下一步)。
8. 在「步驟 2：新增來源階段」頁面的「來源提供者」中，選擇GitHub。
9. 在「連線」下，選擇現有的連線或建立新的連線。若要建立或管理來 GitHub 源動作的連線，請參閱GitHub 連接。
10. 在 Step 3: Add build stage (步驟 3：新增組建階段)，選擇 Add Jenkins (新增 Jenkins)。在提供者名稱中，輸入您在 Jenkins 的 CodePipeline 外掛程式中提供的動作名稱 (例如 *`MyJenkinsProviderName`*)。此名稱必須完全符合 Jenkins CodePipeline 外掛程式中的名稱。在 Server URL (伺服器 URL)，輸入 Jenkins 安裝的 EC2 執行個體。在 [專案名稱] 中，輸入您在 Jenkins 中建立的專案名稱，例如 *`MyDemoProject`*，然後選擇 [下一步]。
11. 在步驟 4：新增部署階段中，重複使用您在中建立的 CodeDeploy 應用程式和部署群組教學：建立簡易管道 (S3 儲存貯體)。在部署提供者中，選擇CodeDeploy。在 Application name (應用程式名稱) 中，輸入 **`CodePipelineDemoApplication`**，或選擇 refresh (重新整理) 按鈕，然後從清單中選擇應用程式名稱。在 Deployment group (部署群組) 中，輸入 **`CodePipelineDemoFleet`**，或從清單中選擇，然後選擇 Next (下一步)。

    > **Note**
    >
    > 您可以使用自己的 CodeDeploy 資源或建立新資源，但可能會產生額外費用。

12. 在 Step 5: Review (步驟 5：檢閱) 中，檢閱資訊，然後選擇 Create pipeline (建立管道)。

13. 管道會自動在管道中啟動並執行範本。當管道將 Haml 範例建置為 HTML，並將網頁部署到部署中的每個 Amazon EC2 執行個體時，您可以檢視進度和成功和失敗訊息。 CodeDeploy

步驟 3：新增另一個階段至您的管道

現在您將建立測試階段然後和測試動作到階段中，該階段使用包含在範本中的 Jenkins 測試來判定網頁是否有任何內容。此測試僅用於示範用途。

> **Note**
>
> 如果您不希望新增另一個階段到您的管道，您可以在部署動作之前或之後新增測試動作到管道的 Staging (預備) 階段。

新增測試階段到管道

主題

- 查詢執行個體的 IP 地址
- 建立用於測試部署的 Jenkins 專案
- 建立第四個階段

查詢執行個體的 IP 地址

驗證您想要部署程式碼的執行個體 IP 地址

1. 在該管道狀態顯示 Succeeded (成功) 後，在 Staging (預備) 階段的狀態區域中，選擇 Details (詳細資訊)。
2. 在部署詳細資訊區段中，在執行個體 ID 中選擇其中一個成功部署的執行個體中的執行個體 ID。
3. 複製執行個體的 IP 地址 (例如，*192.168.0.4*)。您將會在 Jenkins 測試中使用此 IP 地址。

建立用於測試部署的 Jenkins 專案

建立 Jenkins 專案

1. 在您安裝了 Jenkins 的執行個體上開啟 Jenkins，並從首頁選擇 New Item (新項目)。

2. 在項目名稱中，輸入 Jenkins 專案的名稱 (例如，*MyTestProject*)。選擇 Freestyle project (自由形式專案)，然後選擇 OK (確定)。

   > **Note**
   >
   > 請確定專案的名稱符合 CodePipeline 需求。如需詳細資訊，請參閱AWS CodePipeline 中的配額。

3. 在專案的組態頁面上，勾選 Execute concurrent builds if necessary (若需要請執行同時進行的組建) 核取方塊。 在 Source Code Management (原始程式碼管理) 中，選擇 AWS CodePipeline。如果您已在 EC2 執行個體上安裝 Jenkins，並使用您為整合而建立的 IAM 使用者設定檔 CodePipeline 和 Jenkins，請將所有其他欄位保留空白。AWS CLI

   > **Important**
   >
   > 如果您正在設定 Jenkins 專案，但該專案並未安裝在 Amazon EC2 執行個體上，或者安裝在執行 Windows 作業系統的 EC2 執行個體上，請根據代理主機和連接埠設定的要求完成欄位，並提供 IAM 使用者或角色的登入資料，以便在 Jenkins 和之間進行整合。CodePipeline

4. 選擇 Advanced (進階)，並在 Category (目錄) 中選擇 Test (測試)。

5. 在提供者中，輸入您用於組建專案的相同名稱 (例如，*MyJenkinsProviderName*)。在此教學後面部分中，您將會在新增測試動作到管道時使用此名稱。

   > **Note**
   >
   > 此名稱必須符合動作的 CodePipeline 命名需求。如需詳細資訊，請參閱AWS CodePipeline 中的配額。

6. 在 Build Triggers (組建觸發) 中，清除任何核取方塊，然後選擇 Poll SCM (輪詢 SCM)。在 Schedule (排程) 中，輸入五個星號，並以空格間隔，如下所示：

   ```
   * * * * *
   ```

   這次民意調查 CodePipeline 每分鐘。

7. 在 Build (組建) 中，選擇 Add build step (新增組建步驟)。如果您要部署到 Amazon Linux、RHEL 或 Ubuntu 伺服器執行個體，請選擇執行殼層。接著，輸入下列內容，其中 IP 地址為您之前複製的 EC2 執行個體地址：

```
TEST_IP_ADDRESS=192.168.0.4 rake test
```

如果您要部署到 Windows Server 執行個體，請選擇執行批次命令，然後輸入下列命令，其中 IP 位址是您之前複製的 EC2 執行個體的位址：

```
set TEST_IP_ADDRESS=192.168.0.4 rake test
```

> **Note**
>
> 測試假設預設連接埠為 80。若您想要指定不同的連接埠，請新增測試連接埠陳述式，如下所示：
>
> ```
> TEST_IP_ADDRESS=192.168.0.4 TEST_PORT=8000 rake test
> ```

8. 選擇 [新增建置後動作]，然後選擇 [發行AWS CodePipeline 者]。請勿選擇 Add (新增)。
9. 選擇 Save (儲存) 來儲存您的 Jenkins 專案。

建立第四個階段

新增階段到內含 Jenkins 測試動作的管道

1. 請登入AWS Management Console並開啟 CodePipeline 主控台，網址為 http://console.aws.amazon.com/codesuite/codepipeline/home。
2. 在 Name (名稱) 中，選擇您建立的管道名稱， MySecondPipeline。
3. 在管道詳細資訊頁面上，選擇 Edit (編輯)。
4. 在 Edit (編輯) 頁面上，選擇 + Stage (+ 階段) 以在 Build (建置) 階段後立即新增一個階段。
5. 在新階段的名稱欄位中，輸入名稱 (例如 **`Testing`**)，然後選擇 + Add action group (+ 新增動作群組)。
6. 在動作名稱中，輸入 *`MyJenkinsTest-##`*。在測試提供者中，選擇您在 Jenkins 中指定的提供者名稱 (例如，*`MyJenkinsProviderName`*)。在「專案名稱」中，輸入您在 Jenkins 中建立

的專案名稱 (例如 *MyTestProject*)。在 [輸入成品] 中，從預設名稱為 Jenkins 組建中選擇成品 *BuildArtifact*，然後選擇 [完成]。

> **Note**
>
> 由於 Jenkins 測試動作會在 Jenkins 建置步驟中建置的應用程式上運作，請使用建置成品作為測試動作的輸入成品。

如需有關輸入和輸出成品以及管道結構的詳細資訊，請參閱 CodePipeline 配管結構參照。

7. 在 Edit (編輯) 頁面上，選擇 Save pipeline changes (儲存管道變更)。在 Save pipeline changes (儲存管道變更) 對話方塊中，選擇 Save and continue (儲存並繼續)。
8. 雖然新階段已新增至您的管道，但由於沒有發生觸發另一個管道執行的變更，所以會顯示該階段為 No executions yet (尚未執行)。若要透過修訂後的管線執行範例，請在管線詳細資訊頁面上選擇「發行變更」。

   管道檢視會顯示管道中的階段與動作，以及在那四個階段間執行的版本狀態。管道執行所有階段所需花費的時間將根據成品大小、組建與測試動作的複雜度、以及其他因素而定。

步驟 4：清除資源

在您完成本教學之後，您應該刪除管道以及其所使用的資源，如此您才不會因為持續使用那些資源而付費。如果您不打算繼續使用，請刪除管道 CodePipeline，然後刪除 CodeDeploy 應用程式及其關聯的 Amazon EC2 執行個體，最後刪除用於存放成品的 Amazon S3 儲存貯體。如果您不打算繼續使用它們，也應該考慮是否刪除其他資源，例如 GitHub 存儲庫。

清除此教學中使用的資源

1. 在本機 Linux、macOS 或 Unix 電腦上開啟終端機工作階段，或在本機 Windows 電腦上開啟命令提示字元，然後執行delete-pipeline指令以刪除您建立的管線。對於 **MySecondPipeline**，建議您輸入下列命令：

   ```
   aws codepipeline delete-pipeline --name "MySecondPipeline"
   ```

   此命令不會傳回任何結果。
2. 若要清理您的 CodeDeploy 資源，請依照清理中的指示操作。

3. 若要清除您的執行個體資源，請刪除您安裝 Jenkins 的 EC2 執行個體。如需詳細資訊，請參閱清理您的執行個體。
4. 如果您不打算建立更多管道或 CodePipeline 再次使用，請刪除用於存放管道成品的 Amazon S3 儲存貯體。若要刪除儲存貯體，請按照刪除儲存貯體中的說明進行。
5. 若您不想要再次使用此管道的其他資源，請考慮依照該特定資源的說明來刪除這些資源。例如，如果您要刪除 GitHub 存放庫，請按照刪除 GitHub 網站上的存放庫中的指示進行操作。

教學課程：設定 CloudWatch Events 規則以接收管道狀態變更的電子郵件通知

設置管道後，在AWS CodePipeline中，您可以設定 CloudWatch Events 規則，在您管道的執行狀態有任何變更時，或是在您管道的階段或動作中時傳送通知。如需使用 CloudWatch Events 設定管道狀態變更通知的詳細資訊，請參監控 CodePipeline 事件。

在本教學中，您會設定通知，在管道狀態變更為 FAILED (失敗) 時傳送電子郵件。本教學會在建立 CloudWatch Events 規則時使用輸入變壓器方法。它會將訊息結構描述詳細資訊轉換成可供人閱讀的文字訊息。

> **Note**
>
> 在您建立本教學的資源時，例如 Amazon SNS 通知和 CloudWatch Events 規則時，請確定在相同AWS區域作為您的管道。

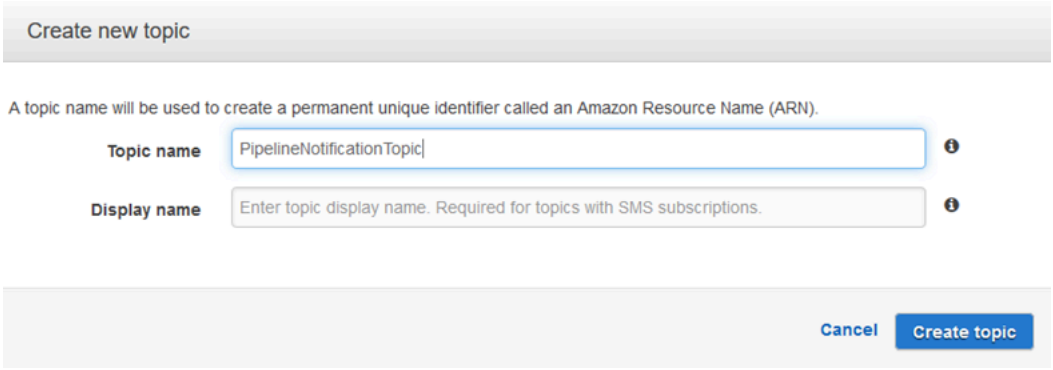
主題

步驟 1：使用 Amazon SNS 設定電子郵件通知

Amazon SNS 會協調使用主題，將訊息交付給訂端點或用户端。使用 Amazon SNS 建立通知主題，然後使用您的電子郵件地址訂主題。Amazon SNS 主題會作為目標新增至您 CloudWatch Events 規則。如需詳細資訊，請參閱《Amazon Simple Notification Service 開發人員指南》。

在 Amazon SNS 中建立或確定一個主題。CodePipeline 會使用 CloudWatch Events 通知通知通知通知通知通知通知通知會通知至此主題。建立主題：

1. 在開啟 Amazon SNS 主控台https://console.aws.amazon.com/sns。
2. 請選擇 Create topic (建立主題)。
3. 在 Create new topic (建立新主題) 對話方塊中，針對 Topic name (主題名稱)，輸入主題的名稱 (例如 **`PipelineNotificationTopic`**)。



4. 請選擇 Create topic (建立主題)。

   如需詳細資訊，請參閱「」建立主題中的Amazon SNS 開發人員指南。

讓一或多個收件人訂閱主題來接收電子郵件通知。讓收件人訂閱主題：

1. 在 Amazon SNS 主控台中，從主題列表中，選取您新主題旁邊的核取方塊。選擇 Actions, Subscribe to topic (動作、訂閱主題)。
2. 在 Create subscription (建立訂閱) 對話方塊中，確認 ARN 有出現在 Topic ARN (主題 ARN) 中。
3. 對於 Protocol (通訊協定)，選擇 Email (電子郵件)。
4. 針對 Endpoint (端點)，輸入收件人的完整電子郵件地址。
5. 選擇 Create Subscription (建立訂閱)。
6. Amazon SNS 會將訂確認電子郵件傳送給收件人。若要接收電子郵件通知，收件人必須選擇此電子郵件中的 Confirm subscription (確認訂閱) 連結。在收件人按一下連結後，若訂成功，Amazon SNS 會在收件人的 Web 瀏覽器中顯示確認訊息。

   如需詳細資訊，請參閱「」訂閱主題中的Amazon SNS 開發人員指南。

步驟 2：建立規則並將 SNS 主題新增為目標

使用 CodePipeline 建立 CloudWatch Events 通知規則，作為事件源。

1. 透過 https://console.aws.amazon.com/cloudwatch/ 開啟 CloudWatch 主控台。
2. 在導覽窗格中，選擇 Events (事件)。
3. 選擇 Create rule (建立規則)。在 Event source (事件來源) 下，選擇 AWS CodePipeline。針對事件類型，選擇管道執行狀態變更。
4. 選擇 Specific state(s) (特定狀態)，然後選擇 **FAILED**。
5. 選擇 Edit (編輯)，開啟 Event Pattern Preview (事件模式預覽) 窗格的 JSON 編輯器。使用您管道的名稱新增 **pipeline** 參數，如下列名為 "myPipeline" 管道的範例所示。

   您可以複製此處的事件模式並將其貼到主控台中：

```
{
  "source": [
    "aws.codepipeline"
  ],
  "detail-type": [
    "CodePipeline Pipeline Execution State Change"
  ],
  "detail": {
    "state": [
      "FAILED"
    ],
    "pipeline": [
      "myPipeline"
    ]
  }
}
```

6. 在 Targets (目標) 中，選擇 Add target (新增目標)。
7. 在目標清單中，選擇 SNS topic (SNS 主題)。針對 Topic (主題)，輸入您建立的主題。
8. 展開 Configure input (設定輸入)，然後選擇 Input Transformer (輸入轉換器)。
9. 在 Input Path (輸入路徑) 方塊中，輸入下列鍵/值對。

```
{ "pipeline" : "$.detail.pipeline" }
```

   在 Input Template (輸入範本) 方塊中，輸入下列內容：

```
"The Pipeline <pipeline> has failed."
```

10. 選擇 Configure details (設定詳細資訊)。

11. 在 Configure rule details (設定規則詳細資訊) 頁面上，輸入名稱及選擇性描述。針對 State (狀態)，將 Enabled (啟用) 保留在選取狀態。
12. 選擇 Create rule (建立規則)。
13. 確認 CodePipeline 現在傳送建置通知。例如，檢查您的收件匣中是否有組建通知電子郵件。
14. 若要變更規則的行為，請在 CloudWatch 主控台中，選擇規則，然後選擇動作、Edit (編輯)。編輯規則，選擇 Configure details (設定詳細資訊)，然後選擇 Update rule (更新規則)。

    若要停止使用規則來傳送建置通知，請在 CloudWatch 主控台中，選擇規則，然後選擇動作、Disable。

    若要刪除規則，請在 CloudWatch 主控台中，選擇規則，然後選擇動作、刪除。

步驟 3：清除 資源

在您完成本教學之後，您應該刪除管道以及其所使用的資源，如此您才不會因為持續使用那些資源而付費。

如需如何清理 SNS 通知及刪除 Amazon CloudWatch Events 規則的資訊，請參清理（取消訂閱 Amazon SNS 主題）和參考`DeleteRule`中的Amazon CloudWatch Events API 參考。

教程：創建一個管道，用於構建和測試您的 Android 應用程序 AWS Device Farm

您可以使用 AWS CodePipeline 來配置持續集成流程，在每次推送提交時構建和測試應用程序。本教程介紹瞭如何創建和配置管道，以使用 GitHub 存儲庫中的源代碼構建和測試 Android 應用程序。管道會偵測到新 GitHub 提交的到來，然後使CodeBuild用建置應用程式和 Device Farm 來測試它。

> **Important**
>
> 您在此程序中新增至管線的許多動作都涉及建立管線之前需要建立的 AWS 資源。 AWS 來源動作的資源必須始終建立在您建立管道的相同 AWS 區域中。例如，如果您在美國東部 (俄亥俄) 區域建立管道，則您的 CodeCommit 存放庫必須位於美國東部 (俄亥俄) 區域。
> 您可以在建立管道時新增跨區域動作。 AWS 跨區域作業的資源必須位於您計劃執行作業的相同「 AWS 區域」中。如需詳細資訊，請參閱 在 CodePipeline 中新增跨區域動作。

您可以使用現有的 Android 應用程序和測試定義來嘗試此操作，也可以使用 Device Farm 提供的示例應用程序和測試定義。

> **Note**

開始之前

1. 登入 AWS Device Farm 主控台，然後選擇 [建立新專案]。
2. 選擇您的專案。在瀏覽器中，複製新專案的 URL。URL 包含專案 ID。
3. 複製並保留此專案 ID。您會將其用於在 CodePipeline 中建立管道。

   這裡有專案的 URL 範例。若要擷取專案 ID，請複製 `projects/` 後面的值。在此範例中，專案 ID 為 `eec4905f-98f8-40aa-9afc-4c1cfexample`。

   ```
   https://<region-URL>/devicefarm/home?region=us-west-2#/projects/
   eec4905f-98f8-40aa-9afc-4c1cfexample/runs
   ```

設定 CodePipeline 為使用 Device Farm 測試

1. 添加並提交`buildspec.yml`在應用程序代碼根目錄中調用的文件，並將其推送到存儲庫中。CodeBuild 使用此文件執行命令並訪問構建應用程序所需的成品。

   ```
   version: 0.2

   phases:
     build:
       commands:
         - chmod +x ./gradlew
         - ./gradlew assembleDebug
   artifacts:
     files:
        - './android/app/build/outputs/**/*.apk'
     discard-paths: yes
   ```

2. (選擇性) 若您使用 Calabash 或 Appium 測試您的應用程式，請將測試定義檔案新增至您的儲存庫。在稍後的步驟中，您可以將 Device Farm 設定為使用定義來執行測試套件。

如果您使用 Device Farm 內建測試，則可以略過此步驟。

3. 若要建立您的管道及新增來源階段，請執行下列作業：

   a. 請登入 AWS Management Console 並開啟 CodePipeline 主控台，網址為 https://console.aws.amazon.com/codepipeline/。

   b. 選擇 Create pipeline (建立管道)。在 Step 1: Choose pipeline settings (步驟 1：選擇管道設定) 頁面的 Pipeline name (管道名稱) 中輸入管道的名稱。

   c. 在管線類型中，針對本教學課程的目的選擇 V1。您也可以選擇 V2；但是請注意，管道類型在特性和價格上有所不同。如需詳細資訊，請參閱 管線類型。

   d. 在 Service role (服務角色) 中，讓 New service role (新服務角色) 維持在選取狀態，然後讓 Role name (角色名稱) 維持不變。若您已擁有現有服務角色，您也可以選擇使用它。

   > **Note**
   >
   > 如果您使用在 2018 年 7 月之前建立的 CodePipeline 服務角色，則需要新增 Device Farm 的權限。若要這麼做，請開啟 IAM 主控台、尋找角色，然後將下列權限新增至角色的政策。如需詳細資訊，請參閱 將許可新增至 CodePipeline 服務角色。
   >
   > ```
   > {
   >      "Effect": "Allow",
   >      "Action": [
   >         "devicefarm:ListProjects",
   >         "devicefarm:ListDevicePools",
   >         "devicefarm:GetRun",
   >         "devicefarm:GetUpload",
   >         "devicefarm:CreateUpload",
   >         "devicefarm:ScheduleRun"
   >      ],
   >      "Resource": "*"
   > }
   > ```

   e. 將 Advanced settings (進階設定) 下的設定保留為預設值，然後選擇 Next (下一步)。

   f. 在「步驟 2：新增來源階段」頁面的「來源提供者」中，選擇GitHub。

   g. 在「連線」下，選擇現有的連線或建立新的連線。若要建立或管理來 GitHub 源動作的連線，請參閱GitHub 連接。

   h. 在 Repository (儲存庫) 中，選擇來源儲存庫。

   i. 在 Branch (分支) 中，選擇您希望使用的分支。

j. 保留來源動作的其餘預設值。選擇 Next (下一步)。

4. 在 Add build stage (新增建置階段) 中，新增建置階段：

   a. 在 Build provider (建置供應商) 中，選擇 AWS CodeBuild。允許 Region (區域) 預設為管道區域。

   b. 選擇建立專案。

   c. 在 Project name (專案名稱) 中，輸入此建置專案的名稱。

   d. 在 Environment image (環境映像) 中，選擇 Managed image (受管映像)。針對 Operating system (作業系統)，選擇 Ubuntu。

   e. 針對 Runtime (執行時間)，選擇 Standard (標準)。針對映像，選擇 aws/codebuild/standard:5.0。

      CodeBuild 使用此操作系統映像，其中安裝了 Android 工作室，來構建您的應用程序。

   f. 對於服務角色，請選擇現有的 CodeBuild 服務角色或建立新的服務角色。

   g. 對於 Build specifications (建置規格)，選擇 Use a buildspec file (使用 buildspec 檔案)。

   h. 選擇「繼續」 CodePipeline。這將返回到 CodePipeline控制台並創建一個 CodeBuild 項目，該項目使用存儲庫`buildspec.yml`中的配置。組建專案會使用服務角色來管理 AWS 服務 權限。此步驟可能需要數分鐘。

   i. 選擇下一步。

5. 在 Step 4: Add deploy stage (步驟 4：新增部署階段) 頁面上，選擇 Skip deploy stage (跳過部署階段)，然後再次選擇 Skip (跳過) 來接受警告訊息。選擇下一步。

6. 在 Step 5: Review (步驟 5：檢閱) 上，選擇 Create pipeline (建立管道)。您應該會看到圖表，顯示該來源及建置階段。

7. 將 Device Farm 測試動作新增至您的管道：

   a. 在右上角，選擇 Edit (編輯)。

   b. 在圖表的底部，選擇 + Add stage (+ 新增階段)。在 Stage name (階段名稱) 中，輸入名稱，例如 **`Test`**。

   c. 選擇 + Add action group (+ 新增動作群組)。

   d. 在 Action name (動作名稱) 中，輸入名稱。

   e. 在動作提供者中，選擇 AWS Device Farm 允許 Region (區域) 預設為管道區域。

   f. 在 Input artifacts (輸入成品) 中，選擇與測試階段之前的階段輸出成品相符的輸入成品，例如 `BuildArtifact`。

在 AWS CodePipeline 主控台中，您可以將游標暫留在管線圖中的資訊圖示上，找到每個階段的輸出成品名稱。如果您的管道直接從「來源」階段測試您的應用程式，請選擇SourceArtifact。如果管線包含「建置」階段，請選擇BuildArtifact。

g. 在中 ProjectId，輸入您的 Device Farm 專案 ID。使用本教學課程開頭的步驟，擷取您的專案 ID。

h. 在中 DevicePoolArn，輸入裝置集區的 ARN。若要取得專案的可用裝置集區 ARN (包括常用裝置的 ARN)，請使用 AWS CLI 輸入下列命令：

```
aws devicefarm list-device-pools --arn arn:aws:devicefarm:us-
west-2:account_ID:project:project_ID
```

i. 在中 AppType，輸入安卓系統。

以下是有效值的清單 AppType：

- iOS
- Android
- Web

j. 在 App (應用程式) 中，輸入已編譯的應用程式套件路徑。路徑為相對於測試階段輸入成品根的相對路徑。通常，此路徑與 `app-release.apk` 相似。

k. 在中TestType輸入測試類型，然後在「測試」中輸入測試定義檔案的路徑。路徑為相對於您測試輸入成品根的相對路徑。

以下是有效值的清單 TestType：

- 爪哇胡尼特
- Appium_ 爪哇測試
- APPIUM 節點
- 蘋果紅寶石
- 蟒蛇
- 阿皮亚姆网站 JA_JUnit
- 蘋果網站測試
- 应用网节点
- 蘋果紅寶石
- 阿皮亞網蟒

- 內建模糊
- INSTRUMENTATION
- XCTEST
- XCTEST_UI

> **Note**
>
> 不支援自訂環境節點。

l. 在剩餘欄位中，提供適用於您測試及應用程式類型的組態。

m. (選擇性) 在 Advanced (進階) 中，提供您測試執行的組態資訊。

n. 選擇儲存。

o. 在您編輯的階段上，選擇 Done (完成)。在 AWS CodePipeline 窗格中，選擇 Save (儲存)，然後在警告訊息中選擇 Save (儲存)。

p. 若要提交您的變更並啟動管道建置，請選擇 Release change (發行變更)，然後選擇 Release (發行)。

教學課程：建立測試 iOS 應用程式的管道 AWS Device Farm

您可使用 AWS CodePipeline 輕鬆設定持續整合流程，於每次來源儲存貯體變更時測試您的應用程式。本教學課程示範如何建立及設定管道，以從 S3 儲存貯體測試您建置的 iOS 應用程式。管道會偵測透過 Amazon E CloudWatch vents 儲存的變更到達，然後使用 Device Farm 測試建置的應用程式。

> **Important**
>
> 您在此程序中新增至管線的許多動作都涉及建立管線之前需要建立的AWS資源。 AWS來源動作的資源必須始終建立在您建立管道的相同AWS區域中。例如，如果您在美國東部 (俄亥俄) 區域建立管道，則您的 CodeCommit 存放庫必須位於美國東部 (俄亥俄) 區域。
> 您可以在建立管道時新增跨區域動作。 AWS跨區域作業的資源必須位於您計劃執行作業的相同「AWS區域」中。如需詳細資訊，請參閱在 CodePipeline 中新增跨區域動作。

您可以透過使用您現有的 iOS 應用程式，或可使用範例 iOS 應用程式來試用。

> **Note**

開始之前

1. 登入 AWS Device Farm 主控台，然後選擇 Create a new project (建立新專案)。
2. 選擇您的專案。在瀏覽器中，複製新專案的 URL。URL 包含專案 ID。
3. 複製並保留此專案 ID。您會將其用於在 CodePipeline 中建立管道。

   這裡有專案的 URL 範例。若要擷取專案 ID，請複製 `projects/` 後面的值。在此範例中，專案 ID 為 `eec4905f-98f8-40aa-9afc-4c1cfexample`。

   ```
   https://<region-URL>/devicefarm/home?region=us-west-2#/projects/
   eec4905f-98f8-40aa-9afc-4c1cfexample/runs
   ```

設定 CodePipeline 以使用您的 Device Farm 測試 (Amazon S3 範例)

1. 建立或使用已啟用版本控制的 S3 儲存貯體。遵循步驟 1：為您的應用程式建立 S3 儲存貯體中的說明，建立 S3 儲存貯體。
2. 在儲存貯體的 Amazon S3 主控台中，選擇「上傳」，然後按照指示上傳 .zip 檔案。

   您的範例應用程式必須封裝在 .zip 檔案中。
3. 若要建立您的管道及新增來源階段，請執行下列作業：

   a. 請登入AWS Management Console並開啟 CodePipeline 主控台，網址為 https://console.aws.amazon.com/codepipeline/。

   b. 選擇 Create pipeline (建立管道)。在 Step 1: Choose pipeline settings (步驟 1：選擇管道設定) 頁面的 Pipeline name (管道名稱) 中輸入管道的名稱。

   c. 在管線類型中，針對本教學課程的目的選擇 V1。您也可以選擇 V2；但是請注意，管道類型在特性和價格上有所不同。如需詳細資訊，請參閱管線類型。

   d. 在 Service role (服務角色) 中，讓 New service role (新服務角色) 維持在選取狀態，然後讓 Role name (角色名稱) 維持不變。若您已擁有現有服務角色，您也可以選擇使用它。

> **Note**
>
> 如果您使用在 2018 年 7 月之前建立的 CodePipeline 服務角色，則必須新增 Device Farm 的權限。若要這麼做，請開啟 IAM 主控台、尋找角色，然後將下列權限新增至角色的政策。如需詳細資訊，請參閱將許可新增至 CodePipeline 服務角色。
>
> ```
> {
>      "Effect": "Allow",
>      "Action": [
>         "devicefarm:ListProjects",
>         "devicefarm:ListDevicePools",
>         "devicefarm:GetRun",
>         "devicefarm:GetUpload",
>         "devicefarm:CreateUpload",
>         "devicefarm:ScheduleRun"
>      ],
>      "Resource": "*"
> }
> ```

e. 將 Advanced settings (進階設定) 下的設定保留為預設值，然後選擇 Next (下一步)。

f. 在 Step 2: Add source stage (步驟 2：新增來源階段) 頁面的 Source provider (來源提供者) 中，選擇 Amazon S3 (Amazon S3)。

g. 在 Amazon S3 位置中，輸入儲存貯體，例如`my-storage-bucket`，和物件金鑰，`s3-ios-test-1.zip`例如 .zip 檔案。

h. 選擇下一步。

4. 在 Build (建置) 中，建立您管道建置階段的預留位置。這可讓您在精靈中建立管道。在您使用精靈建立您的二階段管道之後，您便不再需要此預留位置建置階段。在完成管道後，便會刪除此第二階段，並會在步驟 5 中建立新的測試階段。

   a. 在 Build provider (建置提供者) 中，選擇 Add Jenkins (新增 Jenkins)。此建置選取為預留位置。不會使用。

   b. 在 Provider name (提供者名稱) 中，輸入名稱。該名稱為預留位置。不會使用。

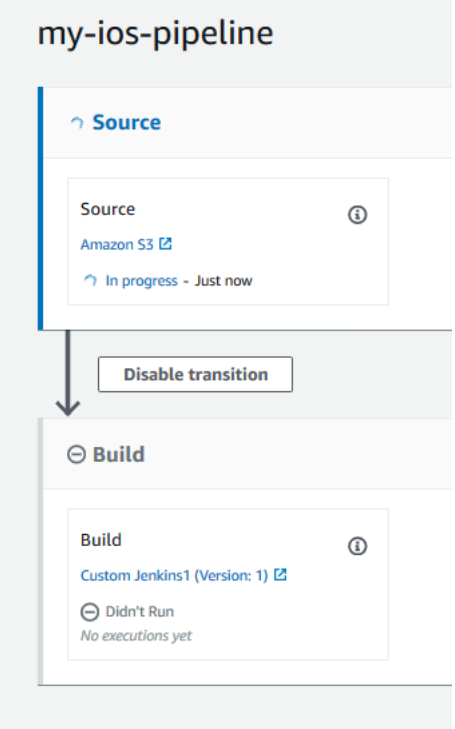
   c. 在 Server URL (伺服器 URL) 中，輸入文字。該文字為預留位置。不會使用。

   d. 在 Project name (專案名稱) 中，輸入名稱。該名稱為預留位置。不會使用。

   e. 選擇下一步。

f. 在 Step 4: Add deploy stage (步驟 4：新增部署階段) 頁面上，選擇 Skip deploy stage (跳過部署階段)，然後再次選擇 Skip (跳過) 來接受警告訊息。

g. 在 Step 5: Review (步驟 5：檢閱) 上，選擇 Create pipeline (建立管道)。您應該會看到圖表，顯示該來源及建置階段。



5. 將 Device Farm 測試動作新增至管線，如下所示：

a. 在右上角，選擇 Edit (編輯)。

b. 選擇 Edit stage (編輯階段)。選擇刪除。這會刪除預留位置階段，因為針對建立管道，您已不再需要它。

c. 在圖表的底部，選擇 + Add stage (+ 新增階段)。

d. 在階段名稱中，輸入階段的名稱，例如測試，然後選擇 Add stage (新增階段)。

e. 選擇 + Add action group (+ 新增動作群組)。

f. 在動作名稱中，輸入名稱，例如 DeviceFarmTest。

g. 在動作提供者中，選擇 AWSDevice Farm 允許 Region (區域) 預設為管道區域。

h. 在 Input artifacts (輸入成品) 中，選擇與測試階段之前的階段輸出成品相符的輸入成品，例如 `SourceArtifact`。

在 AWS CodePipeline 主控台中，您可以透過在管道圖表中資訊圖示的上方暫留，找到每個階段的輸出成品名稱。如果您的管道直接從「來源」階段測試您的應用程式，請選擇SourceArtifact。如果管線包含「建置」階段，請選擇BuildArtifact。

i. 在中 ProjectId，選擇您的 Device Farm 專案 ID。使用本教學課程開頭的步驟，擷取您的專案 ID。

j. 在中 DevicePoolArn，輸入裝置集區的 ARN。若要取得專案的可用裝置集區 ARN (包括常用裝置的 ARN)，請使用 AWS CLI 輸入下列命令：

```
aws devicefarm list-device-pools --arn arn:aws:devicefarm:us-
west-2:account_ID:project:project_ID
```

k. 在中 AppType，輸入 iOS。

以下是有效值的清單 AppType：

- iOS
- Android
- Web

l. 在 App (應用程式) 中，輸入已編譯的應用程式套件路徑。路徑為相對於測試階段輸入成品根的相對路徑。通常，此路徑與 `ios-test.ipa` 相似。

m. 在中TestType輸入測試類型，然後在「測試」中輸入測試定義檔案的路徑。路徑為相對於您測試輸入成品根的相對路徑。

如果您使用其中一個內建的 Device Farm 測試，請輸入 Device Farm 專案中設定的測試類型，例如 BUILTIN_FUZZ。在中 FuzzEventCount，輸入以毫秒為單位的時間，例如 6000。在中 FuzzEventThrottle，輸入以毫秒為單位的時間，例如 50。

如果您未使用其中一個內建的 [Device Farm] 測試，請輸入測試類型，然後在 [測試] 中輸入測試定義檔案的路徑。路徑為相對於您測試輸入成品根的相對路徑。

以下是有效值的清單 TestType：

- 爪哇胡尼特
- Appium_ 爪哇測試
- APPIUM 節點
- 蘋果紅寶石

- 蟒蛇
- 阿皮亚姆网站 JA_JUnit
- 蘋果網站測試
- 应用网节点
- 蘋果紅寶石
- 阿皮亞網蟒
- 內建模糊
- INSTRUMENTATION
- XCTEST
- XCTEST_UI

> **Note**
>
> 不支援自訂環境節點。

n. 在剩餘欄位中，提供適用於您測試及應用程式類型的組態。

o. (選擇性) 在 Advanced (進階) 中，提供您測試執行的組態資訊。

p. 選擇儲存。

q. 在您編輯的階段上，選擇 Done (完成)。在 AWS CodePipeline 窗格中，選擇 Save (儲存)，然後在警告訊息中選擇 Save (儲存)。

r. 若要提交您的變更並啟動管道執行，請選擇 Release change (發行變更)，然後選擇 Release (發行)。

教學課程：建立部署至 Service Catalog 的管線

Service Catalog 可讓您根據AWS CloudFormation範本建立及佈建產品。本教學說明如何建立和設定管道，將產品範本部署到 Service Catalog，並傳遞您在來源儲存庫 (已在 GitHub CodeCommit、或 Amazon S3 中建立) 中所做的變更。

> **Note**
>
> 當 Amazon S3 是管道的來源供應商時，您必須將所有封裝為單一 .zip 檔案的來源檔案上傳到儲存貯體。否則，來源動作會失敗。

首先，您在 Service Catalog 中建立產品，然後在中建立管道AWS CodePipeline。本教學課程提供兩種設定部署組態的選項：

- 在 Service Catalog 中建立產品，並將範本檔案上傳至來源儲存庫。在 CodePipeline 主控台中提供產品版本和部署規劃 (不需要個別的規劃檔)。請參閱 選項 1：不使用組態檔案部署至 Service Catalog。

  > **Note**
  >
  > 範本檔案可以 YAML 或 JSON 格式建立。

- 在 Service Catalog 中建立產品，並將範本檔案上傳至來源儲存庫。以單獨組態檔提供產品版本和部署組態。請參閱 選項 2：使用組態檔部署至 Service Catalog。

選項 1：不使用組態檔案部署至 Service Catalog

在此範例中，您上傳 S3 儲存貯體的AWS CloudFormation範例範本檔案，然後在 Service Catalog 中建立產品。接下來，您可以建立管道並在 CodePipeline 主控台中指定部署組態。

步驟 1：上傳範例範本檔案到來源儲存庫

1. 開啟文字編輯器。將以下項目貼到檔案以建立範例範本。儲存檔案為 `S3_template.json`。

```
{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Description": "CloudFormation Sample Template S3_Bucket: Sample template showing
 how to create a privately accessible S3 bucket. **WARNING** This template creates
 an S3 bucket. You will be billed for the resources used if you create a stack from
 this template.",
  "Resources": {
    "S3Bucket": {
      "Type": "AWS::S3::Bucket",
      "Properties": {}
    }
  },
  "Outputs": {
    "BucketName": {
      "Value": {
        "Ref": "S3Bucket"
      },
      "Description": "Name of Amazon S3 bucket to hold website content"
```

```
    }
  }
}
```

此範本可AWS CloudFormation讓您建立可供 Service Catalog 使用的 S3 儲存貯體。

2. 將 `S3_template.json` 檔案上傳至 AWS CodeCommit 儲存庫。

步驟 2：在 Service Catalog 中建立產品

1. 身為 IT 管理員，請登入 Service Catalog 主控台，移至 [產品] 頁面，然後選擇 [上傳新產品]。
2. 在 Upload new product (上傳新產品) 頁面上，完成下列動作：
   a. 在 Product name (產品名稱) 中，輸入您想要使用的新產品名稱。
   b. 在 Description (敘述) 中輸入產品型錄描述。此處的描述會顯示在產品列表中，以協助使用者選擇正確的產品。
   c. 在 Provided by (提供者) 中輸入 IT 部門或管理員的名稱。
   d. 選擇下一步。
3. (選擇性) 在 Enter support details (輸入支援詳細資訊) 中，請輸入產品支援聯絡資訊，然後選擇 Next (下一步)。
4. 於 Version details (版本詳細資訊) 中，完成以下項目：
   a. 選擇 Upload a template file (上傳範本檔案)。瀏覽您的 `S3_template.json` 檔案並上傳。
   b. 在 Version title (版本標題) 中，輸入產品版本名稱 (例如，**`devops S3 v2`**)。
   c. 在 Description (敘述) 中，輸入區分此版本與其他版本的詳細資訊。
   d. 選擇下一步。
5. 在 Review (檢閱) 頁面上，確認資訊正確，然後選擇 Create (建立)。
6. 在瀏覽器中複製 Products (產品) 頁面上的 URL。這會包含產品 ID。複製並保留此產品 ID。您會將其用於在 CodePipeline 中建立管道。

   以下是名為 `my-product` 的產品 URL。若要擷取產品 ID，請複製等號 (=) 和 & (&) 之間的值。在此範例中，產品 ID 為 `prod-example123456`。

```
https://<region-URL>/servicecatalog/home?region=<region>#/admin-products?
productCreated=prod-example123456&createdProductTitle=my-product
```

> **Note**
>
> 在您離開頁面之前複製您的產品 URL。您離開此頁面後，您必須使用 CLI 取得您的產品 ID。

幾秒鐘後，您的產品即會出現在 Products (產品) 頁面上。可能需要重新整理瀏覽器才能在清單中看到產品。

步驟 3：建立管道

1. 若要命名管道和選擇管道參數，請執行下列動作：
   a. 請登入AWS Management Console並開啟 CodePipeline 主控台，網址為 https://console.aws.amazon.com/codepipeline/。
   b. 選擇 Getting started (入門)。選擇 Create pipeline (建立管道) 然後輸入管道名稱。
   c. 在管線類型中，針對本教學課程的目的選擇 V1。您也可以選擇 V2；但是請注意，管道類型在特性和價格上有所不同。如需詳細資訊，請參閱管線類型。
   d. 在 [服務角色] 中，選擇 [新增服務角色] CodePipeline 以允許在 IAM 中建立服務角色。
   e. 將 Advanced settings (進階設定) 下的設定保留為預設值，然後選擇 Next (下一步)。
2. 若要新增來源階段，請執行以下操作：
   a. 在 Source provider (來源提供者) 中選擇 AWS CodeCommit。
   b. 在 Repository name (儲存庫名稱) 和 Branch name (分支名稱) 中，輸入您想為來源動作使用的儲存庫和分支。
   c. 選擇下一步。
3. 在 Add build stage (新增建置階段) 中，選擇 Skip build stage (跳過建置階段)，然後再次選擇 Skip (跳過) 來接受警告訊息。
4. 在 Add deploy stage (新增部署階段) 中，完成以下項目：
   a. 在 [部署提供者] 中，選擇AWS Service Catalog。
   b. 對於部署設定，選擇 Enter deployment configuration (輸入部署設定)。
   c. 在產品 ID 中，貼上您從 Service Catalog 主控台複製的產品 ID。
   d. 在 Template file path (範本檔案路徑) 中，輸入範本檔案存放的相對路徑。

e. 在 [產品類型] 中選擇 [AWS CloudFormation範本]。

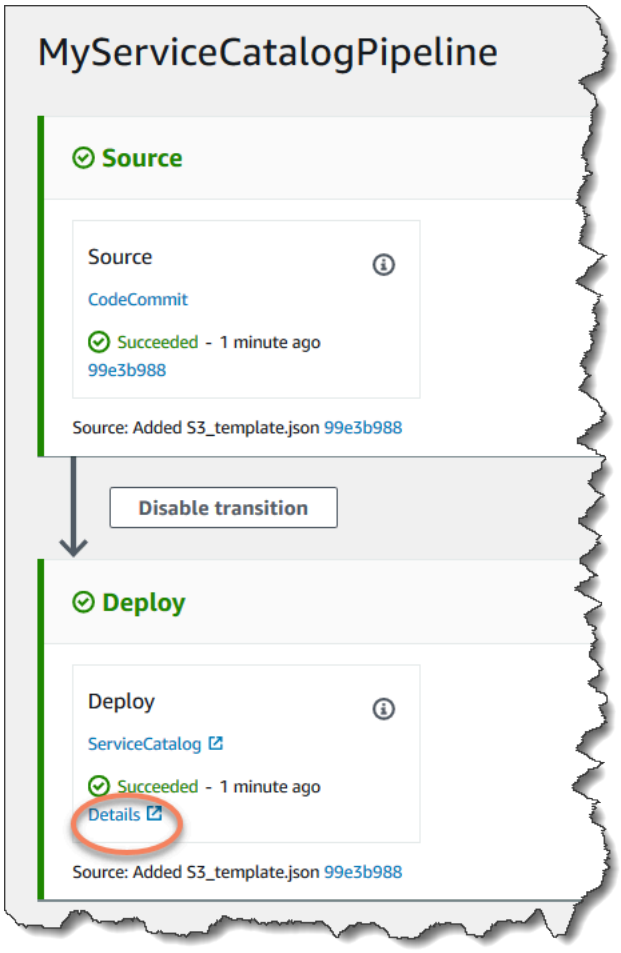
f. 在產品版本名稱中，輸入您在 Service Catalog 中指定的產品版本名稱。如果要將範本變更部署到新產品版本，請輸入同一產品之前版本中未曾使用過的產品版本名稱。

g. 對於 Input artifact (輸入成品)，請選擇來源輸入成品。

h. 選擇下一步。

5. 在 Review (檢閱) 中檢閱您的管道設定，然後選擇 Create (建立)。

6. 管道成功執行之後，在部署階段選擇 Details (詳細資訊)。這會在「Service Catalog」中開啟您的產品。



7. 在您的產品資訊下，選擇您的版本名稱以開啟產品範本。檢視範本部署。

步驟 4：在 Service Catalog 中推送變更並驗證您的產品

1. 在 CodePipeline 主控台中檢視您的管道，然後在來源階段選擇 [詳細資訊]。您的來源 AWS CodeCommit 儲存庫會在主控台中開啟。選擇 Edit (編輯)，並在檔案中進行變更 (例如，對描述進行變更)。

   ```
   "Description": "Name of Amazon S3 bucket to hold and version website content"
   ```

2. 遞交並推送您的變更。您的管道將在您推送變更之後啟動。管線執行完成時，請在部署階段選擇 [詳細資料]，在 Service Catalog 中開啟您的產品。
3. 在您的產品資訊下，選擇新版本名稱以開啟產品範本。查看部署的範本變更。

選項 2：使用組態檔部署至 Service Catalog

在此範例中，您上傳 S3 儲存貯體的AWS CloudFormation範例範本檔案，然後在 Service Catalog 中建立產品。您也可以上傳指定您部署組態的單獨組態檔案。接著，您將建立管道並指定組態檔案的位置。

步驟 1：上傳範例範本檔案到來源儲存庫

1. 開啟文字編輯器。將以下項目貼到檔案以建立範例範本。儲存檔案為 `S3_template.json`。

   ```
   {
     "AWSTemplateFormatVersion": "2010-09-09",
     "Description": "CloudFormation Sample Template S3_Bucket: Sample template showing
    how to create a privately accessible S3 bucket. **WARNING** This template creates
    an S3 bucket. You will be billed for the resources used if you create a stack from
    this template.",
     "Resources": {
       "S3Bucket": {
         "Type": "AWS::S3::Bucket",
         "Properties": {}
       }
     },
     "Outputs": {
       "BucketName": {
         "Value": {
           "Ref": "S3Bucket"
         },
         "Description": "Name of Amazon S3 bucket to hold website content"
       }
   ```

```
  }
}
```

此範本可AWS CloudFormation讓您建立可供 Service Catalog 使用的 S3 儲存貯體。

2. 將 S3_template.json 檔案上傳至 AWS CodeCommit 儲存庫。

步驟 2：建立您的產品部署組態檔案

1. 開啟文字編輯器。為您的產品建立組態檔案。組態檔是用來定義您的 Service Catalog 部署參數/偏好設定。您將在建立管道時使用此檔案。

   此範例提供一個「devops S3 v2」的 ProductVersionName 和 MyProductVersionDescription 的 ProductVersionDescription。如果您要將範本變更部署到新產品版本，只要輸入同一產品之前版本中未曾使用過的產品版本名稱即可。

   儲存檔案為 sample_config.json。

```
{
    "SchemaVersion": "1.0",
    "ProductVersionName": "devops S3 v2",
    "ProductVersionDescription": "MyProductVersionDescription",
    "ProductType": "CLOUD_FORMATION_TEMPLATE",
    "Properties": {
        "TemplateFilePath": "/S3_template.json"
    }
}
```

   此檔案會在您每次執行管道時建立產品版本資訊。

2. 將 sample_config.json 檔案上傳至 AWS CodeCommit 儲存庫。請確定您上傳此檔案到您的來源儲存庫。

步驟 3：在 Service Catalog 中建立產品

1. 身為 IT 管理員，請登入 Service Catalog 主控台，移至 [產品] 頁面，然後選擇 [上傳新產品]。
2. 在 Upload new product (上傳新產品) 頁面上，完成下列動作：

   a. 在 Product name (產品名稱) 中，輸入您想要使用的新產品名稱。

b. 在 Description (敘述) 中輸入產品型錄描述。此處的描述顯示於產品列表中，以協助使用者選擇正確的產品。

c. 在 Provided by (提供者) 中輸入 IT 部門或管理員的名稱。

d. 選擇下一步。

3. (選擇性) 在 Enter support details (輸入支援詳細資訊) 中，請輸入產品支援聯絡資訊，然後選擇 Next (下一步)。

4. 於 Version details (版本詳細資訊) 中，完成以下項目：

a. 選擇 Upload a template file (上傳範本檔案)。瀏覽您的 `S3_template.json` 檔案並上傳。

b. 在 Version title (版本標題) 中，輸入產品版本名稱 (例如「devops S3 v2」)。

c. 在 Description (敘述) 中，輸入區分此版本與其他版本的詳細資訊。

d. 選擇下一步。

5. 在 Review (檢閱) 頁面上，確認資訊正確，然後選擇確認並上傳。

6. 在瀏覽器中複製 Products (產品) 頁面上的 URL。這會包含產品 ID。複製並保留此產品 ID。您會用於在 CodePipeline 中建立管道。

   以下是名為 `my-product` 的產品 URL。若要擷取產品 ID，請複製等號 (=) 和 & (&) 之間的值。在此範例中，產品 ID 為 `prod-example123456`。

   ```
   https://<region-URL>/servicecatalog/home?region=<region>#/admin-products?
   productCreated=prod-example123456&createdProductTitle=my-product
   ```

   > **Note**
   >
   > 在您離開頁面之前複製您的產品 URL。您離開此頁面後，您必須使用 CLI 取得您的產品 ID。

   幾秒鐘後，您的產品即會出現在 Products (產品) 頁面上。可能需要重新整理瀏覽器才能在清單中看到產品。

步驟 4：建立管道

1. 若要命名管道和選擇管道參數，請執行下列動作：

a. 請登入AWS Management Console並開啟 CodePipeline 主控台，網址為 https://console.aws.amazon.com/codepipeline/。

b. 選擇 Getting started (入門)。選擇 Create pipeline (建立管道) 然後輸入管道名稱。

c. 在 [服務角色] 中，選擇 [新增服務角色] CodePipeline 以允許在 IAM 中建立服務角色。

d. 將 Advanced settings (進階設定) 下的設定保留為預設值，然後選擇 Next (下一步)。

2. 若要新增來源階段，請執行以下操作：

a. 在 Source provider (來源提供者) 中選擇 AWS CodeCommit。

b. 在 Repository name (儲存庫名稱) 和 Branch name (分支名稱) 中，輸入您想為來源動作使用的儲存庫和分支。

c. 選擇下一步。

3. 在 Add build stage (新增建置階段) 中，選擇 Skip build stage (跳過建置階段)，然後再次選擇 Skip (跳過) 來接受警告訊息。

4. 在 Add deploy stage (新增部署階段) 中，完成以下項目：

a. 在 [部署提供者] 中，選擇AWS Service Catalog。

b. 選擇 Use configuration file (使用組態檔案)。

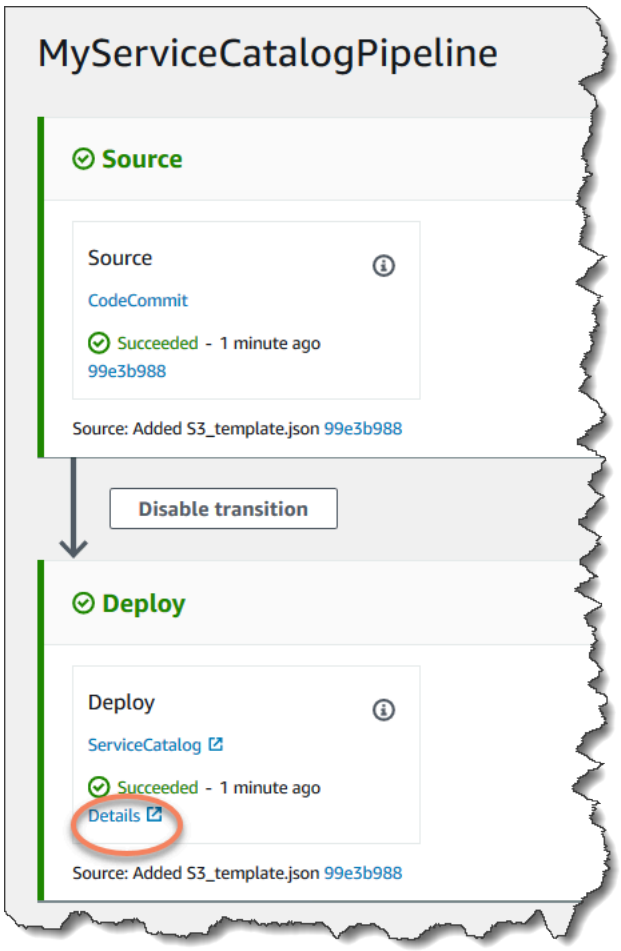
c. 在產品 ID 中，貼上您從 Service Catalog 主控台複製的產品 ID。

d. 在 Configuration file path (組態檔案路徑中)，輸入您儲存庫中組態檔案的檔案路徑。

e. 選擇下一步。

5. 在 Review (檢閱) 中檢閱您的管道設定，然後選擇 Create (建立)。

6. 管道成功執行之後，請在部署階段選擇 [詳細資料]，在 Service Catalog 中開啟您的產品。



7. 在您的產品資訊下，選擇您的版本名稱以開啟產品範本。檢視範本部署。

步驟 5：在 Service Catalog 中推送變更並驗證您的產品

1. 在 CodePipeline 主控台中檢視您的管道，然後在來源階段選擇「詳細資訊」。您的來源 AWS CodeCommit 儲存庫會在主控台中開啟。選擇 Edit (編輯)，然後在檔案中進行變更 (例如，對描述進行變更)。

```
"Description": "Name of Amazon S3 bucket to hold and version website content"
```

2. 遞交並推送您的變更。您的管道將在您推送變更之後啟動。管線執行完成時，請在部署階段選擇 [詳細資料]，在 Service Catalog 中開啟您的產品。
3. 在您的產品資訊下，選擇新版本名稱以開啟產品範本。查看部署的範本變更。

教學：使用 AWS CloudFormation 建立管道

這些範例提供範例範本，可讓您在每次來源碼變更時，使用 AWS CloudFormation 建立一個管道，將您的應用程式部署至您的執行個體。範例範本會建立您可以在 AWS CodePipeline 中檢視的管道。管道會偵測透過 Amazon CloudWatch 活動儲存的變更到達。

主題

範例 1：使用 AWS CloudFormation 建立 AWS CodeCommit 管道

本演練說明如何使用AWS CloudFormation主控台來建立包含連接至 CodeCommit 來源儲存放庫的管道的基礎結構。在本教學中，您將使用提供的範例範本檔案建立資源堆疊，其中包括成品存放區、管道和變更偵測資源，例如 Amazon E CloudWatch vents 規則。在 AWS CloudFormation 中建立您的資源堆疊之後，您可以在 AWS CodePipeline 主控台中檢視您的管道。管線是具有 CodeCommit 來源階段和 CodeDeploy 部署階段的兩階段管線。

先決條件：

您必須先建立下列資源，以搭配 AWS CloudFormation 範例範本使用：

- 您必須先建立來源儲存庫。您可以使用在 教學：建立範本管道 (CodeCommit 儲存庫) 中建立的 AWS CodeCommit 儲存庫。
- 您必須已建立 CodeDeploy 應用程式和部署群組。您可以使用在 教學：建立範本管道 (CodeCommit 儲存庫) 中建立的 CodeDeploy 資源。
- 選擇下列其中一個連結，下載 AWS CloudFormation 範本檔案範例以建立管道：YAML | JSON

  解壓縮檔案並將它放在您的本機電腦。
- 下載 SampleApp_Linux.zip 範例應用程式檔案。

在 AWS CloudFormation 中建立您的管道

1. 從 SampleApp_Linux.zip 解壓縮文件並將文件上傳到您的AWS CodeCommit存儲庫。您必須將解壓縮的檔案上傳到您儲存庫的根目錄。您可以依照 步驟 2：將示例代碼添加到存 CodeCommit 儲庫 中的指示，將檔案推送到您的儲存庫。

2. 開啟 AWS CloudFormation 主控台，然後選擇 Create Stack (建立堆疊)。選擇 With new resources (standard) (使用新資源 (標準))。
3. 在「指定範本」下，選擇「上傳範本」。選取 [選擇檔案]，然後從本機電腦選擇範本檔案。選擇下一步。
4. 在 Stack name (堆疊名稱) 中，輸入管道的名稱。即會顯示範例範本指定的參數。輸入下列參數：
   a. 在中 ApplicationName，輸入 CodeDeploy應用程式的名稱。
   b. 在中 BetaFleet，輸入 CodeDeploy 部署群組的名稱。
   c. 在中 BranchName，輸入您要使用的儲存庫分支。
   d. 在中 RepositoryName，輸入 CodeCommit 來源儲存庫的名稱。
5. 選擇下一步。接受以下頁面上的預設值，然後選擇 Next (下一步)。
6. 在 [功能] 中，選取 [我確認AWS CloudFormation可能會建立 IAM 資源]，然後選擇 [建立堆疊]。
7. 在完成您的堆疊建立之後，請檢視事件清單以檢查是否有任何錯誤。

   疑難排解

   在 AWS CloudFormation 中建立管道的 IAM 使用者可能需要額外的許可，來為管道建立資源。政策中需要下列許可，才能AWS CloudFormation為 CodeCommit管道建立所需的 Amazon E CloudWatch vents 資源：

```
{
     "Effect": "Allow",
     "Action": [
        "events:PutRule",
        "events:PutEvents",
        "events:PutTargets",
        "events:DeleteRule",
        "events:RemoveTargets",
        "events:DescribeRule"
     ],
     "Resource": "resource_ARN"
}
```

8. 請登入AWS Management Console並開啟 CodePipeline 主控台，網址為 https://console.aws.amazon.com/codepipeline/。

   在 Pipelines (管道) 下，選擇您的管道，然後選擇 View (檢視)。本圖顯示您的管道來源和部署階段。

> **Note**
>
> 若要檢視已建立的管線，請在中的堆疊的 [資源] 索引標籤下找到 [邏輯 ID] 欄AWS CloudFormation。請注意管線「實體 ID」欄中的名稱。在中 CodePipeline，您可以在建立堆疊的區域中檢視具有相同實體 ID (管道名稱) 的管道。

9. 在來源儲存庫中遞交並推送變更。您的變更偵測資源會套用該變更，然後您的管道便會啟動。

範例 2：使用以下的方式來建立 Amazon S3 管道AWS CloudFormation

本逐步解說說明如何使用AWS CloudFormation主控台建立基礎設施，其中包含連接至 Amazon S3 來源儲存貯體的管道。在本教學中，您將使用提供的範例範本檔案建立資源堆疊，其中包括來源儲存貯體、成品存放區、管道和變更偵測資源，例如 Amazon E CloudWatch vents 規則和 CloudTrail追蹤。在 AWS CloudFormation 中建立您的資源堆疊之後，您可以在 AWS CodePipeline 主控台中檢視您的管道。管道是具有 Amazon S3 來源階段和 CodeDeploy部署階段的兩階段管道。

先決條件：

您必須具備下列資源，才能搭配 AWS CloudFormation 範例範本使用：

- 您必須已建立 Amazon EC2 執行個體，並在執行個體上安裝 CodeDeploy 代理程式。您必須已建立 CodeDeploy 應用程式和部署群組。使用您在其中建立的 Amazon EC2 和 CodeDeploy 資源教學：建立範本管道 (CodeCommit 儲存庫)。
- 選擇下列連結以下載範例AWS CloudFormation範本檔案，以使用 Amazon S3 來源建立管道：
  - 下載管道的範例範本：YAML | JSON
  - 下載 CloudTrail 值區和追蹤記錄的範例範本：YAML | JSON
  - 解壓縮檔案並將它們放在您的本機電腦。
- 請從 SampleApp_Linux.zip 下載範例應用程式。

  將 .zip 檔案儲存至本機電腦。在建立堆疊之後，您會上傳 .zip 檔案。

在 AWS CloudFormation 中建立您的管道

1. 開啟 AWS CloudFormation 主控台，然後選擇 Create Stack (建立堆疊)。選擇 With new resources (standard) (使用新資源 (標準))。

2. 在 [選擇範本] 中，選擇 [上傳範本]。選取 [選擇檔案]，然後從本機電腦選擇範本檔案。選擇下一步。
3. 在 Stack name (堆疊名稱) 中，輸入管道的名稱。即會顯示範例範本指定的參數。輸入下列參數：
   a. 在中 ApplicationName，輸入 CodeDeploy 應用程式的名稱。您可以取代 `DemoApplication` 預設名稱。
   b. 在中 BetaFleet，輸入 CodeDeploy 部署群組的名稱。您可以取代 `DemoFleet` 預設名稱。
   c. 在中 SourceObjectKey，輸入`SampleApp_Linux.zip`。您可以在範本建立儲存貯體和管道後，將此檔案上傳到您的儲存貯體。
4. 選擇下一步。接受以下頁面上的預設值，然後選擇 Next (下一步)。
5. 在 [功能] 中，選取 [我確認AWS CloudFormation可能會建立 IAM 資源]，然後選擇 [建立堆疊]。
6. 在完成您的堆疊建立之後，請檢視事件清單以檢查是否有任何錯誤。

   疑難排解

   在中建立管道的 IAM 使用者AWS CloudFormation可能需要其他許可，才能為管道建立資源。政策中需要下列許可，才能AWS CloudFormation為 Amazon S3 管道建立所需的 Amazon CloudWatch 活動資源：

```
{
     "Effect": "Allow",
     "Action": [
        "events:PutRule",
        "events:PutEvents",
        "events:PutTargets",
        "events:DeleteRule",
        "events:RemoveTargets",
        "events:DescribeRule"
     ],
     "Resource": "resource_ARN"
}
```

7. 在 AWS CloudFormation 中，請在您堆疊的 Resources (資源) 標籤內檢視您為堆疊建立的資源。

> **Note**
>
> 若要檢視已建立的管線，請在中的堆疊的 [資源] 索引標籤下找到 [邏輯 ID] 欄AWS CloudFormation。請注意管線「實體 ID」欄中的名稱。在中 CodePipeline，您可以在建立堆疊的區域中檢視具有相同實體 ID (管道名稱) 的管道。

選擇名稱中包含 `sourcebucket` 標籤的 S3 儲存貯體，例如 `s3-cfn-codepipeline-sourcebucket-y04EXAMPLE.`。請不要選擇管道成品儲存貯體。

來源儲存貯體是空白的，因為 AWS CloudFormation 才剛建立資源。開啟 Amazon S3 主控台並找到儲存貯存貯存貯存`sourcebucket`貯存貯 選擇 Upload (上傳) 並遵循說明，來上傳您的 `SampleApp_Linux.zip` .zip 檔案。

> **Note**
>
> 當 Amazon S3 是管道的來源供應商時，您必須將所有封裝為單一 .zip 檔案的來源檔案上傳到儲存貯體。否則，來源動作會失敗。

8. 請登入AWS Management Console並開啟 CodePipeline 主控台，網址為 https://console.aws.amazon.com/codepipeline/。

   在 Pipelines (管道) 下，選擇您的管道，然後選擇 View (檢視)。本圖顯示您的管道來源和部署階段。

9. 完成下列程序中的步驟來建立 AWS CloudTrail 資源。

在 AWS CloudFormation 中建立您的 AWS CloudTrail 資源

1. 開啟 AWS CloudFormation 主控台，然後選擇 Create Stack (建立堆疊)。
2. 在 Choose a template (選擇範本) 中，請選擇 Upload a template to Amazon S3 (上傳範本至 Amazon S3)。選擇 Browse (瀏覽)，然後從您的本機電腦選擇 AWS CloudTrail 資源的範本檔案。選擇下一步。
3. 在 Stack name (堆疊名稱) 中，輸入資源堆疊的名稱。即會顯示範例範本指定的參數。輸入下列參數：

   - 在中 SourceObjectKey，接受範例應用程式 zip 檔案的預設值。

4. 選擇下一步。接受以下頁面上的預設值，然後選擇 Next (下一步)。
5. 在 [功能] 中，選取 [我確認AWS CloudFormation可能會建立 IAM 資源]，然後選擇 [建立]。
6. 在完成您的堆疊建立之後，請檢視事件清單以檢查是否有任何錯誤。

   政策中需要下列許可AWS CloudFormation，才能為 Amazon S3 管道建立所需的 CloudTrail 資源：

   ```
   {
        "Effect": "Allow",
        "Action": [
           "cloudtrail:CreateTrail",
           "cloudtrail:DeleteTrail",
           "cloudtrail:StartLogging",
           "cloudtrail:StopLogging",
           "cloudtrail:PutEventSelectors"
        ],
        "Resource": "resource_ARN"
   }
   ```

7. 請登入AWS Management Console並開啟 CodePipeline 主控台，網址為 https://console.aws.amazon.com/codepipeline/。

   在 Pipelines (管道) 下，選擇您的管道，然後選擇 View (檢視)。本圖顯示您的管道來源和部署階段。
8. 在來源儲存貯體中遞交並推送變更。您的變更偵測資源會套用該變更，然後您的管道便會啟動。

教學：建立透過 AWS CloudFormation 部署動作使用變數的管道

在本教學中，您使用 AWS CodePipeline 主控台，建立具有部署動作的管道。管道執行時，此範本會建立堆疊，並建立 `outputs` 檔案。由堆疊範本產生的輸出是由 CodePipeline 中 AWS CloudFormation 動作產生的變數。

在從範本建立堆疊的動作中，您可以指定變數命名空間。然後，該 `outputs` 檔案產生的變數可以被後續動作使用。在此範例中，您會根據 AWS CloudFormation 動作所產生的 `StackName` 變數來建立變更集。手動核准之後，您執行變更集，然後建立根據 `StackName` 變數刪除堆疊的刪除堆疊動作。

主題

- 先決條件：建立AWS CloudFormation服務角色和 CodeCommit 存放庫
- 步驟 1：下載、編輯及上傳範例 AWS CloudFormation 範本

- 步驟 2：建立管道
- 步驟 3：新增 AWS CloudFormation 部署動作以建立變更集
- 步驟 4：新增手動核准動作
- 步驟 5：新增 CloudFormation部署動作以執行變更集
- 步驟 6：新增 CloudFormation部署動作以刪除堆疊

先決條件：建立AWS CloudFormation服務角色和 CodeCommit 存放庫

您必須已擁有下列各項目：

- 儲 CodeCommit 存庫。您可以使用在 教學：建立範本管道 (CodeCommit 儲存庫) 中建立的 AWS CodeCommit 儲存庫。
- 此範例會從範本建立 Amazon DocumentDB 堆疊。您必須使用 AWS Identity and Access Management (IAM) 建立具有下列 Amazon DocumentDB 許可的AWS CloudFormation服務角色。

```
"rds:DescribeDBClusters",
"rds:CreateDBCluster",
"rds:DeleteDBCluster",
"rds:CreateDBInstance"
```

步驟 1：下載、編輯及上傳範例 AWS CloudFormation 範本

下載範例AWS CloudFormation範本檔案並將其上傳至您的 CodeCommit 存放庫。

1. 導覽至您區域的範例範本頁面。例如，us-west-2 的頁面位於 https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/sample-templates-services-us-west-2.html。
   在 Amazon DocumentDB 下，下載 Amazon DocumentDB 集群的模板。檔案名稱為 `documentdb_full_stack.yaml`。
2. 將 `documentdb_full_stack.yaml` 檔案解壓縮，然後在文字編輯器中開啟檔案。進行下列變更：

   a. 在此範例中，將下列 `Purpose:` 參數新增至範本中的 `Parameters` 區段。

```
Purpose:
  Type: String
  Default: testing
  AllowedValues:
```

```
      - testing
      - production
    Description: The purpose of this instance.
```

b. 在此範例中，將下列 StackName 輸出新增至範本中的 Outputs: 區段。

```
  StackName:
    Value: !Ref AWS::StackName
```

3. 將範本檔案上傳至 AWS CodeCommit 儲存庫。您必須將解壓縮和編輯過的範本檔案上傳至儲存庫的根目錄。

   若要使用 CodeCommit 主控台上傳檔案：

   a. 開啟主 CodeCommit 控台，然後從「儲存庫」清單中選擇您的存放庫。
   b. 選擇 Add file (新增檔案)，然後選擇 Upload file (上傳檔案)。
   c. 選取 Choose file (選擇檔案)，然後瀏覽您的檔案。輸入您的使用者名稱和電子郵件地址來確定變更。選擇 Commit changes (遞交變更)。

   您的檔案在儲存庫的根層級看起來應該像這樣：

```
documentdb_full_stack.yaml
```

步驟 2：建立管道

在本節中，您可以採取下列動作建立管道：

- 具有 CodeCommit 動作的來源階段，其中來源成品是您的範本檔案。
- 具有 AWS CloudFormation 部署動作的部署階段。

系統會為精靈所建立的來源和部署階段中每個動作指派變數命名空間，分別是 SourceVariables 和 DeployVariables。由於動作已指派命名空間，因此在此範例中設定的變數可供下游動作使用。如需詳細資訊，請參閱Variables。

使用精靈建立管道

1. 請登入AWS Management Console並開啟 CodePipeline 主控台，網址為 http://console.aws.amazon.com/codesuite/codepipeline/home。

2. 在 Welcome (歡迎) 頁面、Getting started (入門) 頁面、或者 Pipelines (管道) 頁面上，選擇 Create pipeline (建立管道)。
3. 在 Step 1: Choose pipeline settings (步驟 1：選擇管道設定) 的 Pipeline name (管道名稱) 中，輸入 **MyCFNDeployPipeline**。
4. 在管線類型中，針對本教學課程的目的選擇 V1。您也可以選擇 V2；但是請注意，管道類型在特性和價格上有所不同。如需詳細資訊，請參閱管線類型。
5. 在 Service role (服務角色) 中，執行下列其中一項作業：
   - 選擇 [新增服務角色] CodePipeline 以允許在 IAM 中建立服務角色。
   - 選擇 Existing service role (現有服務角色)。在 Role name (角色名稱) 中，從清單選擇您的服務角色。
6. 在 Artifact store (成品存放區) 中：
   a. 選擇預設位置以針對管道選取的區域中的管道使用預設成品存放區，例如指定為預設值的 Amazon S3 成品儲存貯體。
   b. 如果您已經在管道的相同區域中擁有成品存放區 (例如 Amazon S3 成品儲存貯體)，請選擇「自訂位置」。

> **Note**
>
> 這不是原始碼的來源儲存貯體。這是管道的成品存放區。每個管道都需要有個別成品存放區，例如 S3 儲存貯體。建立或編輯管道時，管線區域中必須有一個人工因素儲存貯體，而每個執行動作的「AWS區域」都必須有一個成品儲存貯體。
> 如需詳細資訊，請參閱 輸入和輸出成品 及 CodePipeline 配管結構參照。

選擇下一步。

7. 在 Step 2: Add source stage (步驟 2：新增來源階段)：
   a. 在 Source provider (來源提供者) 中選擇 AWS CodeCommit。
   b. 在存放庫名稱中，選擇您在其中建立的 CodeCommit 存放庫名稱步驟 1：建立 CodeCommit 儲存庫。
   c. 在 Branch name (分支名稱) 中，選擇包含最新程式碼更新的分支名稱。

選取儲存庫名稱和分支後，會顯示要為此管道建立的 Amazon E CloudWatch vents 規則。

選擇下一步。

8. 在 Step 3: Add build stage (步驟 3：新增建置階段) 中，選擇 Skip build stage (跳過建置階段)，然後再次選擇 Skip (跳過) 來接受警告訊息。

   選擇下一步。

9. 在 Step 4: Add deploy stage (步驟 4：新增部署階段) 中：

   a. 在 Action name (動作名稱) 中，選擇 Deploy (部署)。在部署提供者中，選擇CloudFormation。

   b. 在 Action mode (動作模式) 中，選擇 Create or update a stack (建立或更新堆疊)。

   c. 在 Stack name (堆疊名稱) 中，輸入堆疊的名稱。這是範本將建立的堆疊名稱。

   d. 在 Output file name (輸出檔案名稱) 中，輸入輸出檔案的名稱，例如 **`outputs`**。這是在建立堆疊後由此動作建立的檔案名稱。

   e. 展開 Advanced (進階)。在 Parameter overrides (參數覆寫) 下，輸入範本覆寫作為索引鍵/值組。例如，此範本需要下列覆寫。

      ```
      {
      "DBClusterName": "MyDBCluster",
      "DBInstanceName": "MyDBInstance",
      "MasterUser": "UserName",
      "MasterPassword": "Password",
      "DBInstanceClass": "db.r4.large",
      "Purpose": "testing"}
      ```

      如果您未輸入覆寫，範本會建立具有預設值的堆疊。

   f. 選擇下一步。

   g. 選擇 Create pipeline (建立管道)。允許您的管道執行。您的兩階段管道已完成，並準備好新增其他階段。

步驟 3：新增 AWS CloudFormation 部署動作以建立變更集

在您的管道中建立下一個動作，以便 AWS CloudFormation 在手動核准動作之前建立變更集。

1. 開啟主 CodePipeline 控台，網址為 https://console.aws.amazon.com/codepipeline/。

在 Pipelines (管道) 下，選擇您的管道，然後選擇 View (檢視)。本圖顯示您的管道來源和部署階段。

2. 選擇編輯管道，或繼續在 Edit (編輯) 模式中顯示管道。
3. 選擇此選項可編輯「部署」階段。
4. 新增部署動作，該動作將為在上一個動作中建立的堆疊建立變更集。您可以在階段中的現有動作之後加入此動作。

   a. 在 Action name (動作名稱) 中，輸入 Change_Set。在 [動作提供者] 中，選擇AWS CloudFormation。

   b. 在輸入人工因素中，選擇SourceArtifact。

   c. 在 Action mode (動作模式) 中，選擇 Create or replace a change set (建立或取代變更集)。

   d. 在 Stack name (堆疊名稱) 中，輸入變數語法，如下所示。這是堆疊名稱，變更集是為此堆疊而建立，其中預設命名空間 `DeployVariables` 已指派給該動作。

   ```
   #{DeployVariables.StackName}
   ```

   e. 在 Change set name (變更集名稱) 中，輸入變更集的名稱。

   ```
   my-changeset
   ```

   f. 在 Parameter Overrides (參數覆寫) 中，將 `Purpose` 參數從 `testing` 變更為 `production`。

   ```
   {
   "DBClusterName": "MyDBCluster",
   "DBInstanceName": "MyDBInstance",
   "MasterUser": "UserName",
   "MasterPassword": "Password",
   "DBInstanceClass": "db.r4.large",
   "Purpose": "production"}
   ```

   g. 選擇 Done (完成) 以儲存動作。

步驟 4：新增手動核准動作

在管道中建立手動核准動作。

1. 選擇編輯管道，或繼續在 Edit (編輯) 模式中顯示管道。
2. 選擇此選項可編輯「部署」階段。
3. 在建立變更集的部署動作之後新增手動核准動作。此動作可讓您在管道執行變更集之前，在 AWS CloudFormation 中驗證已建立的資源變更集。

步驟 5：新增 CloudFormation部署動作以執行變更集

在管道中建立下一個動作，以便 AWS CloudFormation 在手動核准動作之後執行變更集。

1. 開啟主 CodePipeline 控台，網址為 https://console.aws.amazon.com/codepipeline/。

   在 Pipelines (管道) 下，選擇您的管道，然後選擇 View (檢視)。本圖顯示您的管道來源和部署階段。
2. 選擇編輯管道，或繼續在 Edit (編輯) 模式中顯示管道。
3. 選擇此選項可編輯「部署」階段。
4. 新增部署動作，以執行上一個手動動作中核准的變更集：

   a. 在 Action name (動作名稱) 中，輸入 Execute_Change_Set。在 [動作提供者] 中，選擇AWS CloudFormation。

   b. 在輸入人工因素中，選擇SourceArtifact。

   c. 在 Action mode (動作模式) 中，選擇 Execute a change set (執行變更組合)。

   d. 在 Stack name (堆疊名稱) 中，輸入變數語法，如下所示。這是堆疊名稱，變更集是為此堆疊而建立。

   ```
   #{DeployVariables.StackName}
   ```

   e. 在 Change set name (變更集名稱) 中，輸入您在上一個動作中建立的變更集名稱。

   ```
   my-changeset
   ```

   f. 選擇 Done (完成) 以儲存動作。

   g. 繼續管道執行。

步驟 6：新增 CloudFormation部署動作以刪除堆疊

在管道中建立最終動作，以便 AWS CloudFormation 從輸出檔案中的變數取得堆疊名稱並刪除堆疊。

1. 開啟主 CodePipeline 控台，網址為 https://console.aws.amazon.com/codepipeline/。

   在 Pipelines (管道) 下，選擇您的管道，然後選擇 View (檢視)。本圖顯示您的管道來源和部署階段。
2. 選擇編輯管道。
3. 選擇此選項可編輯「部署」階段。
4. 新增將刪除堆疊的部署動作：

   a. 在 [動作名稱] 中，選擇DeleteStack。在部署提供者中，選擇CloudFormation。

   b. 在 Action mode (動作模式) 中，選擇 Delete a stack (刪除堆疊)。

   c. 在 Stack name (堆疊名稱) 中，輸入變數語法，如下所示。這是動作將刪除的堆疊名稱。

   d. 選擇 Done (完成) 以儲存動作。

   e. 選擇 Save (儲存) 以儲存管道。

   管道會在儲存時執行。

教學課程：Amazon ECS 標準部署 CodePipeline

本教學課程可協助您使用 Amazon ECS 建立完整、 end-to-end 持續的部署 (CD) 管道。 CodePipeline

> **Note**
>
> 本教學課程適用於的 Amazon ECS 標準部署動作 CodePipeline。如需在中使用 Amazon ECS 進行 CodeDeploy 藍/綠部署動作的教學課程 CodePipeline，請參閱。教學課程：使用 Amazon ECR 來源和 ECS 到部署建立管道 CodeDeploy

必要條件

您必須先有幾個資源，才能使用此教學來建立 CD 管道。以下是在開始使用前需準備的事項：

> **Note**
>
> 所有這些資源都應該在相同的 AWS 區域內建立。

- 一個源代碼控制存儲庫（本教程使用 CodeCommit）與您的 Dockerfile 和應用程序源。如需詳細資訊，請參閱《AWS CodeCommit 使用指南》中的〈建立 CodeCommit存放庫〉。
- Docker 映像儲存庫 (本教學使用 Amazon ECR)，其中包含您從 Docker 檔案和應用程式來源建立的映像檔。如需詳細資訊，請參閱 Amazon 彈性容器登錄使用者指南中的建立儲存庫和推送映像。
- Amazon ECS 任務定義，參考您映像儲存庫中託管的 Docker 映像檔。如需詳細資訊，請參閱 Amazon 彈性容器服務開發人員指南中的建立任務定義。

> **Important**
>
> 用於的 Amazon ECS 標準部署動作 CodePipeline 會根據 Amazon ECS 服務使用的修訂版，建立自己的任務定義修訂版本。如果您在不更新 Amazon ECS 服務的情況下為任務定義建立新的修訂版本，則部署動作將忽略這些修訂。

以下是本教學課程使用的範例工作定義。您使用的`name`和值`family`將用於建置規格檔案的下一個步驟。

```
{
  "ipcMode": null,
  "executionRoleArn": "role_ARN",
  "containerDefinitions": [
    {
      "dnsSearchDomains": null,
      "environmentFiles": null,
      "logConfiguration": {
        "logDriver": "awslogs",
        "secretOptions": null,
        "options": {
          "awslogs-group": "/ecs/hello-world",
          "awslogs-region": "us-west-2",
          "awslogs-stream-prefix": "ecs"
        }
      },
      "entryPoint": null,
      "portMappings": [
        {
          "hostPort": 80,
          "protocol": "tcp",
          "containerPort": 80
        }
```

```
      ],
      "command": null,
      "linuxParameters": null,
      "cpu": 0,
      "environment": [],
      "resourceRequirements": null,
      "ulimits": null,
      "dnsServers": null,
      "mountPoints": [],
      "workingDirectory": null,
      "secrets": null,
      "dockerSecurityOptions": null,
      "memory": null,
      "memoryReservation": 128,
      "volumesFrom": [],
      "stopTimeout": null,
      "image": "image_name",
      "startTimeout": null,
      "firelensConfiguration": null,
      "dependsOn": null,
      "disableNetworking": null,
      "interactive": null,
      "healthCheck": null,
      "essential": true,
      "links": null,
      "hostname": null,
      "extraHosts": null,
      "pseudoTerminal": null,
      "user": null,
      "readonlyRootFilesystem": null,
      "dockerLabels": null,
      "systemControls": null,
      "privileged": null,
      "name": "hello-world"
    }
  ],
  "placementConstraints": [],
  "memory": "2048",
  "taskRoleArn": null,
  "compatibilities": [
    "EC2",
    "FARGATE"
  ],
  "taskDefinitionArn": "ARN",
```

```
  "family": "hello-world",
  "requiresAttributes": [],
  "pidMode": null,
  "requiresCompatibilities": [
    "FARGATE"
  ],
  "networkMode": "awsvpc",
  "cpu": "1024",
  "revision": 1,
  "status": "ACTIVE",
  "inferenceAccelerators": null,
  "proxyConfiguration": null,
  "volumes": []
}
```

- 執行使用您先前提到的任務定義之服務的 Amazon ECS 叢集。如需詳細資訊，請參閱 Amazon 彈性容器服務開發人員指南中的建立叢集和建立服務。

滿足這些先決條件之後，即可繼續教學並建立 CD 管道。

步驟 1：將組建規格檔案新增至來源儲存庫

本教學課程用 CodeBuild 來建立您的 Docker 映像檔，並將映像推送至 Amazon ECR。將 `buildspec.yml` 檔案新增至來源碼儲存庫，以通知 CodeBuild 如何執行這項作業。下面的範例組建規格執行下列動作：

- 預先建置階段：
  - 登錄 Amazon ECR。
  - 將儲存庫 URI 設定為 ECR 映像，並新增具有來源 Git 遞交 ID 之前七個字元的映像標籤。
- 建置階段：
  - 建置 Docker 影像，並將映像標記為 `latest` 且具有 Git 遞交 ID。
- 後置建置階段：
  - 使用兩個標籤將映像推送至 ECR 儲存庫。
  - 撰寫在組建根目錄`imagedefinitions.json`中呼叫的檔案，其中包含 Amazon ECS 服務的容器名稱以及映像檔和標籤。您 CD 管道的部署階段使用此資訊來建立服務之任務定義的新修訂，接著將服務更新為使用新的任務定義。`imagedefinitions.json` 是 ECS 任務工作者所需的檔案。

貼上此範例文字以建立buildspec.yml檔案，並取代影像和工作定義的值。此文字使用的是帳戶識別碼範例

```
version: 0.2

phases:
  pre_build:
    commands:
      - echo Logging in to Amazon ECR...
      - aws --version
      - aws ecr get-login-password --region $AWS_DEFAULT_REGION | docker login --
username AWS --password-stdin 111122223333.dkr.ecr.us-west-2.amazonaws.com
      - REPOSITORY_URI=012345678910.dkr.ecr.us-west-2.amazonaws.com/hello-world
      - COMMIT_HASH=$(echo $CODEBUILD_RESOLVED_SOURCE_VERSION | cut -c 1-7)
      - IMAGE_TAG=${COMMIT_HASH:=latest}
  build:
    commands:
      - echo Build started on `date`
      - echo Building the Docker image...
      - docker build -t $REPOSITORY_URI:latest .
      - docker tag $REPOSITORY_URI:latest $REPOSITORY_URI:$IMAGE_TAG
  post_build:
    commands:
      - echo Build completed on `date`
      - echo Pushing the Docker images...
      - docker push $REPOSITORY_URI:latest
      - docker push $REPOSITORY_URI:$IMAGE_TAG
      - echo Writing image definitions file...
      - printf '[{"name":"hello-world","imageUri":"%s"}]' $REPOSITORY_URI:$IMAGE_TAG >
 imagedefinitions.json
artifacts:
    files: imagedefinitions.json
```

建置規格是針對中提供的範例任務定義所撰寫的必要條件，Amazon ECS 服務用於本教學課程。REPOSITORY_URI 值對應至 image 儲存庫 (不含任何映像標籤)，而接近檔案結尾的 *hello-world* 值對應至服務任務定義中的容器名稱。

將 **buildspec.yml** 檔案新增至來源儲存庫

1. 開啟文字編輯器，然後將上面的建置規格複製並貼入新的檔案。

2. 將REPOSITORY_URI值 (*012345678910.dkr.ecr.us-west-2.amazonaws.com/hello-world*) 取代為您的碼頭映像檔的 Amazon ECR 儲存庫 URI (不含任何影像標記)。將 *hello-world* 取代為服務任務定義中參考您 Docker 影像的容器名稱。
3. 遞交您的 `buildspec.yml` 檔案並將之推送至來源儲存庫。

   a. 新增檔案。

   ```
   git add .
   ```

   b. 遞交變更。

   ```
   git commit -m "Adding build specification."
   ```

   c. 推送認可。

   ```
   git push
   ```

步驟 2：建立持續部署管道

使用 CodePipeline 精靈建立管線階段，並將來源儲存庫連線至 ECS 服務。

建立管道

1. 請在以下位置開啟 CodePipeline 主控台。 https://console.aws.amazon.com/codepipeline/
2. 在 Welcome (歡迎使用) 頁面上，選擇 Create pipeline (建立管道)。

   如果這是您第一次使用 CodePipeline，則會顯示介紹頁面，而不是「歡迎」。選擇 Get Started Now (立即開始)。
3. 在 [步驟 1：名稱] 頁面上，為管線名稱輸入管線的名稱。在此教學中，管道名稱為 hello-world。
4. 在管線類型中，針對本教學課程的目的選擇 V1。您也可以選擇 V2；但是請注意，管道類型在特性和價格上有所不同。如需詳細資訊，請參閱 管線類型。選擇下一步。
5. 在「步驟 2：新增來源階段」頁面上，針對來源提供者，選擇 AWS CodeCommit。

   a. 針對 Repository name (儲存庫名稱)，選擇要做為管道來源位置的 CodeCommit 儲存庫名稱。

   b. 針對 Branch name (分支名稱)，選擇要使用的分支，然後選擇 Next (下一步)。

6. 在 [步驟 3：新增建置階段] 頁面上，針對 [建置提供者] 選擇 AWS CodeBuild，然後選擇 [建立專案]。

   a. 針對 Project name (專案名稱)，選擇您組建專案的唯一名稱。在此教學中，專案名稱為 hello-world。

   b. 針對 Environment image (環境映像)，選擇 Managed image (受管映像)。

   c. 針對 Operating system (作業系統)，請選擇 Amazon Linux 2。

   d. 針對 Runtime(s) (執行時間)，選擇 Standard (標準)。

   e. 對於「影像」，請選擇**`aws/codebuild/amazonlinux2-x86_64-standard:3.0`**。

   f. 針對 Image version (映像版本) 和 Environment type (環境類型)，請使用預設值。

   g. 選取 Enable this flag if you want to build Docker images or want your builds to get elevated privileges (若想建置 Docker 影像或讓您的建置提升權限，請啟用此標記)。

   h. 取消選取CloudWatch 記錄。您可能需要展開「進階」。

   i. 選擇「繼續」 CodePipeline。

   j. 選擇下一步。

   > **Note**
   >
   > 該嚮導為您的構建項目創建一個 CodeBuild 服務角色，稱為代碼生成 ***`build-project-name`***- 服務角色。請記下此角色名稱，以便稍後在其中新增 Amazon ECR 許可時。

7. 在「步驟 4：新增部署階段」頁面上，針對部署供應商，選擇 Amazon ECS。

   a. 對於叢集名稱，請選擇執行服務的 Amazon ECS 叢集。在此教學中，叢集是 default。

   b. 針對 Service name (服務名稱)，選擇要更新的服務，然後選擇 Next (下一步)。在此教學中，服務名稱為 hello-world。

8. 在 Step 5: Review (步驟 5：檢閱) 頁面上，檢閱您的管道組態，然後選擇 Create pipeline (建立管道) 來建立管道。

   > **Note**
   >
   > 現在已建立管道，將會嘗試執行不同的管道階段。但是，精靈建立的預設 CodeBuild 角色沒有執行`buildspec.yml`檔案中包含的所有命令的權限，因此建置階段會失敗。下節會新增建置階段的許可。

步驟 3：將 Amazon ECR 許可添加到角色 CodeBuild

該 CodePipeline 嚮導為 CodeBuild 構建項目創建了一個 IAM 角色，稱為代碼生成 ***build-project-name***- 服務角色。在本教程中，名稱是 codebuild-hello-world-service-role。由於`buildspec.yml`檔案會呼叫 Amazon ECR API 操作，因此該角色必須具有允許進行這些 Amazon ECR 呼叫的許可政策。下列程序可協助您將適當的許可連接至角色。

若要將 Amazon ECR 許可新增至角色 CodeBuild

1. 前往 https://console.aws.amazon.com/iam/ 開啟 IAM 主控台。
2. 在左側導覽窗格中，選擇 Roles (角色)。
3. 在搜尋方塊中，輸入 codebuild-，然後選擇 CodePipeline 精靈所建立的角色。在本教學課程中，角色名稱是 codebuild-hello-world-service-role。
4. 在 Summary (摘要) 頁面上，選擇 Attach policies (連接政策)。
5. 選取 AmazonEC2 ContainerRegistryPowerUser 政策左側的核取方塊，然後選擇附加政策。

步驟 4：測試管道

您的管道應該擁有執行 end-to-end 原生 AWS 持續部署的一切。現在，將程式碼變更推送至來源儲存庫，以測試其功能。

測試管道

1. 對設定的來源儲存庫進行程式碼變更、遞交並推送變更。
2. 請在以下位置開啟 CodePipeline 主控台。 https://console.aws.amazon.com/codepipeline/
3. 從清單中選擇管道。
4. 觀看管道階段的管道進度。您的管道應該已完成，而 Amazon ECS 服務會執行透過程式碼變更建立的 Docker 映像檔。

教學課程：使用 Amazon ECR 來源和 ECS 到部署建立管道 CodeDeploy

在本教學課程中，您會在 AWS CodePipeline 中設定一個管道，使用支援 Docker 映像的藍/綠部署來部署容器應用程式。在藍/綠部署中，您可以同時啟動新舊版本的應用程式，並在重新路由流量之前測試新版本。您也可以監控部署程序，並在發生問題時快速復原。

> **Note**
>
> 本教學課程適用於 Amazon ECS 的 CodeDeploy 藍色/綠色部署動作。 CodePipeline如需中使用 Amazon ECS 標準部署動作的教學課程 CodePipeline，請參閱教學課程：Amazon ECS 標準部署 CodePipeline。

完成的管道會偵測映像的變更 (儲存在 Amazon ECR 等映像儲存庫中)，並用於 CodeDeploy 將流量路由和部署到 Amazon ECS 叢集和負載平衡器。 CodeDeploy 使用接聽程式將流量重新路由到 AppSpec 檔案中指定的更新容器的連接埠。如需如何在藍/綠部署中使用負載平衡器、生產接聽程式、目標群組和 Amazon ECS 應用程式的詳細資訊，請參閱教學課程：部署 Amazon EC S 服務。

管道也設定為使用來源位置，例如 CodeCommit儲存 Amazon ECS 任務定義的位置。在本教學課程中，您會設定這些AWS資源，然後使用包含每個資源動作的階段來建立管線。

每當原始程式碼變更或將新的基礎映像上傳到 Amazon ECR 時，您的持續交付管道都會自動建立和部署容器映像。

此流程使用以下成品：

- Docker 映像檔，用於指定 Amazon ECR 映像儲存庫的容器名稱和儲存庫 URI。
- 列出您的 Docker 映像名稱、容器名稱、Amazon ECS 服務名稱和負載平衡器組態的 Amazon ECS 任務定義。
- 指定 Amazon ECS 任務定義 CodeDeploy AppSpec 檔案名稱、更新應用程式容器的名稱，以及 CodeDeploy重新路由生產流量的容器連接埠的檔案。它也可以指定選用的網路組態和 Lambda 函數，您可以在部署生命週期事件勾點中執行它們。

> **Note**
>
> 當您對 Amazon ECR 映像儲存庫提交變更時，管道來源動作會為該提交建立`imageDetail.json`檔案。如需 `imageDetail.json` 詳細資訊，請參閱 適用於 Amazon ECS 藍色/綠色部署動作的 imageDetail.json 檔案。

當您建立或編輯管道，以及更新或指定部署階段的來源成品時，請確保指向的來源成品具有您想要使用的最新名稱和版本。在您設定管道之後，若您變更映像或任務定義檔案，則您可能需要在儲存庫中更新來源成品，然後在管道中編輯部署階段。

主題

必要條件

您必須已建立以下資源：

- 一個 CodeCommit 存儲庫。您可以使用在 教學：建立範本管道 (CodeCommit 儲存庫) 中建立的 AWS CodeCommit 儲存庫。
- 啟動 Amazon EC2 Linux 執行個體並安裝 Docker 以建立映像檔，如本教學所示。如果您已擁有想要使用的映像，則可略過此先先決條件。

步驟 1：建立映像並推送至 Amazon ECR 儲存庫

在本節中，您可以使用 Docker 建立映像，然後使用建立 Amazon ECR 儲存庫並將映像推送到存放庫。AWS CLI

> **Note**
>
> 如果您已擁有想要使用的映像，則可略過此先步驟。

建立映像

1. 登入您已在其中安裝 Docker 安裝的 Linux 執行個體。

   拉下 `nginx` 的映像。此指令提供影`nginx:latest`像：

   ```
   docker pull nginx
   ```

2. 執行 docker images。您應該會在清單中看到映像。

```
docker images
```

若要建立 Amazon ECR 儲存庫並推送映像

1. 建立 Amazon ECR 儲存庫，以便存放 映像。記下輸出中的 repositoryUri。

```
aws ecr create-repository --repository-name nginx
```

輸出：

```
{
    "repository": {
        "registryId": "aws_account_id",
        "repositoryName": "nginx",
        "repositoryArn": "arn:aws:ecr:us-east-1:aws_account_id:repository/nginx",
        "createdAt": 1505337806.0,
        "repositoryUri": "aws_account_id.dkr.ecr.us-east-1.amazonaws.com/nginx"
    }
}
```

2. 為映像標記上一步中的 repositoryUri 值。

```
docker tag nginx:latest aws_account_id.dkr.ecr.us-east-1.amazonaws.com/nginx:latest
```

3. 執行命aws ecr get-login-password令，如本範例中所示，針對「us-west-2地區」和 111122223333 帳戶識別碼。

```
aws ecr get-login-password --region us-west-2 | docker login --username AWS --
password-stdin 111122223333.dkr.ecr.us-west-2.amazonaws.com/nginx
```

4. 使用前面的步驟將映像推送到 Amazon ECR。repositoryUri

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/nginx:latest
```

第 2 步：創建任務定義和 AppSpec 源文件並推送到 CodeCommit 存儲庫

在本節中，您會建立任務定義 JSON 檔案，並在 Amazon ECS 中註冊該檔案。然後，您可以為 Git 客戶端創建一個 AppSpec 文件， CodeDeploy 並使用該文件將文件推送到您的 CodeCommit 存儲庫中。

針對您的映像建立任務定義

1. 使用下列內容建立名為 `taskdef.json` 的檔案。對於 `image`，輸入您的映像名稱，例如 nginx。當您的管道執行時，此值即會更新。

> **Note**
>
> 請確定任務定義中指定的執行角色包含 `AmazonECSTaskExecutionRolePolicy`。如需詳細資訊，請參閱 Amazon ECS 開發人員指南中的 Amazon ECS 任務執行 IAM 角色。

```
{
    "executionRoleArn": "arn:aws:iam::account_ID:role/ecsTaskExecutionRole",
    "containerDefinitions": [
        {
            "name": "sample-website",
            "image": "nginx",
            "essential": true,
            "portMappings": [
                {
                    "hostPort": 80,
                    "protocol": "tcp",
                    "containerPort": 80
                }
            ]
        }
    ],
    "requiresCompatibilities": [
        "FARGATE"
    ],
    "networkMode": "awsvpc",
    "cpu": "256",
    "memory": "512",
    "family": "ecs-demo"
}
```

2. 註冊具有 taskdef.json 檔案的任務定義。

```
aws ecs register-task-definition --cli-input-json file://taskdef.json
```

3. 在註冊任務定義之後，請編輯您的檔案以移除映像名稱，並在映像欄位中包含 <IMAGE1_NAME> 預留位置文字。

```
{
    "executionRoleArn": "arn:aws:iam::account_ID:role/ecsTaskExecutionRole",
    "containerDefinitions": [
        {
            "name": "sample-website",
            "image": "<IMAGE1_NAME>",
            "essential": true,
            "portMappings": [
                {
                    "hostPort": 80,
                    "protocol": "tcp",
                    "containerPort": 80
                }
            ]
        }
    ],
    "requiresCompatibilities": [
        "FARGATE"
    ],
    "networkMode": "awsvpc",
    "cpu": "256",
    "memory": "512",
    "family": "ecs-demo"
}
```

建立 AppSpec 檔案的步驟

- 該 AppSpec 檔案用於 CodeDeploy 部署。此檔案會使用下列格式，其中包含選用欄位：

```
version: 0.0
Resources:
  - TargetService:
      Type: AWS::ECS::Service
      Properties:
```

```
        TaskDefinition: "task-definition-ARN"
        LoadBalancerInfo:
          ContainerName: "container-name"
          ContainerPort: container-port-number
# Optional properties
        PlatformVersion: "LATEST"
        NetworkConfiguration:
            AwsvpcConfiguration:
              Subnets: ["subnet-name-1", "subnet-name-2"]
              SecurityGroups: ["security-group"]
              AssignPublicIp: "ENABLED"
Hooks:
- BeforeInstall: "BeforeInstallHookFunctionName"
- AfterInstall: "AfterInstallHookFunctionName"
- AfterAllowTestTraffic: "AfterAllowTestTrafficHookFunctionName"
- BeforeAllowTraffic: "BeforeAllowTrafficHookFunctionName"
- AfterAllowTraffic: "AfterAllowTrafficHookFunctionName"
```

如需有關 AppSpec 檔案的詳細資訊 (包括範例)，請參閱CodeDeploy AppSpec 檔案參考。

使用下列內容建立名為 appspec.yaml 的檔案。對於 TaskDefinition，不要變更 <TASK_DEFINITION> 預留位置文字。當您的管道執行時，此值即會更新。

```
version: 0.0
Resources:
  - TargetService:
      Type: AWS::ECS::Service
      Properties:
        TaskDefinition: <TASK_DEFINITION>
        LoadBalancerInfo:
          ContainerName: "sample-website"
          ContainerPort: 80
```

若要將檔案推送至您的 CodeCommit 儲存庫

1. 將文件推送或上傳到您的 CodeCommit 存儲庫。這些檔案是 Create Pipeline (建立管道) 精靈針對 CodePipeline 中的部署動作所建立的來源成品。在本機目錄中，您的檔案應該如下所示：

```
/tmp
  |my-demo-repo
    |-- appspec.yaml
```

```
    |-- taskdef.json
```

2. 選擇您要用來上傳檔案的方法：

   a. 若要在您的本機電腦上從複製的儲存庫中使用 Git 命令列：

      i. 將目錄變更為您的本機儲存庫：

      ```
      (For Linux, macOS, or Unix) cd /tmp/my-demo-repo
      (For Windows) cd c:\temp\my-demo-repo
      ```

      ii. 請執行下列命令來同時將所有檔案放入階段：

      ```
      git add -A
      ```

      iii. 請執行下列命令來確認檔案並附加確認訊息：

      ```
      git commit -m "Added task definition files"
      ```

      iv. 運行以下命令將文件從本地存儲庫推送到存 CodeCommit 儲庫：

      ```
      git push
      ```

   b. 若要使用 CodeCommit 主控台上傳檔案：

      i. 開啟主 CodeCommit 控台，然後從「儲存庫」清單中選擇您的存放庫。

      ii. 選擇 Add file (新增檔案)，然後選擇 Upload file (上傳檔案)。

      iii. 選擇 Choose file (選擇檔案)，然後瀏覽您的檔案。輸入您的使用者名稱和電子郵件地址來確定變更。選擇 Commit changes (遞交變更)。

      iv. 對於您要上傳的每個檔案重複此步驟。

步驟 3：建立 Application Load Balancer 和目標群組

在本節中，您會建立 Amazon EC2 Application Load Balancer。稍後建立 Amazon ECS 服務時，您可以使用透過負載平衡器建立的子網路名稱和目標群組值。您可以建立應用程式負載平衡器或網路負載平衡器。負載平衡器必須使用 VPC 搭配不同可用區域中的兩個子網路。在這些步驟中，您確認預設 VPC、建立一個負載平衡器，然後為您的負載平衡器建立兩個目標群組。如需詳細資訊，請參閱「網路負載平衡器的目標群組」。

驗證您的預設 VPC 和公有子網路

1. 登入 AWS Management Console，並在 https://console.aws.amazon.com/vpc/ 開啟 Amazon VPC 主控台。
2. 驗證要使用的預設 VPC。在導覽窗格中，選擇 Your VPCs (您的 VPC)。請注意，哪個 VPC 在 Default VPC (預設 VPC) 欄中顯示 Yes (是)。這是預設 VPC。它包含供您選取的預設子網路。
3. 選擇 Subnets (子網路)。選擇兩個在 Default subnet (預設子網路) 欄中顯示 Yes (是) 的子網路。

   > **Note**
   >
   > 記下您的子網路 ID。本教學的稍後部分將需要這些資訊。

4. 選擇子網路，然後選擇 Description (描述) 標籤。驗證您想要使用的子網路是否位於不同的可用區域中。
5. 選擇子網路，然後選擇 Route Table (路由表) 標籤。若要驗證您想要使用的每個子網路是否為公有子網路，請確認閘道資料列包含在路由表中。

若要建立 Amazon EC2 Application Load Balancer

1. 請登入 AWS Management Console，並在 https://console.aws.amazon.com/ec2/ 開啟 Amazon EC2 主控台。
2. 在導覽窗格中，選擇 Load Balancers (負載平衡器)。
3. 選擇 Create Load Balancer (建立負載平衡器)。
4. 選擇 Application Load Balancer (應用程式負載平衡器)，然後選擇 Create (建立)。
5. 在 Name (名稱) 中，輸入負載平衡器的名稱。
6. 在 Scheme (結構描述) 中，選擇 internet-facing (面向網際網路)。
7. 在 IP address type (IP 地址) 中，選擇 ipv4。
8. 為您的負載平衡器設定兩個接聽程式連接埠：
   a. 在 Load Balancer Protocol (負載平衡器通訊協定) 下，選擇 HTTP。在 Load Balancer Port (負載平衡器連接埠) 下，輸入 **80**。
   b. 選擇 Add listener (新增接聽程式)。
   c. 在第二個接聽程式的 Load Balancer Protocol (負載平衡器通訊協定) 下，選擇 HTTP。在 Load Balancer Port (負載平衡器連接埠) 下，輸入 **8080**。

9. 在 Availability Zones (可用區域) 下，於 VPC 中，選擇預設 VPC。接著，選擇您想要使用的兩個預設子網路。
10. 選擇 Next: Configure Security Settings (下一步：設定安全設定)。
11. 選擇 Next: Configure Security Groups (下一步：設定安全群組)。
12. 選擇 Select an existing security group (選取現有的安全群組)，並記下安全群組 ID。
13. 選擇 Next: Configure Routing (下一步：設定路由)。
14. 在 Target group (目標群組) 中，選擇 New target group (新增目標群組)，然後設定您的第一個目標群組：
    a. 在 Name (名稱) 中，輸入目標群組名稱 (例如，**`target-group-1`**)。
    b. 在 Target type (目標類型) 中，選擇 IP。
    c. 在 Protocol (通訊協定) 中，選擇 HTTP。在 Port (連接埠) 中，輸入 **`80`**。
    d. 選擇 Next: Register Targets (下一步：註冊目標)。
15. 選擇 Next: Review (下一步：檢視)，然後選擇 Create (建立)。

為您的負載平衡器建立第二個目標群組

1. 佈建負載平衡器後，開啟 Amazon EC2 主控台。在導覽窗格中，選擇 Target Groups (目標群組)。
2. 選擇 Create target group (建立目標群組)。
3. 在 Name (名稱) 中，輸入目標群組名稱 (例如，**`target-group-2`**)。
4. 在 Target type (目標類型) 中，選擇 IP。
5. 在 Protocol (通訊協定) 中，選擇 HTTP。在 Port (連接埠) 中，輸入 **`8080`**。
6. 在 VPC 中，選擇預設 VPC。
7. 選擇建立。

> **Note**
>
> 您必須具有兩個針對負載平衡器建立的目標群組，才能執行部署。您只需要記下第一個目標群組的 ARN。在下一個步驟中，此 ARN 會用於 `create-service` JSON 檔案中。

更新您的負載平衡器以包含您的第二個目標群組

1. 開啟 Amazon EC2 主控台。在導覽窗格中，選擇 Load Balancers (負載平衡器)。
2. 選擇您的負載平衡器，然後選擇 Listeners (接聽程式) 標籤。選擇使用連接埠 8080 的接聽程式，然後選擇 Edit (編輯)。
3. 選擇 Forward to (轉寄至) 旁邊的鉛筆圖示。選擇您的第二個目標群組，然後選擇核取記號。選擇 Update (更新) 來儲存更新。

步驟 4：建立您的 Amazon ECS 叢集和服務

在本節中，您會建立 Amazon ECS 叢集和服務，以便在部署期間將流量 CodeDeploy 路由傳送到 Amazon ECS 叢集，而不是 EC2 執行個體。若要建立 Amazon ECS 服務，您必須使用您使用負載平衡器建立的子網路名稱、安全群組和目標群組值來建立服務。

> **Note**
>
> 當您使用這些步驟建立 Amazon ECS 叢集時，您可以使用佈建 AWS Fargate 容器的僅限聯網叢集範本。 AWSFargate 是一項為您管理容器執行個體基礎架構的技術。您不需要為您的 Amazon ECS 叢集選擇或手動建立 Amazon EC2 執行個體。

建立 Amazon ECS 叢集

1. 開啟 Amazon ECS 傳統主控台，網址為 https://console.aws.amazon.com/ecs/。
2. 在導覽窗格中，選擇叢集。
3. 選擇 建立叢集。
4. 選擇使用 AWS Fargate 的僅限網路連線叢集範本，然後選擇 [下一步]。
5. 在 Configure cluster (設定叢集) 頁面上輸入叢集名稱。您可以為您的資源新增選用的標籤。選擇建立。

要創建一個 Amazon ECS 服務

使用在 AWS CLI Amazon ECS 中創建您的服務。

1. 建立 JSON 檔案，並將其命名為 `create-service.json`。將下列內容貼入 JSON 檔案中。

對於該taskDefinition欄位，當您在 Amazon ECS 中註冊任務定義時，您會給它一個系列。這類似用於多個任務定義版本的名稱，以修訂版號碼指定。在此範例中，請將「ecs-demo:1」用於您檔案中的系列和修訂版號碼。在 步驟 3：建立 Application Load Balancer 和目標群組 中使用您建立的子網路名稱、安全群組和目標群組值，以及您的負載平衡器。

> **Note**
>
> 您需要在這個檔案包含您的目標群組 ARN。開啟 Amazon EC2 主控台，然後從導覽窗格的「負載平衡」下選擇「目標群組」。選擇您的第一個目標群組。從 Description (描述) 標籤複製您的 ARN。

```
{
    "taskDefinition": "family:revision-number",
    "cluster": "my-cluster",
    "loadBalancers": [
        {
            "targetGroupArn": "target-group-arn",
            "containerName": "sample-website",
            "containerPort": 80
        }
    ],
    "desiredCount": 1,
    "launchType": "FARGATE",
    "schedulingStrategy": "REPLICA",
    "deploymentController": {
        "type": "CODE_DEPLOY"
    },
    "networkConfiguration": {
        "awsvpcConfiguration": {
            "subnets": [
                "subnet-1",
                "subnet-2"
            ],
            "securityGroups": [
                "security-group"
            ],
            "assignPublicIp": "ENABLED"
        }
    }
```

```
}
```

2. 執行 create-service 命令，指定 JSON 檔案：

> **Important**
>
> 請確認在檔案名稱之前包含 `file://`。這是此命令必要項目。

此範例會建立名為 `my-service` 的服務。

> **Note**
>
> 此範例命令會建立名為 my-service 的服務。如果您已經有使用此名稱的服務，此命令會傳回錯誤。

```
aws ecs create-service --service-name my-service --cli-input-json file://create-service.json
```

輸出會傳回您服務的描述欄位。

3. 執行 describe-services 命令，以驗證是否已建立您的服務。

```
aws ecs describe-services --cluster cluster-name --services service-name
```

步驟 5：建立您的 CodeDeploy 應用程式和部署群組 (ECS 運算平台)

當您為 Amazon ECS 運算平台建立 CodeDeploy 應用程式和部署群組時，會在部署期間使用該應用程式來參考正確的部署群組、目標群組、偵聽程式和流量重新路由行為。

若要建立 CodeDeploy 應用程式

1. 開啟 CodeDeploy 主控台並選擇 [建立應用程式]。
2. 在 Application name (應用程式名稱) 中，輸入您想要使用的名稱。
3. 在 Compute Platform (運算平台) 中，選擇 Amazon ECS。
4. 選擇 建立應用程式。

若要建立 CodeDeploy 部署群組

1. 在您應用程式頁面的 Deployment groups (部署群組) 標籤上，選擇 Create deployment group (建立部署群組)。
2. 在 Deployment group name (部署群組名稱) 中，輸入描述部署群組的名稱。
3. 在服務角色中，選擇授與 Amazon ECS CodeDeploy 存取權的服務角色。若要建立新的服務角色，請遵循下列步驟：

   a. 在 https://console.aws.amazon.com/iam/ 開啟身分與存取權管理主控台。

   b. 從主控台儀表板，選擇 Roles (角色)。

   c. 選擇建立角色。

   d. 在 [選取信任實體的類型] 底下，選取AWS 服務。在 [選擇使用案例] 下，選取CodeDeploy。在 [選取您的使用案例] 下，選取 [CodeDeploy -ECS]。選擇 Next: Permissions (下一步：許可)。`AWSCodeDeployRoleForECS` 受管政策已連接至角色。

   e. 選擇 Next: Tags (下一步：標籤)，然後選擇 Next: Review (下一步：檢閱)。

   f. 輸入角色的名稱 (例如 **`CodeDeployECSRole`**)，然後選擇 Create role (建立角色)。
4. 在環境組態中，選擇您的 Amazon ECS 叢集名稱和服務名稱。
5. 從負載平衡器中，選擇負載平衡器的名稱，該平衡器為 Amazon ECS 服務提供流量。
6. 從 Production listener port (生產接聽程式連接埠) 中，為接聽程式選擇連接埠和通訊協定，而此接聽程式會對 Amazon ECS 服務提供生產流量。從 Test listener port (測試接聽程式連接埠)，為測試接聽程式選擇連接埠和通訊協定。
7. 從 Target group 1 name (目標群組 1 名稱) 和 Target group 2 name (目標群組 2 名稱) 中，選擇在部署期間用來路由流量的目標群組。確定這些是您為負載平衡器建立的目標群組。
8. 選擇「立即重新路由傳送流量」，以決定成功部署後多久，將流量重新路由到您更新的 Amazon ECS 任務。
9. 選擇 Create deployment group (建立部署群組)。

步驟 6：建立管道

在本節中，您可以採取下列動作建立管道：

- 來源人工因素為 CodeCommit 作業定義與 AppSpec 檔案的動作。
- 具有 Amazon ECR 來源動作的來源階段，其中來源成品為映像檔。
- 具有 Amazon ECS 部署動作的部署階段，部署會與 CodeDeploy 應用程式和部署群組一起執行。

使用精靈建立兩階段管道

1. 請登入AWS Management Console並開啟 CodePipeline 主控台，網址為 http://console.aws.amazon.com/codesuite/codepipeline/home。
2. 在 Welcome (歡迎) 頁面、Getting started (入門) 頁面、或者 Pipelines (管道) 頁面上，選擇 Create pipeline (建立管道)。
3. 在 Step 1: Choose pipeline settings (步驟 1：選擇管道設定) 的 Pipeline name (管道名稱) 中，輸入 **`MyImagePipeline`**。
4. 在管線類型中，針對本教學課程的目的選擇 V1。您也可以選擇 V2；但是請注意，管道類型在特性和價格上有所不同。如需詳細資訊，請參閱管線類型。
5. 在 Service role (服務角色)，選擇 New service role (新增服務角色)，以允許 CodePipeline 在 IAM 中建立服務角色。
6. 將 Advanced settings (進階設定) 下的設定保留為預設值，然後選擇 Next (下一步)。
7. 在 Step 2: Add source stage (步驟 2：新增來源階段) 的 Source provider (來源提供者) 中，選擇 AWS CodeCommit。在 Repository name (儲存庫名稱) 中，選擇您在 步驟 1：建立 CodeCommit 儲存庫 中建立的 CodeCommit 儲存庫名稱。在 Branch name (分支名稱) 中，選擇包含最新程式碼更新的分支名稱。

   選擇下一步。
8. 在 Step 3: Add build stage (步驟 3：新增建置階段) 中，選擇 Skip build stage (跳過建置階段)，然後再次選擇 Skip (跳過) 來接受警告訊息。選擇下一步。
9. 在 Step 4: Add deploy stage (步驟 4：新增部署階段) 中：

   a. 在 Deploy provider (部署提供者) 中，選擇 Amazon ECS (Blue/Green) (Amazon ECS (藍/綠))。在 Application name (應用程式名稱) 中，輸入應用程式名稱或從清單中選擇應用程式名稱，例如，`codedeployapp`。在 Deployment group (部署群組) 中，輸入部署群組名稱或從清單中選擇部署群組名稱，例如，`codedeploydeplgroup`。

   > **Note**
   >
   > 名稱「Deploy」(部署)，是預設指定給在 Step 4: Deploy (步驟 4：部署) 步驟中建立的階段名稱，如同「Source」(來源) 是管道的第一階段所指定的名稱。

   b. 在 Amazon ECS 任務定義下，選擇SourceArtifact。在欄位中，輸入 **`taskdef.json`**。

c. 在「AWS CodeDeploy AppSpec 檔案」下方，選擇SourceArtifact。在欄位中，輸入 **`appspec.yaml`**。

> **Note**
>
> 此時，不要在 Dynamically update task definition image (動態更新任務定義映像) 下填寫任何資訊。

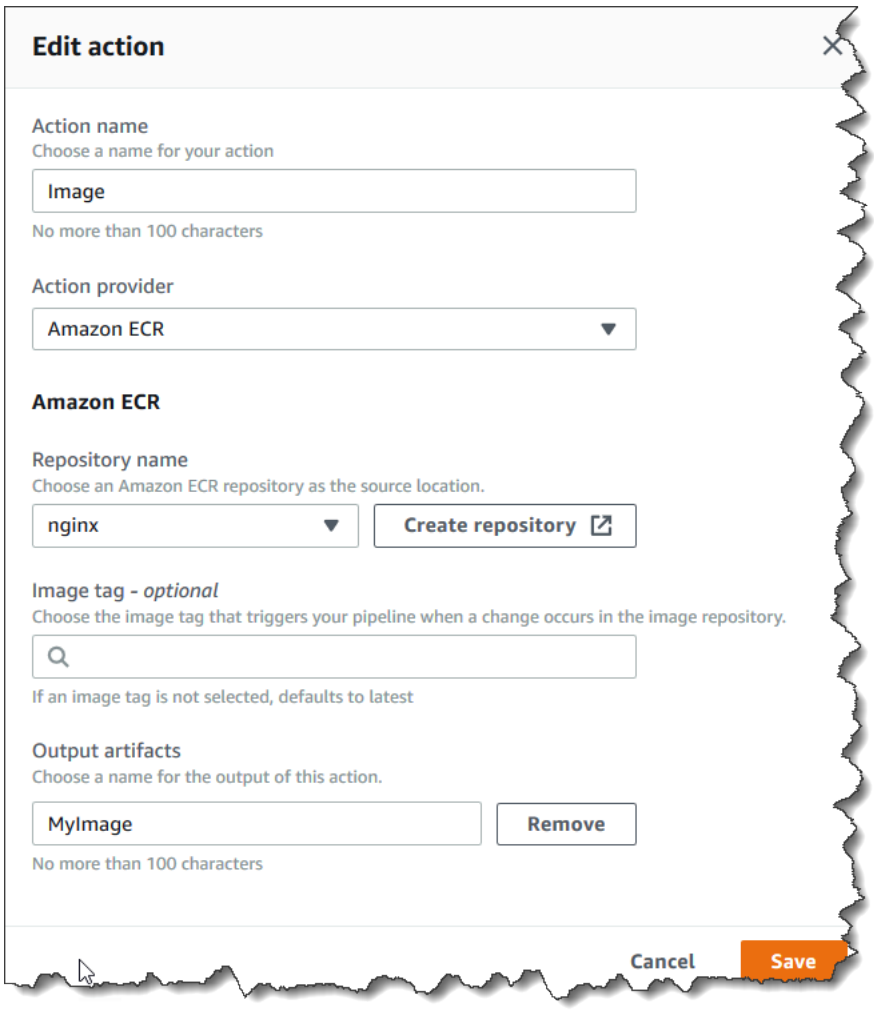
d. 選擇下一步。

10. 在 Step 5: Review (步驟 5：檢閱) 中，檢閱資訊，然後選擇 Create pipeline (建立管道)。

若要將 Amazon ECR 來源動作新增至您的管道

檢視您的管道，並將 Amazon ECR 來源動作新增至您的管道。

1. 選擇您的管道。在左上角，選擇 Edit (編輯)。
2. 在來源階段中，選擇 Edit stage (編輯階段)。
3. 通過選擇 CodeCommit 源操作旁邊的 + 添加操作添加 parallel 操作。
4. 在 Action name (動作名稱) 中，輸入名稱 (例如，**`Image`**)。
5. 在Action provider (動作提供者)中，選擇 Amazon ECR。



6. 在儲存庫名稱中，選擇 Amazon ECR 儲存庫的名稱。
7. 在 Image tag (映像標籤) 中，指定映像名稱和版本，如果與最新不同的話。
8. 在 Output artifacts (輸出成品) 中，選擇輸出成品預設值 (例如 `MyImage`)，其中包含您要下一個階段使用的映像名稱和儲存庫 URI 資訊。
9. 在動作畫面上選擇 Save (儲存)。在階段畫面上選擇 Done (完成)。在管道上選擇 Save (儲存)。會出現一則訊息，顯示要針對 Amazon ECR 來源動作建立的 Amazon CloudWatch 事件規則。

將您的來源成品連線至部署動作

1. 在您的部署階段選擇編輯，然後選擇圖示以編輯 Amazon ECS (藍色/綠色) 動作。
2. 捲動到窗格底部。在 Input artifacts (輸入成品) 中，選擇 Add (新增)。從新的 Amazon ECR 儲存庫新增來源成品 (例如，`MyImage`)。
3. 在「作業定義」中，選擇 SourceArtifact，然後確認**`taskdef.json`**是否已輸入。

4. 在AWS CodeDeploy AppSpec 檔案中，選擇 SourceArtifact，然後確認**appspec.yaml**已輸入。
5. 在動態更新工作定義影像的使用影像 URI 的輸入 Artifact 影中 MyImage，選擇，然後輸入taskdef.json檔案中使用的預留位置文字： **IMAGE1_NAME**。選擇儲存。
6. 在 AWS CodePipeline 窗格中，選擇 Save pipeline change (儲存管道變更)，然後選擇 Save change (儲存變更)。檢視已更新的管道。

   建立此範例管道後，主控台項目的動作組態會顯示在管道結構中，如下所示：

```
"configuration": {
  "AppSpecTemplateArtifact": "SourceArtifact",
  "AppSpecTemplatePath": "appspec.yaml",
  "TaskDefinitionTemplateArtifact": "SourceArtifact",
  "TaskDefinitionTemplatePath": "taskdef.json",
  "ApplicationName": "codedeployapp",
  "DeploymentGroupName": "codedeploydeplgroup",
  "Image1ArtifactName": "MyImage",
  "Image1ContainerName": "IMAGE1_NAME"
},
```

7. 若要提交您的變更並啟動管道建置，請選擇 Release change (發行變更)，然後選擇 Release (發行)。
8. 選擇要檢視的部署動作， CodeDeploy 並查看流量轉移的進度。

> **Note**
>
> 您可以查看顯示選用等待時間的部署步驟。依預設，會在成功部署後 CodeDeploy 等待一小時，才終止原始工作集。您可以利用這段時間復原或終止任務，但在任務集終止時，您的部署任務會以別的方式完成。

步驟 7：變更您的管道並驗證部署

對映像進行變更，然後將變更推送至 Amazon ECR 儲存庫。這會觸發您的管道執行。驗證是否已部署您的映像來源。

教學：建立部署 Amazon Alexa 技能的管道

在此教學課程中，您會設定管道在您的部署階段中，將 Alexa Skills Kit 做為部署提供者，來持續交付您的 Alexa 技能。當您變更來源儲存庫中的來源檔案時，完整的管道便會偵測到您的技能變更。管道稍後會使用 Alexa Skills Kit 來部署至 Alexa 技能開發階段。

> **Note**
>
> 此功能不適用於亞太區域 (香港) 或歐洲 (米蘭) 地區。若要使用該區域中可用的其他部署動作，請參閱部署動作整合。

若要將您的自訂技能建立為 Lambda 函數，請參閱將自訂技能託管為 AWS Lambda 函數。您也可以建立使用 Lambda 來源檔 CodeBuild 案的管道，以及針對您的技能將變更部署至 Lambda 的專案。例如，若要建立新技能和相關 Lambda 功能，您可以建立 AWS CodeStar 專案。請參閱在 AWS CodeStar 中建立 Alexa 技能專案。對於該選項，管道包括具有 CodeBuild 動作和在部署階段中用於 AWS CloudFormation 之動作的第三建置階段。

必要條件

您必須已擁有下列各項目：

- 一個 CodeCommit 存儲庫。您可以使用在 教學：建立範本管道 (CodeCommit 儲存庫) 中建立的 AWS CodeCommit 儲存庫。
- Amazon 開發人員帳戶。這是擁有您 Alexa 技能的帳戶。您可以在 Alexa Skills Kit 免費建立帳戶。
- Alexa 技能。您可以依據取得自訂技能範本程式碼教學課程建立範例技能。
- 安裝 ASK CLI 並使用`ask init`您的AWS憑據進行配置。請參閱安裝並初始化 ASK CLI。

步驟 1：建立 Alexa 開發人員服務 LWA 安全性描述檔

在本節，您會建立安全性描述檔，來搭配 Login with Amazon (LWA) 使用。如果您已有描述檔，則可以略過此步驟。

- 使用 generate-lwa-tokens 中的步驟建立安全性描述檔。
- 建立描述檔後，請記下 Client ID (用戶端 ID) 和 Client Secret (用戶端密碼)。
- 請確定您如說明所述輸入 Allowed Return URLs (允許的傳回 URL)。URL 允許 ASK CLI 命令重新導向重新整理字符請求。

步驟 2：創建 Alexa 技能源文件並推送到您的 CodeCommit 存儲庫

在本節，您會建立並推送 Alexa 技能來源檔案到管道用於來源階段的儲存庫。對於您已在 Amazon 開發人員主控台中建立的技能，您會產生和推送下列項目：

- `skill.json` 檔案。
- `interactionModel/custom` 資料夾。

> **Note**
>
> 此目錄結構符合 Alexa Skills Kit 技能套件格式要求，如技能套件格式中所述。如果目錄結構未使用正確的技能套件格式，變更不會順利部署到 Alexa Skills Kit 主控台。

為您的技能建立來源檔案

1. 從 Alexa Skills Kit 開發人員主控台擷取您的技能 ID。使用此命令：

   ```
   ask api list-skills
   ```

   依名稱找出您的技能，然後複製 `skillId` 欄位中已關聯的 ID。
2. 產生包含您的技能詳細資訊的 `skill.json` 檔案。使用此命令：

   ```
   ask api get-skill -s skill-ID > skill.json
   ```
3. (選用) 建立 `interactionModel/custom` 資料夾。

   使用此命令產生資料夾內的互動模型檔案。在地區設定上，此教學使用 en-US 做為檔案名稱中的地區設定。

   ```
   ask api get-model --skill-id skill-ID --locale locale >
       ./interactionModel/custom/locale.json
   ```

若要將檔案推送至您的 CodeCommit 儲存庫

1. 將文件推送或上傳到您的 CodeCommit 存儲庫。這些檔案是 Create Pipeline (建立管道) 精靈針對 AWS CodePipeline 中的部署動作所建立的來源成品。在本機目錄中，您的檔案應該如下所示：

```
skill.json
/interactionModel
  /custom
    |en-US.json
```

2. 選擇您要用來上傳檔案的方法：

   a. 若要在您的本機電腦上從複製的儲存庫中使用 Git 命令列：

      i. 請執行下列命令來同時將所有檔案放入階段：

      ```
      git add -A
      ```

      ii. 請執行下列命令來確認檔案並附加確認訊息：

      ```
      git commit -m "Added Alexa skill files"
      ```

      iii. 運行以下命令將文件從本地存儲庫推送到存 CodeCommit 儲庫：

      ```
      git push
      ```

   b. 若要使用 CodeCommit 主控台上傳檔案：

      i. 開啟主 CodeCommit 控台，然後從「儲存庫」清單中選擇您的存放庫。

      ii. 選擇 Add file (新增檔案)，然後選擇 Upload file (上傳檔案)。

      iii. 選擇 Choose file (選擇檔案)，然後瀏覽您的檔案。輸入您的使用者名稱和電子郵件地址來確定變更。選擇 Commit changes (遞交變更)。

      iv. 對於您要上傳的每個檔案重複此步驟。

步驟 3：使用 ASK CLI 命令建立重新整理字符

CodePipeline 根據 Amazon 開發人員帳戶中的用戶端 ID 和密碼使用重新整理權杖來授權其代表您執行的動作。在本節中，您可以使用 ASK CLI 來建立字符。當您使用 Create Pipeline (建立管道) 精靈時，您會使用這些登入資料。

使用您的 Amazon 開發人員帳戶登入資料建立重新整理字符

1. 使用下列命令：

```
ask util generate-lwa-tokens
```

2. 出現提示時，如此範例所示，輸入您的用戶端 ID 和密碼：

```
? Please type in the client ID:
amzn1.application-client.example112233445566
? Please type in the client secret:
example112233445566
```

3. 登入瀏覽器頁面隨即顯示。使用您的 Amazon 開發人員帳戶登入資料登入。
4. 返回命令列畫面。存取字符和重新整理字符會在輸出中產生。複製輸出中傳回的重新整理字符。

步驟 4：建立管道

在本節中，您可以採取下列動作建立管道：

- 帶有 CodeCommit 動作的源階段，其中源成品是支持您技能的 Alexa 技能文件。
- 具有 Alexa Skills Kit 部署動作的部署階段。

使用精靈建立管道

1. 請登入AWS Management Console並開啟 CodePipeline 主控台，網址為 http://console.aws.amazon.com/codesuite/codepipeline/home。
2. 選擇您想要建立專案及其資源的 AWS 區域。下列區域才能取得 Alexa 技能執行時間：
   - 亞太區域 (東京)
   - 歐洲 (愛爾蘭)
   - 美國東部 (維吉尼亞北部)
   - 美國西部 (奧勒岡)
3. 在 Welcome (歡迎) 頁面、Getting started (入門) 頁面、或者 Pipelines (管道) 頁面上，選擇 Create pipeline (建立管道)。
4. 在 Step 1: Choose pipeline settings (步驟 1：選擇管道設定) 的 Pipeline name (管道名稱) 中，輸入 **MyAlexaPipeline**。
5. 在管線類型中，針對本教學課程的目的選擇 V1。您也可以選擇 V2；但是請注意，管道類型在特性和價格上有所不同。如需詳細資訊，請參閱管線類型。

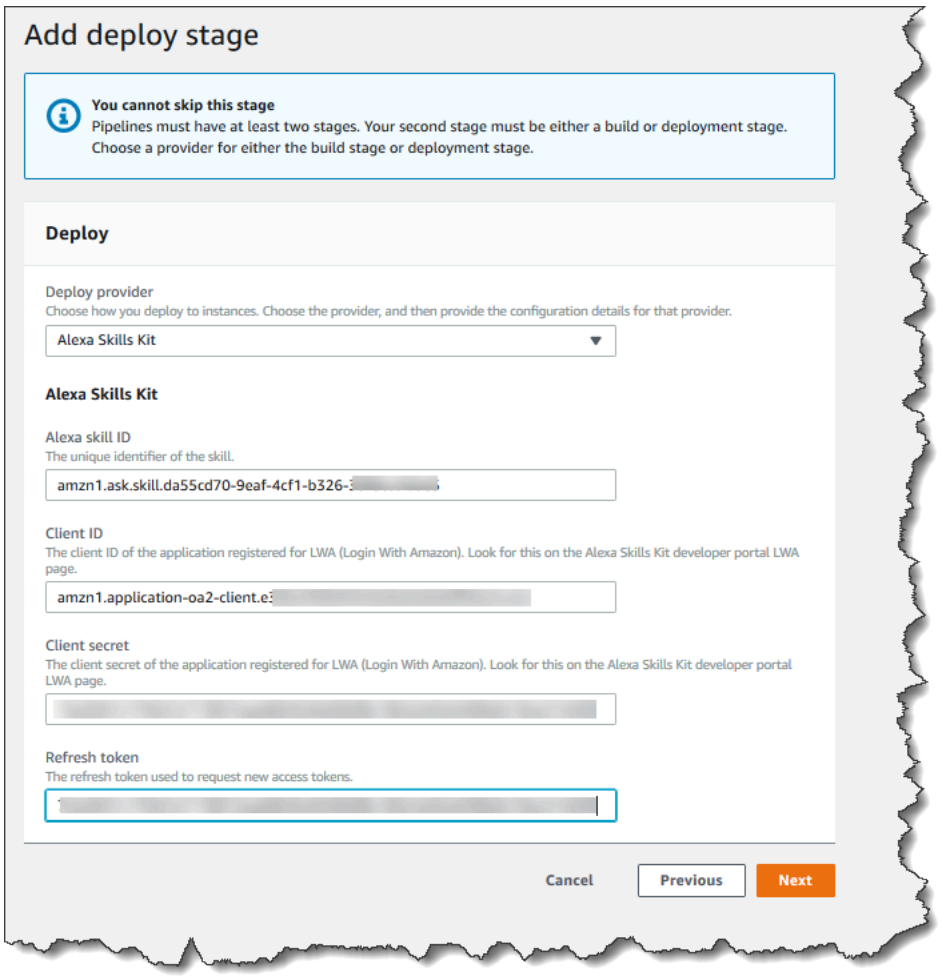
6. 在 Service role (服務角色)，選擇 New service role (新增服務角色)，以允許 CodePipeline 在 IAM 中建立服務角色。
7. 將 Advanced settings (進階設定) 下的設定保留為預設值，然後選擇 Next (下一步)。
8. 在 Step 2: Add source stage (步驟 2：新增來源階段) 的 Source provider (來源提供者) 中，選擇 AWS CodeCommit。在 Repository name (儲存庫名稱) 中，選擇您在 步驟 1：建立 CodeCommit 儲存庫 中建立的 CodeCommit 儲存庫名稱。在 Branch name (分支名稱) 中，選擇包含最新程式碼更新的分支名稱。

   選取儲存庫名稱和分支後，會出現一則訊息，顯示要為此管道建立的 Amazon E CloudWatch vents 規則。

   選擇下一步。
9. 在 Step 3: Add build stage (步驟 3：新增建置階段) 中，選擇 Skip build stage (跳過建置階段)，然後再次選擇 Skip (跳過) 來接受警告訊息。

   選擇下一步。
10. 在 Step 4: Add deploy stage (步驟 4：新增部署階段) 中：

    a. 在 Deploy provider (部署提供者) 中，選擇 Alexa Skills Kit。
    b. 在 Alexa skill ID (Alexa 技能 ID) 中，輸入指派給您在 Alexa Skills Kit 開發人員主控台中技能的技能 ID。
    c. 在 Client ID (用戶端 ID) 中，輸入您註冊之應用程式的 ID。
    d. 在 Client secret (用戶端密碼) 中，輸入您在註冊時選擇的密碼。
    e. 在 Refresh token (重新整理字符) 中，輸入您在步驟 3 產生的字符。



f. 選擇下一步。

11. 在 Step 5: Review (步驟 5：檢閱) 中，檢閱資訊，然後選擇 Create pipeline (建立管道)。

步驟 5：變更任一來源檔案並驗證部署

變更您的技能，然後將變更推送至您的儲存庫。這會觸發您的管道執行。驗證您的技能已在 Alexa Skills Kit 開發人員主控台中更新。

教學課程：建立使用 Amazon S3 做為部署供應商的管道

在本教學中，您將設定管道，在部署階段使用 Amazon S3 做為部署動作提供者持續交付檔案。當您變更來源儲存庫中的來源檔案時，完整的管道便會偵測到變更。接著，管道會使用 Amazon S3 將檔案部署到儲存貯體。每次您在來源位置修改或新增網站檔案時，部署都會以最新檔案建立網站。

> **Note**
>
> 即使您從來源儲存庫刪除檔案，S3 部署動作也不會刪除與已刪除檔案對應的 S3 物件。

本教學課程提供兩種選項：

- 建立將靜態網站部署到您的 S3 公有儲存貯體的管道。此範例會建立具有AWS CodeCommit來源動作和 Amazon S3 部署動作的管道。請參閱 選項 1：將靜態網站檔案部署到 Amazon S3。
- 建立管道，將範例 TypeScript 程式碼編譯成， JavaScript 並將 CodeBuild 輸出成品部署到 S3 儲存貯體以進行存檔。此範例會建立具有 Amazon S3 來源動作、 CodeBuild 建立動作和 Amazon S3 部署動作的管道。請參閱 選項 2：從 S3 來源儲存貯體將建置的存檔檔案部署到 Amazon S3。

> **Important**
>
> 您在此程序中新增至管線的許多動作都涉及建立管線之前需要建立的AWS資源。 AWS來源動作的資源必須始終建立在您建立管道的相同AWS區域中。例如，如果您在美國東部 (俄亥俄) 區域建立管道，則您的 CodeCommit 存放庫必須位於美國東部 (俄亥俄) 區域。
> 您可以在建立管道時新增跨區域動作。 AWS跨區域作業的資源必須位於您計劃執行作業的相同「AWS區域」中。如需詳細資訊，請參閱在 CodePipeline 中新增跨區域動作。

選項 1：將靜態網站檔案部署到 Amazon S3

在此範例中，您會下載範例靜態網站範本檔案、上傳檔案到您的 AWS CodeCommit 儲存庫、建立儲存貯體，以及設定它，以便託管。接下來，您可以使用AWS CodePipeline主控台建立管道，並指定 Amazon S3 部署組態。

必要條件

您必須已擁有下列各項目：

- 一個 CodeCommit 存儲庫。您可以使用在 教學：建立範本管道 (CodeCommit 儲存庫) 中建立的 AWS CodeCommit 儲存庫。
- 您靜態網站的來源檔案。使用此連結來下載 範例靜態網站。sample-website.zip 下載會產生下列檔案：
  - `index.html` 檔案

  - `main.css` 檔案
  - `graphic.jpg` 檔案
- 處理網站託管的 S3 儲存貯體。請參閱託管於 Amazon S3 的靜態網站。確定您建立的儲存貯體與管道位於同一個區域。

> **Note**
>
> 為託管網站，儲存貯體必擁有公有讀取存取權，藉此授予每個人讀取存取權。除了網站託管以外，您應該保留封鎖公有存取 S3 儲存貯體的預設存取設定。

步驟 1：將源文件推送到您的 CodeCommit 存儲庫

在本節，請推送來源檔案到管道用於來源階段的儲存庫。

若要將檔案推送至您的 CodeCommit 儲存庫

1. 將下載的範例檔案解壓縮。請勿將 ZIP 檔案上傳到您的儲存庫。
2. 將文件推送或上傳到您的 CodeCommit 存儲庫。這些檔案是 Create Pipeline (建立管道) 精靈針對 CodePipeline 中的部署動作所建立的來源成品。在本機目錄中，您的檔案應該如下所示：

```
index.html
main.css
graphic.jpg
```

3. 您可以使用 Git 或 CodeCommit 控制台上傳文件：
   a. 若要在您的本機電腦上從複製的儲存庫中使用 Git 命令列：
      i. 請執行下列命令來同時將所有檔案放入階段：

```
git add -A
```

      ii. 請執行下列命令來確認檔案並附加確認訊息：

```
git commit -m "Added static website files"
```

      iii. 運行以下命令將文件從本地存儲庫推送到存 CodeCommit 儲庫：

```
git push
```

b. 若要使用 CodeCommit 主控台上傳檔案：

i. 開啟主 CodeCommit 控台，然後從「儲存庫」清單中選擇您的存放庫。

ii. 選擇 Add file (新增檔案)，然後選擇 Upload file (上傳檔案)。

iii. 選取 Choose file (選擇檔案)，然後瀏覽您的檔案。輸入您的使用者名稱和電子郵件地址來確定變更。選擇 Commit changes (遞交變更)。

iv. 對於您要上傳的每個檔案重複此步驟。

步驟 2：建立管道

在本節中，您可以採取下列動作建立管道：

- 具有 CodeCommit 動作的來源階段，其中來源成品是您網站的檔案。
- 具有 Amazon S3 部署動作的部署階段。

使用精靈建立管道

1. 請登入AWS Management Console並開啟 CodePipeline 主控台，網址為 http://console.aws.amazon.com/codesuite/codepipeline/home。
2. 在 Welcome (歡迎) 頁面、Getting started (入門) 頁面、或者 Pipelines (管道) 頁面上，選擇 Create pipeline (建立管道)。
3. 在 Step 1: Choose pipeline settings (步驟 1：選擇管道設定) 的 Pipeline name (管道名稱) 中，輸入 **`MyS3DeployPipeline`**。
4. 在管線類型中，針對本教學課程的目的選擇 V1。您也可以選擇 V2；但是請注意，管道類型在特性和價格上有所不同。如需詳細資訊，請參閱管線類型。
5. 在 [服務角色] 中，選擇 [新增服務角色] CodePipeline 以允許在 IAM 中建立服務角色。
6. 將 Advanced settings (進階設定) 下的設定保留為預設值，然後選擇 Next (下一步)。
7. 在 Step 2: Add source stage (步驟 2：新增來源階段) 的 Source provider (來源提供者) 中，選擇 AWS CodeCommit。在 Repository name (儲存庫名稱) 中，選擇您在 步驟 1：建立 CodeCommit 儲存庫 中建立的 CodeCommit 儲存庫名稱。在 Branch name (分支名稱) 中，選擇包含最新程式碼更新的分支名稱。除非您自己建立不同分支，否則僅能使用 `main`。

   選取儲存庫名稱和分支後，會顯示要為此管道建立的 Amazon E CloudWatch vents 規則。

選擇下一步。

8. 在 Step 3: Add build stage (步驟 3：新增建置階段) 中，選擇 Skip build stage (跳過建置階段)，然後再次選擇 Skip (跳過) 來接受警告訊息。

   選擇下一步。

9. 在 Step 4: Add deploy stage (步驟 4：新增部署階段) 中：

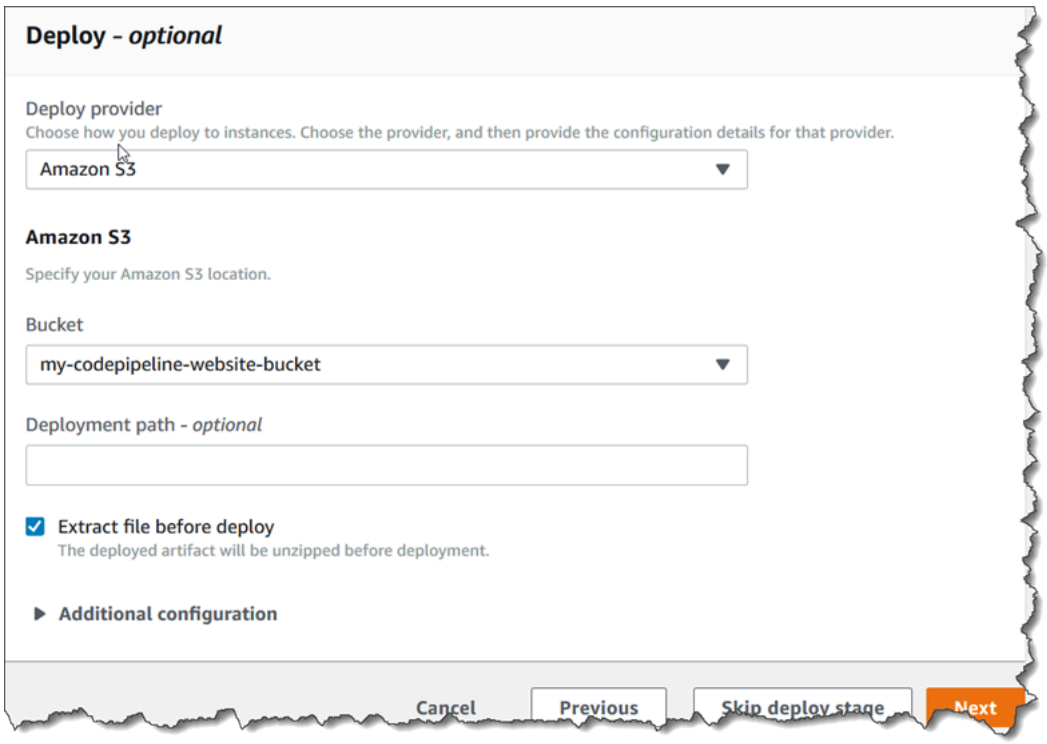
   a. 在Deploy provider (部署提供者)中，選擇 Amazon S3。

   b. 在 Bucket (儲存貯體) 中，輸入您公有儲存貯體的名稱。

   c. 選取 Extract file before deploy (部署前解壓縮檔案)。

> **Note**
>
> 如果您沒有選取 Extract file before deploy (部署前解壓縮檔案)，部署則會失敗。這是因為您管道中的 AWS CodeCommit 動作會壓縮來源成品，且您的檔案是 ZIP 檔案。

選取 Extract file before deploy (部署前解壓縮檔案) 時，則會顯示 Deployment path (部署路徑)。請輸入您要使用的路徑的名稱。這會在 Amazon S3 中建立檔案擷取到的資料夾結構。在本教學課程中，請將此欄位保留空白。



d. (選用) 在 Canned ACL (固定的 ACL) 中，您可以將一組預先定義的授與 (稱為固定的 ACL) 套用至上傳的成品。

e. (選用) 在 Cache control (快取控制) 中，輸入快取參數。您可以將此設為控制請求/回應的快取行為。如需有效值，請參閱 HTTP 操作的 `Cache-Control` 標頭欄位。

f. 選擇下一步。

10. 在 Step 5: Review (步驟 5：檢閱) 中，檢閱資訊，然後選擇 Create pipeline (建立管道)。
11. 管道成功執行後，開啟 Amazon S3 主控台並確認檔案出現在公有儲存貯體中，如下所示：

```
index.html
main.css
graphic.jpg
```

12. 存取您的端點，以測試網站。您的端點遵循此格式：`http://bucket-name.s3-website-region.amazonaws.com/`。

    範例端點：`http://my-bucket.s3-website-us-west-2.amazonaws.com/`。

    範例網頁隨即出現。

步驟 3：變更任一來源檔案並驗證部署

變更您的來源檔案，然後將變更推送至您的儲存庫。這會觸發您的管道執行。驗證您的網站已更新。

選項 2：從 S3 來源儲存貯體將建置的存檔檔案部署到 Amazon S3

在此選項中，建置階段中的建置命令會將 TypeScript 程式碼編譯成程 JavaScript 式碼，並將輸出部署到個別的時間戳記資料夾下的 S3 目標儲存貯體。首先，您創建 TypeScript 代碼和構建規格 .yml 文件。將來源檔案合併為 ZIP 檔案之後，您可以將來源 ZIP 檔案上傳到 S3 來源儲存貯體，並使用 CodeBuild 階段將建置的應用程式 ZIP 檔案部署到 S3 目標儲存貯體。經過編譯的程式碼會在您的目標儲存貯體中做為存檔保留。

必要條件

您必須已擁有下列各項目：

- S3 來源儲存貯體。您可以使用您在 教學：建立簡易管道 (S3 儲存貯體) 中建立的儲存貯體。
- S3 目標儲存貯體。請參閱託管於 Amazon S3 的靜態網站。請務必在與您要建立的管道AWS 區域相同的儲存貯體中建立。

> **Note**
>
> 此範例示範部署檔案到私有儲存貯體。請勿啟用用於網站託管的目標儲存貯體，或連接使儲存貯體公開的任何政策。

步驟 1：建立和上傳來源檔案到您的 S3 來源儲存貯體

在本節，您會建立並上傳來源檔案到管道用於來源階段的儲存貯體。本節提供建立下列來源檔案的說明：

- 一個`buildspec.yml`文件，用於 CodeBuild 構建項目。
- `index.ts` 檔案。

建立 buildspec.yml 檔案

- 使用下列內容建立名為 `buildspec.yml` 的檔案。這些建置命令會安裝 TypeScript 並使用 TypeScript編譯器將程式碼重寫`index.ts`為 JavaScript程式碼。

```
version: 0.2

phases:
  install:
    commands:
      - npm install -g typescript
  build:
    commands:
      - tsc index.ts
artifacts:
  files:
    - index.js
```

建立 index.ts 檔案

- 使用下列內容建立名為 `index.ts` 的檔案。

```
interface Greeting {
    message: string;
```

```
}

class HelloGreeting implements Greeting {
    message = "Hello!";
}

function greet(greeting: Greeting) {
    console.log(greeting.message);
}

let greeting = new HelloGreeting();

greet(greeting);
```

上傳檔案到您的 S3 來源儲存貯體

1. 在本機目錄中，您的檔案應該如下所示：

   ```
   buildspec.yml
   index.ts
   ```

   壓縮檔案，然後命名檔案為 `source.zip`。
2. 在 Amazon S3 主控台中，針對您的來源儲存貯體選擇「上傳」。選擇 Add files (新增檔案)，然後瀏覽您建立的 ZIP 檔案。
3. 選擇上傳。這些檔案是 Create Pipeline (建立管道) 精靈針對 CodePipeline 中的部署動作所建立的來源成品。您的檔案在您的儲存貯體中應該如下所示：

   ```
   source.zip
   ```

步驟 2：建立管道

在本節中，您可以採取下列動作建立管道：

- 具有 Amazon S3 動作的來源階段，其中來源成品是可下載應用程式的檔案。
- 具有 Amazon S3 部署動作的部署階段。

使用精靈建立管道

1. 請登入AWS Management Console並開啟 CodePipeline 主控台，網址為 http://console.aws.amazon.com/codesuite/codepipeline/home。
2. 在 Welcome (歡迎) 頁面、Getting started (入門) 頁面、或者 Pipelines (管道) 頁面上，選擇 Create pipeline (建立管道)。
3. 在 Step 1: Choose pipeline settings (步驟 1：選擇管道設定) 的 Pipeline name (管道名稱) 中，輸入 **MyS3DeployPipeline**。
4. 在 [服務角色] 中，選擇 [新增服務角色] CodePipeline 以允許在 IAM 中建立服務角色。
5. 將 Advanced settings (進階設定) 下的設定保留為預設值，然後選擇 Next (下一步)。
6. 在 Step 2: Add source stage (步驟 2：新增來源階段) 的 Source provider (來源提供者) 中，選擇 Amazon S3。在 Bucket (儲存貯體) 中，輸入您來源儲存貯體的名稱。在 S3 object key (S3 物件金鑰) 中，輸入您的來源 ZIP 檔案名稱。請確保您加入 .zip 副檔名。

   選擇下一步。
7. 在 Step 3: Add build stage (步驟 3：新增建置階段)：

   a. 在 Build provider (建置供應商) 中，選擇 CodeBuild。

   b. 選擇 Create build project (建立建置專案)。在 Create project (建立專案) 頁面：

   c. 在 Project name (專案名稱) 中，輸入此建置專案的名稱。

   d. 在 Environment (環境) 中，選擇 Managed image (受管映像)。針對 Operating system (作業系統)，選擇 Ubuntu。

   e. 針對 Runtime (執行時間)，選擇 Standard (標準)。針對 Runtime version (執行時間版本)，選擇 aws/codebuild/standard:1.0。

   f. 在 Image version (映像版本) 中，選擇 Always use the latest image for this runtime version (總是將最新的映像用於此執行時間版本)。

   g. 對於服務角色，請選擇您的 CodeBuild 服務角色，或建立一個服務角色。

   h. 對於 Build specifications (建置規格)，選擇 Use a buildspec file (使用 buildspec 檔案)。

   i. 選擇「繼續」 CodePipeline。如果已成功建立專案，則會顯示訊息。

   j. 選擇下一步。
8. 在 Step 4: Add deploy stage (步驟 4：新增部署階段) 中：

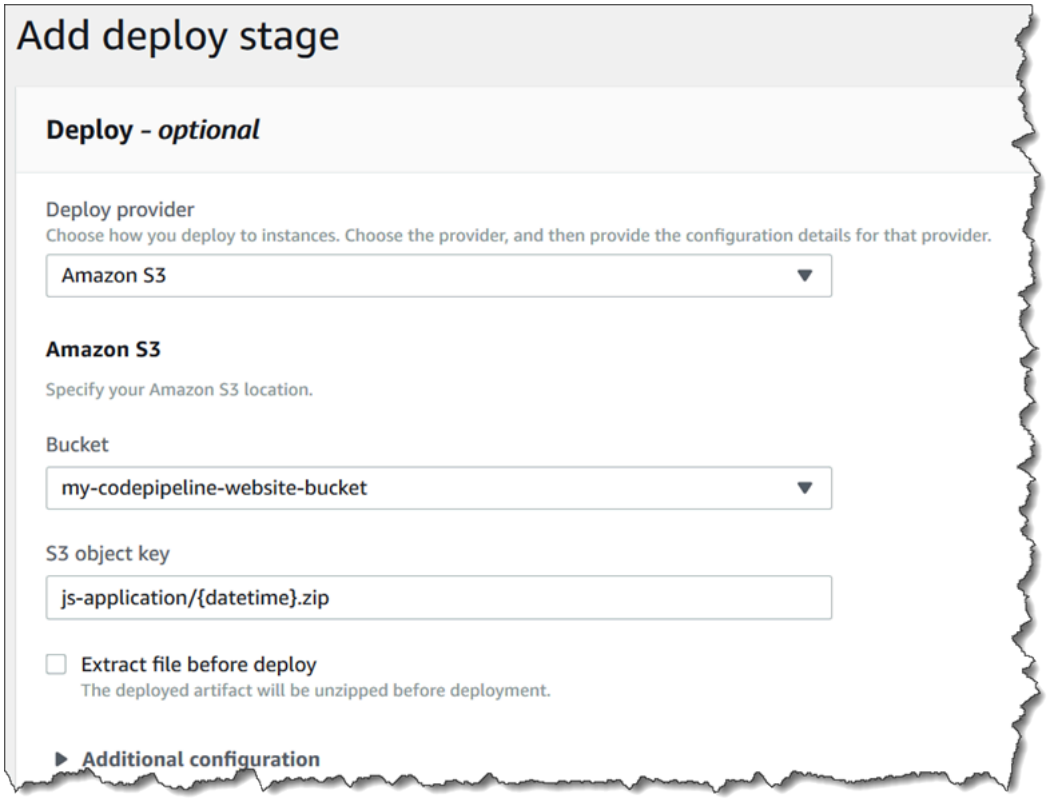
   a. 在Deploy provider (部署提供者)中，選擇 Amazon S3。

   b. 在 Bucket (儲存貯體) 中，輸入您 S3 目標儲存貯體的名稱。

c. 請確定已清除 Extract file before deploy (部署前解壓縮檔案)。

清除 Extract file before deploy (部署前解壓縮檔案) 時，則會顯示 S3 object key (S3 物件金鑰)。請輸入您要使用的路徑名稱：`js-application/{datetime}.zip`

這將在 Amazon S3 中創建一個文件`js-application`夾，該文件夾將文件提取到該文件夾。在此資料夾中，`{datetime}` 變數會在管道執行時在每個輸出檔案上建立時間戳記。



d. (選用) 在 Canned ACL (固定的 ACL) 中，您可以將一組預先定義的授與 (稱為固定的 ACL) 套用至上傳的成品。

e. (選用) 在 Cache control (快取控制) 中，輸入快取參數。您可以將此設為控制請求/回應的快取行為。如需有效值，請參閱 HTTP 操作的 `Cache-Control` 標頭欄位。

f. 選擇下一步。

9. 在 Step 5: Review (步驟 5：檢閱) 中，檢閱資訊，然後選擇 Create pipeline (建立管道)。

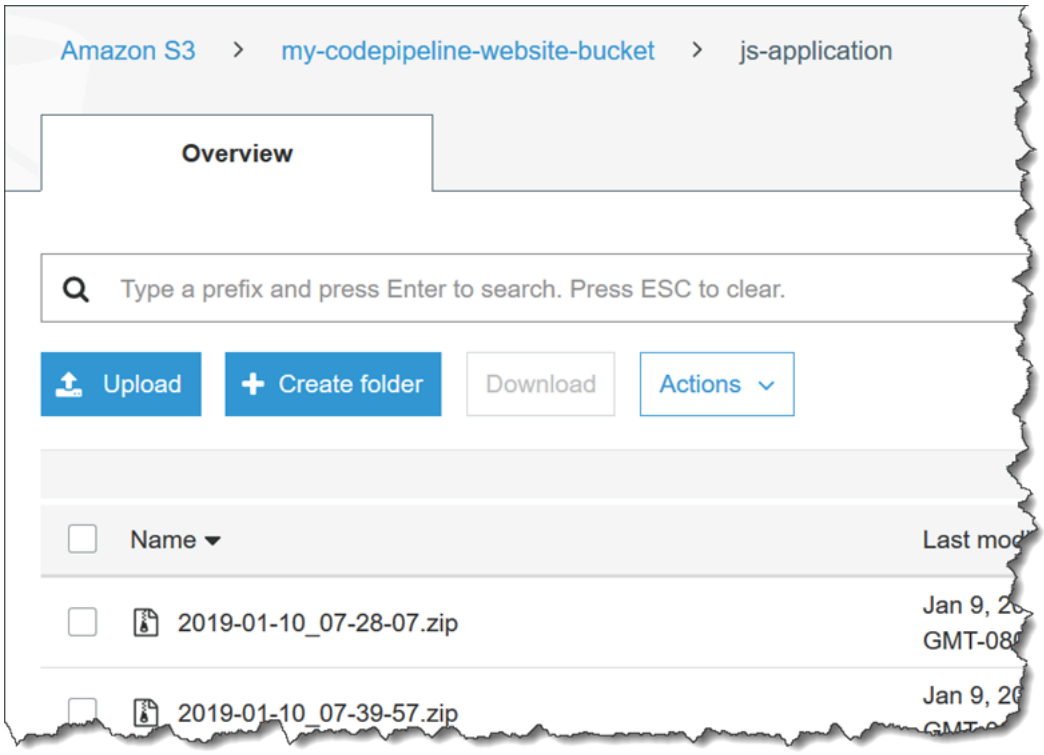
10. 管道成功執行後，請在 Amazon S3 主控台中檢視儲存貯體。驗證您部署的 ZIP 檔案在 `js-application` 資料夾下的目標儲存貯體中顯示。包含在 ZIP JavaScript 文件中的文件應該是`index.js`。`index.js` 檔案包含下列輸出：

```
var HelloGreeting = /** @class */ (function () {
    function HelloGreeting() {
```

```
        this.message = "Hello!";
    }
    return HelloGreeting;
}());
function greet(greeting) {
    console.log(greeting.message);
}
var greeting = new HelloGreeting();
greet(greeting);
```

步驟 3：變更任一來源檔案並驗證部署

變更您的來源檔案，然後將它們推送至您的來源儲存貯體。這會觸發您的管道執行。檢視您的目標儲存貯體，並驗證部署的輸出檔案可在 `js-application` 資料夾中使用，如下所示：



教學課程：建立將無伺服器應用程式發佈到 AWS Serverless Application Repository

您可以用 AWS CodePipeline 來持續將 AWS SAM 無伺服器應用程式交付給. AWS Serverless Application Repository

本教學課程說明如何建立和設定管道，以建置託管在中的無伺服器應用程式， GitHub 並將其 AWS Serverless Application Repository 自動發佈到。管線使用 GitHub 作為來源提供者和 CodeBuild 組建提供者。若要將無伺服器應用程式發佈到 AWS Serverless Application Repository，請部署應用程式 (從 AWS Serverless Application Repository)，並將該應用程式建立的 Lambda 函數關聯為管道中的叫用動作提供者。然後，您可以持續將應用程式更新傳遞至 AWS Serverless Application Repository，而無需撰寫任何程式碼。

> **Important**
>
> 您在此程序中新增至管線的許多動作都涉及建立管線之前需要建立的 AWS 資源。 AWS 來源動作的資源必須始終建立在您建立管道的相同 AWS 區域中。例如，如果您在美國東部 (俄亥俄) 區域建立管道，則您的 CodeCommit 存放庫必須位於美國東部 (俄亥俄) 區域。
> 您可以在建立管道時新增跨區域動作。 AWS 跨區域作業的資源必須位於您計劃執行作業的相同 AWS 區域中。如需詳細資訊，請參閱 在 CodePipeline 中新增跨區域動作。

開始之前

在此教學課程中，我們假設以下情況。

- 您熟悉 AWS Serverless Application Model (AWS SAM) 和 AWS Serverless Application Repository。
- 您有一個託管的無伺服器應 GitHub 用程式，您已 AWS Serverless Application Repository 使用 AWS SAM CLI 發佈到。若要將範例應用程式發佈到 AWS Serverless Application Repository，請參閱開發AWS Serverless Application Repository 人員指南中的快速入門:發佈應用程式。若要將您自己的應用程式發佈到 AWS Serverless Application Repository，請參閱開發AWS Serverless Application Model 人員指南中的使用 AWS SAM CLI 發佈應用程式。

步驟 1：建立 buildspec.yml 檔案

建立包含下列內容的`buildspec.yml`檔案，並將其新增至無伺服器應用程式的 GitHub 儲存庫。將 *`template.yml`* 取代為應用程式的 AWS SAM 範本，並以儲存封裝應用程式*#* S3 儲存貯體取代儲存貯體。

```
version: 0.2
phases:
  install:
    runtime-versions:
```

```
        python: 3.8
  build:
    commands:
      - sam package --template-file template.yml --s3-bucket bucketname --output-
template-file packaged-template.yml
artifacts:
  files:
    - packaged-template.yml
```

步驟 2：建立並設定管道

請遵循下列步驟，在您要發佈無伺服器應用程式的 AWS 區域 位置建立管道。

1. 請登入 AWS Management Console 並開啟 CodePipeline 主控台，網址為 https://console.aws.amazon.com/codepipeline/。
2. 如有必要，請切換至您要發佈無伺服器應用程式的 AWS 區域 位置。
3. 選擇 Create pipeline (建立管道)。在 Choose pipeline settings (選擇管道設定) 頁面的 Pipeline name (管道名稱) 中輸入管道的名稱。
4. 在管線類型中，針對本教學課程的目的選擇 V1。您也可以選擇 V2；但是請注意，管道類型在特性和價格上有所不同。如需詳細資訊，請參閱 管線類型。
5. 在 [服務角色] 中，選擇 [新增服務角色] CodePipeline 以允許在 IAM 中建立服務角色。
6. 將 Advanced settings (進階設定) 下的設定保留為預設值，然後選擇 Next (下一步)。
7. 在 [新增來源階段] 頁面的 [來源提供者] 中，選擇GitHub。
8. 在「連線」下，選擇現有的連線或建立新的連線。若要建立或管理來 GitHub 源動作的連線，請參閱GitHub 連接。
9. 在儲存庫中，選擇您的 GitHub 來源儲存庫。
10. 在「分支」中，選擇您的 GitHub 分支。
11. 保留來源動作的其餘預設值。選擇下一步。
12. 在 Add build stage (新增建置階段) 頁面，新增建置階段：

    a. 在 Build provider (建置供應商) 中，選擇 AWS CodeBuild。對於 Region (區域)，請使用管道區域。

    b. 選擇建立專案。

    c. 在 Project name (專案名稱) 中，輸入此建置專案的名稱。

    d. 在 Environment image (環境映像) 中，選擇 Managed image (受管映像)。針對 Operating system (作業系統)，選擇 Ubuntu。

e. 對於 Runtime (執行時間) 和 Runtime version (執行時間版本)，選擇無伺服器應用程式所需的執行時間和版本。

f. 對於 Service role (服務角色)，選擇 New service role (新服務角色)。

g. 對於 Build specifications (建置規格)，選擇 Use a buildspec file (使用 buildspec 檔案)。

h. 選擇繼續 CodePipeline。這將打開 CodePipeline 控制台並創建一個 CodeBuild 項目，該項目使用存儲庫`buildspec.yml`中的配置。組建專案會使用服務角色來管理 AWS 服務 權限。此步驟可能需要數分鐘。

i. 選擇下一步。

13. 在 Add deploy stage (新增部署階段) 頁面上，選擇 Skip deploy stage (跳過部署階段)，然後再次選擇 Skip (跳過) 來接受警告訊息。選擇下一步。

14. 選擇 Create pipeline (建立管道)。您應該會看到圖表，顯示該來源及建置階段。

15. 授與 CodeBuild 服務角色存取存放封裝應用程式之 S3 儲存貯體的權限。

a. 在新管道的 Build (建置) 階段中，選擇 CodeBuild。

b. 選擇 Build details (建置詳細資訊) 標籤。

c. 在環境中，選擇要開啟 IAM 主控台的 CodeBuild 服務角色。

d. 展開 `CodeBuildBasePolicy` 的選項，然後選擇 Edit policy (編輯政策)。

e. 選擇 JSON。

f. 新增政策陳述式與下列內容。該語句允許 CodeBuild 將對象放入存儲已打包應用程序的 S3 存儲桶中。以您的 S3 儲存貯體名稱取代 *`bucketname`*。

```
{
    "Effect": "Allow",
    "Resource": [
        "arn:aws:s3:::bucketname/*"
    ],
    "Action": [
        "s3:PutObject"
    ]
}
```

g. 選擇檢閱政策。

h. 選擇儲存變更。

步驟 3：部署發佈應用程式

請遵循下列步驟來部署包含執行發佈至 AWS Serverless Application Repository. 這個應用程序是 aws-serverless-codepipeline-serverlessrepo-發布。

> **Note**
>
> 您必須將應用程式部署到與管道 AWS 區域 相同的位置。

1. 移至應用程式頁面，並選擇 Deploy (部署)。
2. 選擇 I acknowledge that this app creates custom IAM roles (我認可此應用程式建立自訂的 IAM 角色)。
3. 選擇部署。
4. 選擇「檢視 AWS CloudFormation 堆疊」以開啟主 AWS CloudFormation 控台。
5. 展開 Resources (資源) 區段。你看 ServerlessRepoPublish，這是類型的AWS::Lambda::Function。請記下這個資源的實體 ID，以用於下一個步驟。您會在 CodePipeline 中建立新的發佈動作時用到此實體 ID。

步驟 4：建立發佈動作

依照以下步驟在管道中建立發佈動作。

1. 請在以下位置開啟 CodePipeline 主控台。 https://console.aws.amazon.com/codepipeline/
2. 在左側導覽區段中，選擇您欲編輯的管道。
3. 選擇編輯。
4. 在目前管道的最後階段之後，選擇 + Add stage (新增階段)。在 Stage name (階段名稱) 中輸入名稱，例如 **`Publish`**，然後選擇 Add stage (新增階段)。
5. 在新階段中，選擇 + Add action group (+ 新增動作群組)。
6. 輸入動作名稱。從 Action provider (動作供應商) 的 Invoke (呼叫) 中，選擇 AWS Lambda。
7. 從輸入人工因素中，選擇BuildArtifact。
8. 在函數名稱中，選擇您在上一個步驟中記下的 Lambda 函數的實體 ID。
9. 為動作選擇 Save (儲存)。
10. 為階段選擇 Done (完成)。
11. 在右上角，選擇 Save (儲存)。

12. 若要驗證您的管道，請在中對應用程式進行變更 GitHub。例如，在範 AWS SAM 本檔案的`Metadata`區段中變更應用程式的說明。提交更改並將其推送到您的 GitHub 分支。這會觸發您的管道執行。當管道完成時，在 AWS Serverless Application Repository 中檢查您的應用程式是否已就您的變更進行更新。

教學課程：使用變數搭配 Lambda 呼叫動作

Lambda 呼叫動作可以使用來自另一個動作的變數作為其輸入的一部分，並與其輸出一起傳回新變數。如需 CodePipeline 中動作變數的相關資訊，請參閱 Variables。

在本教學結束時，您將擁有：

- Lambda 呼叫動作，此動作可以：
  - 使用`CommitId`變數來自 CodeCommit 源動作
  - 輸出三個新變數：`dateTime`、`testRunId` 和 `region`
- 手動核準動作，此動作會透過 Lambda 呼叫動作使用新變數，以提供測試 URL 和測試執行 ID
- 以新動作更新的管道

主題

先決條件

開始之前，您必須準備好以下項目：

- 您可以使用管道和 CodeCommit 來源在教學：建立範本管道 (CodeCommit 儲存庫)。
- 編輯您現有的管道，讓 CodeCommit 來源動作具備命名空間。將命名空間指派 `SourceVariables` 給動作。

步驟 1：建立 Lambda 函數

請使用下列步驟來建立 Lambda 函數和 Lambda 執行角色。您可以在建立 Lambda 函數後，將 Lambda 動作新增到您的管道。

建立 Lambda 函數和執行角色

1. 請登入 AWS Management Console，並開啟位於 https://console.aws.amazon.com/lambda/ 的 AWS Lambda 主控台。
2. 選擇 Create function (建立函數)。將 Author from scratch (從頭開始編寫) 維持在選取狀態。
3. 在 Function name (函數名稱) 中，輸入您函數的名稱 (例如 **myInvokeFunction**)。在 Runtime (執行時間) 中，將預設選項維持在選取狀態。
4. 展開 Choose or create an execution role (選擇或建立執行角色)。選擇 Create a new role with basic Lambda permissions (建立具備基本 Lambda 許可的新角色)。
5. 選擇 Create function (建立函數)。
6. 若要透過另一個動作使用變數，則必須將其傳遞給 Lambda 呼叫動作組態中的 `UserParameters`。您將在稍後教學中在我們的管道中設定動作，但您將新增程式碼，此程式碼會假設此變數將傳遞。

   ```
   const commitId =
   event["CodePipeline.job"].data.actionConfiguration.configuration.UserParameters;
   ```

   若要產生新變數，請將輸入上名為 `outputVariables` 的屬性設定為 `putJobSuccessResult`。請注意，您無法產生變數作為 `putJobFailureResult` 的一部分。

   ```
   const successInput = {
     jobId: jobId,
     outputVariables: {
       testRunId: Math.floor(Math.random() * 1000).toString(),
       dateTime: Date(Date.now()).toString(),
       region: lambdaRegion
     }
   };
   ```

   在您的新函數中，將 Edit code inline (編輯程式碼內嵌) 維持在選取狀態，並將以下範例程式碼貼到 `index.js` 的下方。

   ```
   var AWS = require('aws-sdk');

   exports.handler = function(event, context) {
       var codepipeline = new AWS.CodePipeline();

       // Retrieve the Job ID from the Lambda action
   ```

```
    var jobId = event["CodePipeline.job"].id;

    // Retrieve the value of UserParameters from the Lambda action configuration in
CodePipeline,
    // in this case it is the Commit ID of the latest change of the pipeline.
    var params =
event["CodePipeline.job"].data.actionConfiguration.configuration.UserParameters;

    // The region from where the lambda function is being executed.
    var lambdaRegion = process.env.AWS_REGION;

    // Notify CodePipeline of a successful job
    var putJobSuccess = function(message) {
        var params = {
            jobId: jobId,
            outputVariables: {
                testRunId: Math.floor(Math.random() * 1000).toString(),
                dateTime: Date(Date.now()).toString(),
                region: lambdaRegion
            }
        };
        codepipeline.putJobSuccessResult(params, function(err, data) {
            if(err) {
                context.fail(err);
            } else {
                context.succeed(message);
            }
        });
    };

    // Notify CodePipeline of a failed job
    var putJobFailure = function(message) {
        var params = {
            jobId: jobId,
            failureDetails: {
                message: JSON.stringify(message),
                type: 'JobFailed',
                externalExecutionId: context.invokeid
            }
        };
        codepipeline.putJobFailureResult(params, function(err, data) {
            context.fail(message);
        });
    };
```

```
    var sendResult = function() {
        try {
            console.log("Testing commit - " + params);

            // Your tests here

            // Succeed the job
            putJobSuccess("Tests passed.");
        } catch (ex) {
            // If any of the assertions failed then fail the job
            putJobFailure(ex);
        }
    };

    sendResult();
};
```

7. 選擇 Save (儲存)。
8. 複製畫面頂端的 Amazon Resource Name (ARN)。
9. 最後一個步驟，請在AWS Identity and Access Management(IAM) 控制台https://console.aws.amazon.com/iam/。修改 Lambda 執行角色以新增以下政策：遠AWSCodePipelineCustomActionAccess。如需建立 Lambda 執行角色或修改角色政策的步驟，請參閱 步驟 2：建立 Lambda 函數。

步驟 2：將 Lambda 呼叫動作和手動核準動作新增到您的管道

在此步驟中，您會將 Lambda 呼叫動作新增到您的管道。您會新增名為 Test (測試) 的動作，做為階段的一部分。動作類型是呼叫動作。您接著會在呼叫動作之後新增手動核准動作。

將 Lambda 動作和手動核準動作新增到管道

1. 前往 https://console.aws.amazon.com/codepipeline/ 開啟 CodePipeline 主控台。

   所有與您 AWS 帳戶相關聯的管道名稱都會顯示。選擇您要新增動作的管道。
2. 將 Lambda 測試動作新增到您的管道。

   a. 若要編輯管道，請選擇 Edit (編輯)。在現有管道中於來源動作之後新增階段。輸入階段的名稱，例如 **Test**。

b. 在新階段中，選擇新增動作的圖示。在 Action name (動作名稱) 中，輸入呼叫動作的名稱，例如 **Test_Commit**。

c. In動作供應商，選擇AWS Lambda。

d. 在 Input artifacts (輸入成品) 中，選擇您來源動作輸出成品的名稱，例如 SourceArtifact。

e. In函數名稱中，選擇您建立的 Lambda 函數的名稱。

f. In使用者參數中，輸入 CodeCommit ID 的變數語法。這會建立輸出變數，解析至每次管道執行時要檢閱和核准的遞交。

```
#{SourceVariables.CommitId}
```

g. 在 Variable namespace (變數命名空間) 中，新增命名空間名稱，例如 **TestVariables**。

h. 選擇 Done (完成)。

3. 將手動核准動作新增到您的管道。

a. 在您的管道仍處於編輯模式中時，於呼叫動作之後新增階段。輸入階段的名稱，例如 **Approval**。

b. 在新階段中，選擇新增動作的圖示。在 Action name (動作名稱) 中，輸入核准動作的名稱，例如 **Change_Approval**。

c. 在 Action provider (動作提供者) 中，選擇 Manual approval (手動核准)。

d. 在 URL for review (檢閱 URL) 中，透過新增 region 變數和 CommitId 變數的變數語法來建構 URL。請確定您使用的是指派給提供輸出變數動作的命名空間。

在此範例中，擁有 CodeCommit 動作變數語法的 URL 具備默認命名空間SourceVariables。Lambda 區域輸出變數具備 TestVariables 命名空間。URL 看起來如下。

```
https://#{TestVariables.region}.console.aws.amazon.com/codesuite/codecommit/
repositories/MyDemoRepo/commit/#{SourceVariables.CommitId}
```

在 Comments (註解) 中，透過新增 testRunId 變數的變數語法來建構核准訊息文字。針對此範例，擁有 Lambda testRunId 輸出變數變數語法的 URL 具備 TestVariables 命名空間。輸入以下訊息。

```
Make sure to review the code before approving this action. Test Run ID:
 #{TestVariables.testRunId}
```

4. 選擇 Done (完成) 來關閉動作的編輯畫面，然後選擇 Done (完成) 來關閉階段的編輯畫面。如要儲存管道，請選擇 Done (完成)。已完成的管道現在包含結構，其中包含來源、測試、核准和部署階段。

   選擇 Release change (發行變更) 來透過管道結構執行最新的變更。

5. 管道到達手動核准階段時，請選擇 Review (檢閱)。已解析的變數會做為遞交 ID 的 URL 出現。您的核准者可以選擇 URL 來檢閱遞交。

6. 管道成功執行後，您也可以在 action execution history (動作執行歷史記錄) 頁面上檢視變數值。

教學課程：使用AWS Step Functions叫用管道中的動作

您可以使用 AWS Step Functions 來建立和設定狀態機器。本教學示範如何將叫用動作新增至管道，從您的管道啟動狀態機執行。

在本教學中，您將執行下列操作：

- 在 AWS Step Functions 中建立標準狀態機器。
- 直接輸入狀態機器輸入 JSON。您也可以將狀態機器輸入文件上傳至 Amazon Simple Storage Service (Amazon S3) 儲存貯體。
- 透過新增狀態機器動作來更新您的管道。

> **Note**
>
> 此功能不在亞太區域 (香港) 和歐洲 (米蘭) 區域提供。若要引用其他可行的動作，請參產品與服務整合 CodePipeline。

主題

必要條件：建立或選擇簡易管道

在本教學中，您會將叫用動作新增至現有管道。您可以使用在 教學：建立簡易管道 (S3 儲存貯體) 或 教學：建立範本管道 (CodeCommit 儲存庫) 中建立的管道。

您可以使用具有來源動作和至少有兩階段結構的現有管道，但在此範例中不使用來源成品。

> **Note**
>
> 您可能需要額外許可來更新管道所使用的服務角色，才能執行此動作。若要執行此作業，請開啟AWS Identity and Access Management(IAM) 主控台，尋找角色，然後將許可新增至角色的政策。如需詳細資訊，請參閱 將許可新增至 CodePipeline 服務角色。

步驟 1：建立範例狀態機器

在步進函數控台中，使用`HelloWorld`範例範本。如需說明，請參閱「」建立狀態機器中的AWS Step Functions開發人員指南。

步驟 2：將 Step Functions 叫用動作新增至您的管道

將 Step Functions 叫用動作新增至您的管道，如下所示：

1. 登入AWS Management Console並開啟 CodePipeline 主控台，請開啟http://console.aws.amazon.com/codesuite/codepipeline/home。

   所有與您的 AWS 帳戶相關的管道名稱都會顯示。
2. 在 Name (名稱) 中，選擇您想編輯的管道名稱。這會開啟管道的詳細檢視，包含管道中各階段的每項動作之每項狀態。
3. 在管道詳細資訊頁面上，選擇 Edit (編輯)。
4. 在簡易管道的第二階段，選擇 Edit stage (編輯階段)。選擇 Delete (刪除)。這會刪除第二階段，因為您不再需要它。
5. 在圖表的底部，選擇 + Add stage (+ 新增階段)。
6. 在 Stage name (階段名稱) 中，輸入階段的名稱，例如 **`Invoke`**，然後選擇 Add stage (新增階段)。
7. 選擇 + Add action group (+ 新增動作群組)。
8. 在 Action name (動作名稱) 中，輸入名稱，例如 **`Invoke`**。

9. In動作供應商，選擇AWSStep Functions。允許 Region (區域) 預設為管道區域。
10. 在 Input artifacts (輸入成品) 中，選擇 `SourceArtifact`。
11. 在 State machine ARN (狀態機器 ARN) 中，為您先前建立的狀態機器選擇 Amazon Resource Name (ARN)。
12. (選用) 在 Execution name prefix (執行名稱前綴) 中，輸入要新增至狀態機器執行 ID 的前綴。
13. 在 Input type (輸入類型) 中，選擇 Literal (常值)。
14. 在 Input (輸入) 中，輸入 `HelloWorld` 範例狀態機器預期的輸入 JSON。

    > **Note**
    >
    > 狀態機器執行的輸入與 CodePipeline 中描述動作的輸入工件所用的術語不同。

    在本範例中，輸入下列 JSON：

    ```
    {"IsHelloWorldExample": true}
    ```

15. 選擇 Done (完成)。
16. 在您正在編輯的階段上，選擇 Done (完成)。在 AWS CodePipeline 窗格中，選擇 Save (儲存)，然後在警告訊息中選擇 Save (儲存)。
17. 若要提交您的變更並啟動管道執行，請選擇 Release change (發行變更)，然後選擇 Release (發行)。
18. 在完成的管道上，選擇AWSStep Functions在您的調用操作中。在 AWS Step Functions 主控台中，檢視您的狀態機器執行 ID。該 ID 會顯示您的狀態機器名稱 `HelloWorld` 和帶有前綴 `my-prefix` 的狀態機器執行 ID。

    ```
    arn:aws:states:us-west-2:account-ID:execution:HelloWorld:my-
    prefix-0d9a0900-3609-4ebc-925e-83d9618fcca1
    ```

教學：建立將 AWS AppConfig 做為部署提供者使用的管道

在本教學課程中，您會設定管線，在部署階段中使用AWS AppConfig 做為部署動作提供者，持續提供組態檔案。

主題

必要條件

開始之前，您必須完成以下操作：

- 此範例將 S3 來源用於您的管道。建立或使用已啟用版本控制的 Amazon S3 儲存貯體。遵循步驟 1：為您的應用程式建立 S3 儲存貯體中的說明，建立 S3 儲存貯體。

步驟 1：建立資AWS AppConfig源

在本節中，您將建立下列資源：

- 中的應用程式AWS AppConfig 是為客戶提供功能的邏輯程式碼單元。
- 中的環境AWS AppConfig 是 AppConfig 目標的邏輯部署群組，例如測試版或生產環境中的應用程式。
- 組態設定檔是會影響應用程式行為的設定集合。組態設定檔可AWS AppConfig 讓您在其儲存位置存取您的組態。
- (選擇性) 中的部署策略會AWS AppConfig 定義組態部署的行為，例如在部署期間的任何指定時間，用戶端應接收新部署組態的百分比。

若要建立應用程式、環境、組態設定檔和部署策略

1. 登入 AWS Management Console。
2. 使用下列主題中的步驟，在中建立您的資源AWS AppConfig。
    - 建立應用程式。
    - 建立環境。
    - 建立組AWS CodePipeline態設定描述檔。
    - (選擇性) 選擇預先定義的部署策略或建立自己的部署策略。

步驟 2：將檔案上傳到 S3 來源儲存貯體

在本節中，建立一個或多個組態檔案。然後壓縮源文件並將其推送到管道用於源階段的存儲桶。

建立組態檔案的步驟

1. 為每個區域`configuration.json`中的每個配置創建一個文件。包括以下內容：

```
Hello World!
```

2. 請使用下列步驟壓縮並上傳您的組態檔案。

壓縮和上傳來源檔案

1. 使用您的檔案建立 .zip 檔案，並命名 .zip 檔案。`configuration-files.zip`例如，您的 .zip 檔案可以使用下列結構：

```
.
### appconfig-configurations
    ### MyConfigurations
        ### us-east-1
        #   ### configuration.json
        ### us-west-2
            ### configuration.json
```

2. 在儲存貯體的 Amazon S3 主控台中，選擇「上傳」，然後按照指示上傳 .zip 檔案。

步驟 3：建立管道

在本節中，您可以採取下列動作建立管道：

- 具有 Amazon S3 動作的來源階段，其中來源成品是您組態的檔案。
- 具有部署動作的 AppConfig 部署階段。

使用精靈建立管道

1. 請登入AWS Management Console並開啟 CodePipeline 主控台，網址為 http://console.aws.amazon.com/codesuite/codepipeline/home。

2. 在 Welcome (歡迎) 頁面、Getting started (入門) 頁面、或者 Pipelines (管道) 頁面上，選擇 Create pipeline (建立管道)。
3. 在 Step 1: Choose pipeline settings (步驟 1：選擇管道設定) 的 Pipeline name (管道名稱) 中，輸入 **`MyAppConfigPipeline`**。
4. 在管線類型中，針對本教學課程的目的選擇 V1。您也可以選擇 V2；但是請注意，管道類型在特性和價格上有所不同。如需詳細資訊，請參閱管線類型。
5. 在 [服務角色] 中，選擇 [新增服務角色] CodePipeline 以允許在 IAM 中建立服務角色。
6. 將 Advanced settings (進階設定) 下的設定保留為預設值，然後選擇 Next (下一步)。
7. 在 Step 2: Add source stage (步驟 2：新增來源階段) 的 Source provider (來源提供者) 中，選擇 Amazon S3。在儲存貯體中，選擇 S3 來源儲存貯體的名稱。

   在 S3 物件金鑰中，輸入 .zip 檔案的名稱：`configuration-files.zip`。

   選擇下一步。
8. 在 Step 3: Add build stage (步驟 3：新增建置階段) 中，選擇 Skip build stage (跳過建置階段)，然後再次選擇 Skip (跳過) 來接受警告訊息。

   選擇下一步。
9. 在 Step 4: Add deploy stage (步驟 4：新增部署階段) 中：
   a. 在部署提供者中，選擇AWS AppConfig。
   b. 在應用程式中，選擇您在其中建立的應用程式名稱AWS AppConfig。此欄位會顯示應用程式的 ID。
   c. 在環境中，選擇您在其中建立的環境名稱AWS AppConfig。此欄位會顯示您環境的 ID。
   d. 在「組態設定檔」中，選擇您在中建立的組態設定檔名稱AWS AppConfig。此欄位會顯示您的設定描述檔的 ID。
   e. 在部署策略中，選擇部署策略的名稱。這可以是您在中建立的部署策略， AppConfig 也可以是您從中預先定義的部署策略中選擇的部署策略 AppConfig。此欄位會顯示部署策略的 ID。
   f. 在輸入人工因素組態路徑中，輸入檔案路徑。請確定您的輸入成品組態路徑與 S3 儲存貯體 .zip 檔案中的目錄結構相符。在此範例中，輸入下列檔案路徑：`appconfig-configurations/MyConfigurations/us-west-2/configuration.json`。
   g. 選擇下一步。
10. 在 Step 5: Review (步驟 5：檢閱) 中，檢閱資訊，然後選擇 Create pipeline (建立管道)。

步驟 4：對任何源文件進行更改並驗證部署

變更來源檔案，然後將變更上傳至儲存貯體。這會觸發您的管道執行。檢視版本以確認您的組態可供使用。

教學課程：搭 GitHub 配管線來源使用完整複製

您可以在中為 GitHub 來源動作選擇完整複製選項 CodePipeline。使用此選項可在管線建置動作中執行 Git 中繼資料的 CodeBuild 命令。

在本教學課程中，您將建立連線到 GitHub 儲存庫的管線、使用來源資料的完整複製選項，以及執行複製儲存庫並執行 Git 命令的 CodeBuild 組建。

> **Note**
>
> 亞太區域 (香港)、非洲 (開普敦)、中東 (巴林)、歐洲 (蘇黎世) 或 AWS GovCloud (美國西部) 地區不提供此功能。若要參考其他可用動作，請參閱產品與服務整合 CodePipeline。如需在歐洲 (米蘭) 區域進行此動作的考量，請參閱中的附註CodeStarSourceConnection 適用於 Bitbucket 雲端 GitHub、 GitHub 企業伺服器、 GitLab .com 和 GitLab 自我管理動作。

主題

必要條件

開始之前，您必須執行以下作業：

- 使用您的 GitHub 帳戶創建一個 GitHub 存儲庫。
- 準備好您的 GitHub 憑據。當您使用設AWS Management Console定連線時，系統會要求您使用 GitHub 認證登入。

步驟 1：建立讀我檔案

建立 GitHub 儲存庫之後，請使用下列步驟新增 README 檔案。

1. 登錄到您的 GitHub 存儲庫並選擇您的存儲庫。
2. 若要建立新檔案，請選擇「新增檔案 > 建立新檔案」。命名文件`README.md`。文件並添加以下文本。

   ```
   This is a GitHub repository!
   ```

3. 選擇 Commit changes (遞交變更)。

   確定 `README.md` 檔案位於儲存庫的根層級。

第 2 步：創建管道並構建項目

在本節中，您可以採取下列動作建立管道：

- 與 GitHub 存放庫和動作連線的來源階段。
- 使用 AWS CodeBuild 建置動作的建置階段。

使用精靈建立管道

1. 請在以下位置登入 CodePipeline 主控台：https://console.aws.amazon.com/codepipeline/。
2. 在 Welcome (歡迎) 頁面、Getting started (入門) 頁面、或者 Pipelines (管道) 頁面上，選擇 Create pipeline (建立管道)。
3. 在 Step 1: Choose pipeline settings (步驟 1：選擇管道設定) 的 Pipeline name (管道名稱) 中，輸入 **`MyGitHubPipeline`**。
4. 在管線類型中，針對本教學課程的目的選擇 V1。您也可以選擇 V2；但是請注意，管道類型在特性和價格上有所不同。如需詳細資訊，請參閱管線類型。
5. 在 Service role (服務角色) 中，選擇 New service role (新服務角色)。

> **Note**
>
> 如果您改為選擇使用現有的 CodePipeline 服務角色，請確定已將 `codestar-connections:UseConnection` IAM 權限新增至服務角色政策。如需 CodePipeline 服務角色的指示，請參閱將權限新增至 CodePipeline 服務角色。

6. 在進階設定底下，請保留預設值。在Artifact store (成品存放區) 中，針對您為管道所選取區域中的管道，選擇 Default location (預設位置)，即可使用預設成品存放區 (例如指定為預設值的 Amazon S3 成品儲存貯體)。

> **Note**
>
> 這不是原始碼的來源儲存貯體。這是管道的成品存放區。每個管道都需要有個別成品存放區，例如 S3 儲存貯體。

選擇 Next (下一步)。

7. 在 Step 2: Add source stage (步驟 2：新增來源階段) 頁面上，新增來源階段：

   a. 在來源提供者中，選擇 GitHub (版本 2)。

   b. 在「連線」下，選擇現有的連線或建立新的連線。若要建立或管理來 GitHub 源動作的連線，請參閱GitHub 連接。

   c. 在 Repository name (儲存庫名稱) 中，選擇 GitHub 儲存庫的名稱。

   d. 在「分支名稱」中，選擇您要使用的儲存庫分支。

   e. 請確認已選取在原始程式碼變更時啟動管道選項。

   f. 在 [輸出成品格式] 下，選擇 [完整複製] 以啟用來源儲存庫的 Git 複製選項。只有提供的動作 CodeBuild 可以使用 Git 複製選項。您將步驟 3：更新 CodeBuild 服務角色原則以使用連線在本教學課程中使用來更新 CodeBuild 專案服務角色的權限，以使用此選項。

選擇 Next (下一步)。

8. 在 Add build stage (新增建置階段) 中，新增建置階段：

   a. 在 Build provider (建置供應商) 中，選擇 AWS CodeBuild。允許 Region (區域) 預設為管道區域。

   b. 選擇建立專案。

c. 在 Project name (專案名稱) 中，輸入此建置專案的名稱。

d. 在 Environment image (環境映像) 中，選擇 Managed image (受管映像)。針對 Operating system (作業系統)，選擇 Ubuntu。

e. 針對 Runtime (執行時間)，選擇 Standard (標準)。針對映像，選擇 aws/codebuild/standard:5.0。

f. 對於 Service role (服務角色)，選擇 New service role (新服務角色)。

> **Note**
>
> 記下 CodeBuild 服務角色的名稱。在本教學課程的最後一個步驟中，您將需要角色名稱。

g. 在 BuildSpec 底下，針對 Build specifications (建置規格) 選擇 Insert build commands (插入建置命令)。選擇切換到編輯器，然後將以下內容粘貼到構建命令下。

> **Note**
>
> 在構建規範的env部分中，確保啟用了 git 命令的憑據幫助程序，如本示例所示。

```
version: 0.2

env:
  git-credential-helper: yes
phases:
  install:
    #If you use the Ubuntu standard image 2.0 or later, you must specify
 runtime-versions.
    #If you specify runtime-versions and use an image other than Ubuntu
 standard image 2.0, the build fails.
    runtime-versions:
      nodejs: 12
      # name: version
    #commands:
      # - command
      # - command
  pre_build:
    commands:
      - ls -lt
```

```
      - cat README.md
  build:
    commands:
      - git log | head -100
      - git status
      - ls
      - git archive --format=zip HEAD > application.zip
  #post_build:
    #commands:
      # - command
      # - command
artifacts:
  files:
    - application.zip
    # - location
  #name: $(date +%Y-%m-%d)
  #discard-paths: yes
  #base-directory: location
#cache:
  #paths:
    # - paths
```

h. 選擇繼續 CodePipeline。這將返回到 CodePipeline控制台並創建一個使用構建命令進行配置的 CodeBuild 項目。組建專案會使用服務角色來管理AWS 服務權限。此步驟可能需要數分鐘。

i. 選擇下一步。

9. 在 Step 4: Add deploy stage (步驟 4：新增部署階段) 頁面上，選擇 Skip deploy stage (跳過部署階段)，然後再次選擇 Skip (跳過) 來接受警告訊息。選擇下一步。

10. 在 Step 5: Review (步驟 5：檢閱) 上，選擇 Create pipeline (建立管道)。

步驟 3：更新 CodeBuild 服務角色原則以使用連線

初始管線執行會失敗，因為 CodeBuild 服務角色必須更新為使用連線的權限。將 `codestar-connections:UseConnection` IAM 權限新增至您的服務角色政策。如需在 IAM 主控台中更新政策的指示，請參閱新增連線至 Bitbucket GitHub、 GitHub 企業伺服器或 GitLab .com 的 CodeBuild GitClone 權限。

第 4 步：在構建輸出中查看存儲庫命令

1. 成功更新服務角色後，請在失敗的 CodeBuild 階段中選擇「重試」。

2. 管道成功執行之後，在成功的建置階段中，選擇 [檢視詳細資料]。

   在詳細資料頁面上，選擇記錄檔索引標籤。檢視組 CodeBuild 建輸出。這些指令會輸出所輸入變數的值。

   這些指令會輸出README.md檔案內容、列出目錄中的檔案、複製存放庫、檢視記錄，以及將存放庫封存為 ZIP 檔案。

教學課程：搭 CodeCommit 配管線來源使用完整複製

您可以在中為 CodeCommit 來源動作選擇完整複製選項 CodePipeline。使用此選項可 CodeBuild 允許在管線建置動作中存取 Git 中繼資料。

在本教學課程中，您會建立存取儲存 CodeCommit 庫的管線、使用來源資料的完整複製選項，以及執行複製儲存庫並針對存放庫執行 Git 命令的 CodeBuild 組建。

> **Note**
>
> CodeBuild 動作是唯一支援使用 Git 複製選項提供的 Git 中繼資料的下游動作。此外，雖然您的管道可以包含跨帳戶動作，但 CodeCommit動作和 CodeBuild 動作必須位於相同帳戶中，完整複製選項才能成功。

主題

必要條件

在開始之前，您必須在與管道相同的AWS帳戶和區域中建立 CodeCommit 儲存庫。

步驟 1：創建一個自述文件

請使用下列步驟將 README 檔案新增至來源儲存庫。README 檔案提供了一個範例來源檔案，供下 CodeBuild 游動作讀取。

若要加入讀我檔案

1. 登錄到您的存儲庫並選擇您的存儲庫。
2. 若要建立新檔案，請選擇「新增檔案 > 建立檔案」。命名文件`README.md`。文件並添加以下文本。

   ```
   This is a CodeCommit repository!
   ```

3. 選擇 Commit changes (遞交變更)。

   確定 `README.md` 檔案位於儲存庫的根層級。

第 2 步：創建管道並構建項目

在本節中，您可以採取下列動作建立管道：

- 具有來源動作的 CodeCommit 來源階段。
- 使用 AWS CodeBuild 建置動作的建置階段。

使用精靈建立管道

1. 請在以下位置登入 CodePipeline 主控台：https://console.aws.amazon.com/codepipeline/。
2. 在 Welcome (歡迎) 頁面、Getting started (入門) 頁面、或者 Pipelines (管道) 頁面上，選擇 Create pipeline (建立管道)。
3. 在 Step 1: Choose pipeline settings (步驟 1：選擇管道設定) 的 Pipeline name (管道名稱) 中，輸入 **`MyCodeCommitPipeline`**。
4. 在管線類型中，針對本教學課程的目的選擇 V1。您也可以選擇 V2；但是請注意，管道類型在特性和價格上有所不同。如需詳細資訊，請參閱管線類型。
5. 在 Service role (服務角色) 中，執行下列其中一項作業：
   - 選擇 Existing service role (現有服務角色)。
   - 選擇您現有的 CodePipeline 服務角色。此角色必須具有服務角色政策的 `codecommit:GetRepository` IAM 許可。請參閱將權限新增至 CodePipeline 服務角色。

6. 在進階設定底下，請保留預設值。選擇下一步。
7. 在「步驟 2：新增來源階段」頁面上，執行下列動作：

   a. 在 Source provider (來源提供者) 中選擇 CodeCommit。

   b. 在存放庫名稱中，選擇存放庫的名稱。

   c. 在分支名稱中，選擇您的分支名稱。

   d. 請確認已選取在原始程式碼變更時啟動管道選項。

   e. 在 [輸出成品格式] 下，選擇 [完整複製] 以啟用來源儲存庫的 Git 複製選項。只有提供的動作 CodeBuild 可以使用 Git 複製選項。

   選擇下一步。
8. 在 [新增組建階段] 中，執行下列動作：

   a. 在 Build provider (建置供應商) 中，選擇 AWS CodeBuild。允許 Region (區域) 預設為管道區域。

   b. 選擇建立專案。

   c. 在 Project name (專案名稱) 中，輸入此建置專案的名稱。

   d. 在 Environment image (環境映像) 中，選擇 Managed image (受管映像)。針對 Operating system (作業系統)，選擇 Ubuntu。

   e. 針對 Runtime (執行時間)，選擇 Standard (標準)。針對映像，選擇 aws/codebuild/standard:5.0。

   f. 對於 Service role (服務角色)，選擇 New service role (新服務角色)。

   > **Note**
   >
   > 記下 CodeBuild 服務角色的名稱。在本教學課程的最後一個步驟中，您將需要角色名稱。

   g. 在 BuildSpec 底下，針對 Build specifications (建置規格) 選擇 Insert build commands (插入建置命令)。選擇切換到編輯器，然後在構建命令下粘貼以下代碼。

```
version: 0.2

env:
  git-credential-helper: yes
phases:
```

```
  install:
    #If you use the Ubuntu standard image 2.0 or later, you must specify
 runtime-versions.
    #If you specify runtime-versions and use an image other than Ubuntu
 standard image 2.0, the build fails.
    runtime-versions:
      nodejs: 12
      # name: version
    #commands:
      # - command
      # - command
  pre_build:
    commands:
      - ls -lt
      - cat README.md
  build:
    commands:
      - git log | head -100
      - git status
      - ls
      - git describe --all
  #post_build:
    #commands:
      # - command
      # - command
#artifacts:
  #files:
    # - location
  #name: $(date +%Y-%m-%d)
  #discard-paths: yes
  #base-directory: location
#cache:
  #paths:
    # - paths
```

h. 選擇「繼續」 CodePipeline。這將返回到 CodePipeline 控制台並創建一個使用構建命令進行配置的 CodeBuild 項目。組建專案會使用服務角色來管理AWS 服務權限。此步驟可能需要數分鐘。

i. 選擇下一步。

9. 在 Step 4: Add deploy stage (步驟 4：新增部署階段) 頁面上，選擇 Skip deploy stage (跳過部署階段)，然後再次選擇 Skip (跳過) 來接受警告訊息。選擇下一步。

10. 在 Step 5: Review (步驟 5：檢閱) 上，選擇 Create pipeline (建立管道)。

步驟 3：更新 CodeBuild 服務角色原則以複製存放庫

初始管線執行會失敗，因為您需要使用從存放庫中提取的權限更新 CodeBuild 服務角色。

將 `codecommit:GitPull` IAM 權限新增至您的服務角色政策。如需在 IAM 主控台中更新政策的指示，請參閱新增 CodeCommit來源動作的 CodeBuild GitClone 權限。

第 4 步：在構建輸出中查看存儲庫命令

若要檢視組建輸出

1. 成功更新服務角色後，請在失敗的 CodeBuild 階段中選擇「重試」。
2. 管道成功執行之後，在成功的建置階段中，選擇 [檢視詳細資料]。

   在詳細資料頁面上，選擇記錄檔索引標籤。檢視組 CodeBuild 建輸出。這些指令會輸出所輸入變數的值。

   這些命令會輸出`README.md`檔案內容、列出目錄中的檔案、複製存放庫、檢視記錄並執行`git describe --all`。

教學課程：使用AWS CloudFormation StackSets 部署動作建立管道

在本教學課程中，您會使用AWS CodePipeline主控台建立管道，其中包含建立堆疊集和建立堆疊執行個體的部署動作。管道執行時，範本會建立堆疊集，並建立和更新部署堆疊集的執行個體。

管理堆疊集許可的方式有兩種：自我管理和受AWS管 IAM 角色。本教學課程提供具有自我管理權限的範例。

為了最有效地使用 Stacksets CodePipeline，您應該對背後的概念以AWS CloudFormation StackSets及它們的工作方式有清晰的了解。請參閱《AWS CloudFormation使用者指南》中的StackSets 概念。

> **Note**
>
> 該`CloudFormationStackSet`和`CloudFormationStackInstances`行動不適用於亞太區域 (香港)、歐洲 (蘇黎世)、歐洲 (米蘭)、非洲 (開普敦) 及中東 (巴林) 等地區。若要參考其他可用動作，請參閱產品與服務整合 CodePipeline。

主題

必要條件

對於堆疊集作業，您可以使用兩個不同的帳戶：管理帳戶和目標帳戶。您可以在管理員帳戶中建立堆疊集。您可以建立屬於目標帳戶中堆疊集的個別堆疊。

使用您的管理員帳戶建立系統管理員角色

- 依照設定堆疊集作業的基本權限中的指示進行。您的角色必須命名**AWSCloudFormationStackSetAdministrationRole**。

若要在目標帳戶中建立服務角色

- 在信任管理員帳戶的目標帳戶中建立服務角色。依照設定堆疊集作業的基本權限中的指示進行。您的角色必須命名**AWSCloudFormationStackSetExecutionRole**。

步驟 1：上傳範例AWS CloudFormation範本和參數檔案

為堆疊集範本和參數檔案建立來源值區。下載範例AWS CloudFormation範本檔案、設定參數檔案，然後壓縮檔案，然後再上傳到 S3 來源儲存貯體。

> **Note**
>
> 在上傳到 S3 來源儲存貯體之前，請務必先壓縮來源檔案，即使唯一的來源檔案是範本也一樣。

若要建立 S3 來源儲存貯體

1. 登入 AWS Management Console，並開啟位於 https://console.aws.amazon.com/s3/ 的 Amazon S3 主控台。
2. 選擇 Create bucket (建立儲存貯體)。
3. 在儲存貯體名稱中，輸入儲存貯體的名稱。

   在「區域」中，選擇您要在其中建立管線的「區域」。選擇建立儲存貯體。
4. 建立儲存貯體後，會顯示成功橫幅。選擇 Go to bucket details (前往儲存貯體詳細資訊)。
5. 在 Properties (屬性) 標籤上，選擇 Versioning (版本控制)。選擇 Enable versioning (啟用版本控制)，然後選擇 Save (儲存)。

建立AWS CloudFormation樣板檔的步驟

1. 下載下列範例範本檔案，以產生堆疊集的 CloudTrail 組態：https://s3.amazonaws.com/cloudformation-stackset-sample-templates-us-east-1/EnableAWSCloudtrail.yml
2. 儲存檔案為 `template.yml`。

若要建立 parameters.txt 檔案

1. 使用部署的參數建立檔案。參數是您要在運行時在堆棧中更新的值。下列範例檔案會更新堆疊集的範本參數，以啟用記錄驗證和全域事件。

```
[
  {
    "ParameterKey": "EnableLogFileValidation",
    "ParameterValue": "true"
  },
  {
    "ParameterKey": "IncludeGlobalEvents",
    "ParameterValue": "true"
  }
]
```

2. 儲存檔案為 `parameters.txt`。

若要建立 accounts.txt 檔案

1. 使用您要建立執行個體的帳戶建立檔案，如下列範例檔案所示。

```
[
    "111111222222","333333444444"
]
```

2. 儲存檔案為 `accounts.txt`。

若要建立和上傳來源檔案

1. 將檔案合併為單一 ZIP 檔案。您的文件在 ZIP 文件中看起來應該是這樣的。

```
template.yml
parameters.txt
accounts.txt
```

2. 將 ZIP 檔案上傳到您的 S3 儲存貯體。此檔案是由「建立管線」精靈針對中的部署動作所建立的來源人工因素 CodePipeline。

步驟 2：建立管道

在本節中，您可以採取下列動作建立管道：

- 具有 S3 來源動作的來源階段，其中來源成品是您的範本檔案和任何支援的來源檔案。
- 具有建立AWS CloudFormation堆疊集合部署動作的部署階段。
- 具有AWS CloudFormation堆疊執行個體部署動作的部署階段，可在目標帳戶內建立堆疊和執行個體。

欲使用 CloudFormationStackSet 動作建立配管

1. 請登入AWS Management Console並開啟 CodePipeline 主控台，網址為 http://console.aws.amazon.com/codesuite/codepipeline/home。
2. 在 Welcome (歡迎) 頁面、Getting started (入門) 頁面、或者 Pipelines (管道) 頁面上，選擇 Create pipeline (建立管道)。
3. 在 Step 1: Choose pipeline settings (步驟 1：選擇管道設定) 的 Pipeline name (管道名稱) 中，輸入 **`MyStackSetsPipeline`**。

4. 在管線類型中，針對本教學課程的目的選擇 V1。您也可以選擇 V2；但是請注意，管道類型在特性和價格上有所不同。如需詳細資訊，請參閱管線類型。
5. 在 [服務角色] 中，選擇 [新增服務角色] CodePipeline 以允許在 IAM 中建立服務角色。
6. 在「Artifact」存放區中，保留預設值。

> **Note**
>
> 這不是原始碼的來源儲存貯體。這是管道的成品存放區。每個管道都需要有個別成品存放區，例如 S3 儲存貯體。建立或編輯管道時，管線區域中必須有一個人工因素儲存貯體，而每個執行動作的「AWS區域」都必須有一個成品儲存貯體。
> 如需詳細資訊，請參閱 輸入和輸出成品 及 CodePipeline 配管結構參照。

選擇下一步。

7. 在 Step 2: Add source stage (步驟 2：新增來源階段) 頁面的 Source provider (來源提供者) 中，選擇 Amazon S3 (Amazon S3)。
8. 在儲存貯體中，輸入您為此教學課程建立的 S3 來源儲存貯體，例如`BucketName`。在 S3 物件金鑰中，輸入 ZIP 檔案的檔案路徑和檔案名稱，例如`MyFiles.zip`。
9. 選擇下一步。
10. 在 Step 3: Add build stage (步驟 3：新增建置階段) 中，選擇 Skip build stage (跳過建置階段)，然後再次選擇 Skip (跳過) 來接受警告訊息。

    選擇下一步。
11. 在 Step 4: Add deploy stage (步驟 4：新增部署階段) 中：
    a. 在部署提供者中，選擇AWS CloudFormation堆疊集。
    b. 在堆疊集名稱中，輸入堆疊集的名稱。這是該模板創建的堆棧集的名稱。

    > **Note**
    >
    > 記下您的堆疊集名稱。當您將第二個 StackSets 部署動作新增至管道時，您將使用它。

    c. 在範本路徑中，輸入您上載範本檔案的人工因素名稱和檔案路徑。例如，使用預設來源人工因素名稱輸入下列內容`SourceArtifact`。

```
SourceArtifact::template.yml
```

d. 在部署目標中，輸入您上傳帳戶檔案的成品名稱和檔案路徑。例如，使用預設來源人工因素名稱輸入下列內容SourceArtifact。

```
SourceArtifact::accounts.txt
```

e. 在部署目標中AWS 區域，輸入一個區域以部署初始堆疊執行個體，例如us-east-1。

f. 展開部署選項。在參數中，輸入您上載參數檔案的人工因素名稱和檔案路徑。例如，使用預設來源人工因素名稱輸入下列內容SourceArtifact。

```
SourceArtifact::parameters.txt
```

若要將參數輸入為常值輸入而非檔案路徑，請輸入下列內容：

```
ParameterKey=EnableLogFileValidation,ParameterValue=true
ParameterKey=IncludeGlobalEvents,ParameterValue=true
```

g. 在 [功能] 中，選擇 [能力 _IAM 和能力名稱]。

h. 在 [權限] 模型中，選擇 [自我管理]。

i. 在失敗公差百分比中，輸入20。

j. 在最大並行百分比中，輸入25。

k. 選擇下一步。

l. 選擇 Create pipeline (建立管道)。您的管道隨即顯示。

m. 允許您的管道執行。

步驟 3：檢視初始部署

檢視初始部署的資源和狀態。確認部署成功建立堆疊集之後，您可以將第二個動作新增至部署階段。

若要檢視資源

1. 開啟主 CodePipeline 控台，網址為 https://console.aws.amazon.com/codepipeline/。
2. 在 Pipelines (管道) 下，選擇您的管道，然後選擇 View (檢視)。本圖顯示您的管道來源和部署階段。

3. 選擇管道中AWS CloudFormationCloudFormationStackSet動作的動作。AWS CloudFormation主控台中會顯示堆疊集的範本、資源和事件。
4. 在左側導覽面板中，選擇StackSets。在清單中，選擇新的堆疊組。
5. 選擇堆疊執行個體索引標籤。確認您提供的每個帳戶都是在 us-east-1 區域中建立一個堆疊執行個體。確認每個堆疊執行個體的狀態為CURRENT。

步驟 4：新增 CloudFormationStackInstances動作

在管道中建立下一個動作，AWS CloudFormation StackSets 以便建立剩餘的堆疊執行個體。

在管道中建立下一個動作

1. 開啟主 CodePipeline 控台，網址為 https://console.aws.amazon.com/codepipeline/。

   在 Pipelines (管道) 下，選擇您的管道，然後選擇 View (檢視)。本圖顯示您的管道來源和部署階段。
2. 選擇編輯管道。配管會以 「編輯」 模式顯示。
3. 在「部署」階段中，選擇 「編輯」。
4. 在「AWS CloudFormation堆疊集合部署」動作下，選擇「新增動作群組」。
5. 在 「編輯動作」 頁面上，新增動作詳細資訊：

   a. 在動作名稱中，輸入動作的名稱。

   b. 在動作提供者中，選擇AWS CloudFormation堆疊執行個體。

   c. 在「輸入人工因素」下選擇SourceArtifact。

   d. 在堆疊集名稱中，輸入堆疊集的名稱。這是您在第一個動作中提供的堆疊集的名稱。

   e. 在部署目標中，輸入您上傳帳戶檔案的成品名稱和檔案路徑。例如，使用預設來源人工因素名稱輸入下列內容SourceArtifact。

   ```
   SourceArtifact::accounts.txt
   ```

   f. 在部署目標中AWS 區域，輸入用於部署剩餘堆疊執行個體的區域us-east-2，eu-central-1如下所示：

   ```
   us-east2, eu-central-1
   ```

   g. 在失敗公差百分比中，輸入20。

h. 在最大並行百分比中，輸入25。

i. 選擇儲存。

j. 。手動發行「變更」。您更新的管道會在「部署」階段中顯示兩個動作。

步驟 5：檢視部署的堆疊集資源

您可以檢視堆疊集部署的資源和狀態。

若要檢視資源

1. 開啟主 CodePipeline 控台，網址為 https://console.aws.amazon.com/codepipeline/。
2. 在「配管」下，選擇您的管線，然後選擇「檢視」。本圖顯示您的管道來源和部署階段。
3. 選擇管道中AWS CloudFormation**AWS CloudFormation Stack Instances**動作的動作。AWS CloudFormation主控台中會顯示堆疊集的範本、資源和事件。
4. 在左側導覽面板中，選擇StackSets。在清單中，選擇您的堆疊組合。
5. 選擇堆疊執行個體索引標籤。確認您提供的每個帳戶的所有剩餘堆疊執行個體都已在預期的區域中建立或更新。確認每個堆疊執行個體的狀態為CURRENT。

步驟 6：對堆疊集進行更新

對您的堆疊集進行更新，並將更新部署到執行個體。在此範例中，您也會變更要指定進行更新的部署目標。不屬於更新的實例會移至過期狀態。

1. 開啟主 CodePipeline 控台，網址為 https://console.aws.amazon.com/codepipeline/。
2. 在「配管」下，選擇您的配管，然後選擇「編輯」。在「部署」階段中，選擇 「編輯」。
3. 選擇此選項可在管線中編輯「AWS CloudFormation堆疊集合」動作。在說明中，寫入現有的描述與堆疊集的新描述。
4. 選擇此選項可在管線中編輯「AWS CloudFormation堆疊執行個體」動作。在部署目標中AWS 區域，刪除建立動作時輸入的`us-east-2`值。
5. 儲存變更。選擇 [發行變更] 以執行管道。
6. 在中開啟您的動作AWS CloudFormation。選擇「StackSet 資訊」標籤。在StackSet 說明中，確認已顯示新描述。
7. 選擇堆疊執行個體索引標籤。在狀態下，確認 us-east-2 中堆疊執行個體的狀態為。OUTDATED

CodePipeline 最佳做法和使用案例

下列各節說明的最佳作法 CodePipeline。

主題

使用案例 CodePipeline

您可以建立與其他管道整合的管線 AWS 服務。這些可以是 AWS 服務，如 Amazon S3，或第三方產品，如 GitHub. 本節提供使用不同產品整合 CodePipeline 來自動化程式碼發行的範例。如需依動作類型 CodePipeline 組織的整合完整清單，請參閱CodePipeline 配管結構參照。

主題

CodePipeline 與 Amazon S3 一起使用 AWS CodeCommit，和 AWS CodeDeploy

建立管道時，會 CodePipeline 與在管道每個階段充當動作提供者的 AWS 產品和服務整合。當您在精靈中選擇階段時，必須選擇來源階段以及至少建置或部署階段。此精靈會使用無法變更的預設名稱來建立階段。這些階段名稱是您在精靈中設定完整三階段管道時所建立：

- 預設名稱為 "Source" 的來源動作階段。
- 預設名稱為 "Build" 的建置動作階段。
- 預設名稱為 "Staging" 的部署動作階段。

您可以使用本指南中的教學來建立管道並指定階段：

- 中的步驟可教學：建立簡易管道 (S3 儲存貯體)協助您使用精靈建立具有兩個預設階段的管道：「來源」和「暫存」，其中 Amazon S3 儲存庫是來源供應商。本教學建立一個管道，用於 AWS CodeDeploy 將範例應用程式從 Amazon S3 儲存貯體部署到執行 Amazon Linux 的 Amazon EC2 執行個體。
- 中的步驟可教學：建立範本管道 (CodeCommit 儲存庫)協助您使用精靈建立具有「來源」階段的管道，該階段會使用您的 AWS CodeCommit 存放庫做為來源提供者。本教學課程建立一個管道，用 AWS CodeDeploy 於將範例應用程式從 AWS CodeCommit 儲存庫部署到執行 Amazon Linux 的 Amazon EC2 執行個體。

CodePipeline 與第三方動作提供者 (以GitHub及 Jenkins) 搭配使用

您可以建立與第三方產品 (例如 GitHub 和 Jenkins) 整合的管道。教學：建立四階段管道中的步驟顯示如何建立管道，以：

- 從 GitHub 存儲庫獲取源代碼，
- 使用 Jenkins 建置和測試來源碼、
- 用於 AWS CodeDeploy 將構建和測試的源代碼部署到運行 Amazon Linux 或 Microsoft Windows 服務器的亞馬遜 EC2 實例。

CodePipeline 搭配 AWS CodeStar 使用在程式碼專案中建置管線

AWS CodeStar 是一種基於雲的服務，提供統一的用戶界面來管理軟件開發項目 AWS。 AWS CodeStar 使用將 AWS 資源合併 CodePipeline 到項目開發工具鏈中。您可以使用 AWS CodeStar 儀表板自動建立管道、儲存庫、原始程式碼、建置規格檔案、部署方法，以及託管完整程式碼專案所需的執行個體或無伺服器執行個體。

若要建立 AWS CodeStar 專案，您可以選擇編碼語言和要部署的應用程式類型。您可以建立下列專案類型：Web 應用程式、Web 服務或 Alexa 技能。

您可以隨時將偏好的 IDE 整合到 AWS CodeStar 儀表板中。您也可以新增和移除團隊成員，以及管理團隊成員對您專案的許可。如需說明如何為無伺服器應用 AWS CodeStar 程式建立範例管線的教學課程，請參閱中的教學課程：建立和管理無伺服器專案。 AWS CodeStar

用 CodePipeline 於編譯、建置和測試程式碼 CodeBuild

CodeBuild 是雲端中的託管構建服務，可讓您在沒有服務器或系統的情況下構建和測試代碼。CodePipeline 搭配使用可透過管道自動執行修訂， CodeBuild 以便在原始程式碼發生變更時持續交付軟體組建。 如需詳細資訊，請參閱 CodePipeline 搭配使用 CodeBuild 來測試程式碼和執行組建。

CodePipeline 搭配 Amazon ECS 使用，將容器型應用程式持續交付到雲端

Amazon ECS 是一種容器管理服務，可讓您將容器型應用程式部署到雲端中的 Amazon ECS 執行個體。 CodePipeline 與 Amazon ECS 搭配使用，透過管道自動執行修訂，以便在來源映像儲存庫發生變更時持續部署容器型應用程式。如需詳細資訊，請參閱教學：使用 CodePipeline 持續部署。

CodePipeline 與 Elastic Beanstalk 一起使用，將 Web 應用程式持續傳遞到雲端

Elastic Beanstalk 是一種計算服務，可讓您將 Web 應用程序和服務部署到 Web 服務器。CodePipeline 搭配 Elastic Beanstalk 使用，可將 Web 應用程式持續部署到您的應用程式環境。您也可以使用透過 Elastic Beanstalk 部署動作 AWS CodeStar 來建立管道。

CodePipeline 搭配使用以 Lambda AWS Lambda 為基礎和無伺服器應用程式的持續交付

您可以使用 AWS Lambda 與來叫 CodePipeline 用 AWS Lambda 函數，如部署無伺服器應用程式中所述。您也可以使用 AWS Lambda 和建立管道 AWS CodeStar 來部署無伺服器應用程式。

CodePipeline 搭配 AWS CloudFormation 範本使用，以持續傳遞至雲端

您可以與一 AWS CloudFormation 起 CodePipeline 使用，以持續交付和自動化。如需詳細資訊，請參閱使用 CodePipeline. AWS CloudFormation 也可用於為中建立的配管建立範本 AWS CodeStar。

標記 資源

標籤是一種自訂屬性標籤，可由您或 AWS 指派給 AWS 資源。每個 AWS 標籤都有兩個部分：

- 標籤鍵 (例如，CostCenter、Environment、Project 或 Secret)。標籤鍵會區分大小寫。
- 一個名為標籤值 (例如，111122223333 或 Production，或團隊名稱) 的選用欄位。忽略標籤值基本上等同於使用空字串。與標籤鍵相同，標籤值會區分大小寫。

這些合稱為鍵值組。

標籤可協助您識別和整理 AWS 資源。許多 AWS 服務支援標記，因此您可以對來自不同服務的資源指派相同的標籤，指出資源是相關的。例如，您可以將相同的標籤指派給您指派給 Amazon S3 儲存貯體的管道。

如需使用標籤的提示，請參閱 AWS 答案部落格上的 AWS 標記策略文章。

您可以在 CodePipeline 中標記下列資源類型：

- 在 CodePipeline 中標記管道
- 在 CodePipeline 中標記自訂動作

您可以使用AWS CLI、 CodePipeline API 或AWS開發套件來執行下列作業：

- 您可以在建立管道、自訂動作或 Webhook 時，將標籤新增到這些管道、自訂動作或 Webhook。
- 新增、管理和移除管道、自訂動作或 Webhook 的標籤。

您也可以使用主控台來新增、管理和移除管道的標籤。

此外，若要使用標籤來識別、整理和追蹤您的資源，您可以在 IAM 政策中使用標籤，以協助控制誰可以檢視您的資源並與其互動。如需以標籤為基礎的存取政策範例，請參閱使用標籤來控制對 CodePipeline 資源的存取。

CodePipeline 與 Amazon Virtual Private Cloud 一起使用

AWS CodePipeline現在支援採用技術的 Amazon Virtual Private Cloud (Amazon VPC) 端點AWS PrivateLink。這意味著您可以通 CodePipeline 過 VPC 中的私有端點直接連接，從而將所有流量保留在 VPC 和AWS網絡中。

Amazon VPC 是一項AWS 服務可用來在自己定義的虛擬網路中啟動AWS資源。使用 VPC，您可以控制網路設定，例如：

- IP 地址範圍
- 子網
- 路由表
- 網路閘道

VPC 端點是由技術AWS PrivateLink，一項AWS技術，讓私有通訊AWS 服務使用於 elastic network interface 與私有 IP 地址之間的私有 IP 地址。若要將您的 VPC 連接到 CodePipeline，請定義的 VPC 端點介面 CodePipeline。這種類型的端點使您可以將 VPC 連接到AWS 服務。端點能為提供可靠、可擴展性的連線， CodePipeline 無須使用網際網路閘道、網路位址轉譯 (NAT) 執行個體或 VPN 連接。如需設定 VPC 的相關資訊，請參閱 VPC 使用者指南。

可用性

CodePipeline 目前在下列方支援 VPC 端點AWS 區域：

- 美國東部 (俄亥俄)
- 美國東部 (維吉尼亞北部)
- 美國西部 (加利佛尼亞北部)
- 美國西部 (奧勒岡)
- 加拿大 (中部)
- 歐洲 (法蘭克福)
- 歐洲 (愛爾蘭)
- 歐洲 (倫敦)
- 歐洲 (米米) 米) * 米)
- Europe (Paris)

- Europe (Stockholm)
- Asia Pacific (香) *
- 亞太區域 (孟買)
- 亞太區域 (東京)
- 亞太區域 (首爾)
- 亞太區域 (新加坡)
- 亞太區域 (雪梨)
- 南美洲 (聖保羅)
- AWS GovCloud (美國西部)

* 您必須先啟用此區域才能使用它。

建立 CodePipeline 的 VPC 端點

您可以使用 Amazon VPC 主控台來建立。 ##. 代碼管道 VPC 端點。在主控台中，#域是支AWS 區域援的 CodePipeline區域的區域的區域的區域。us-east-2如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的建立界面端點。

端點會預先填入您登入 AWS 時所指定的區域。如果您登入另一個區域，則 VPC 端點會隨著新的區域而更新。

> **Note**
>
> 提供 VPC 支援並與 CodePipeline之整合的其他AWS 服務功能 (例如 CodeCommit) 可能不支援使用 Amazon VPC 端點進行該整合。例如， CodePipeline 和之間的流量 CodeCommit 不能限制在 VPC 子網路範圍內。

為您的 VPC 設定進行故障診斷

對 VPC 問題進行疑難排解時，請使用網際網路連線錯誤訊息中出現的資訊來協助您查明、診斷和解決問題。

1. 確定您的網際網路閘道連接至您的 VPC。
2. 確定公有子網路的路由表指向網際網路閘道。

3. 確定您的網路 ACL 允許流量流動。
4. 確定您的安全群組允許流量流動。
5. 請確定私有子網路的路由表指向虛擬私有閘道。
6. 請確定所使用的服務角色 CodePipeline 具有適當的權限。例如，如果 CodePipeline 沒有使用 Amazon VPC 所需的 Amazon EC2 許可，您可能會收到錯誤訊息，顯示「意外的 EC2 錯誤：」UnauthorizedOperation。

使用中的管線 CodePipeline

若要在 AWS CodePipeline 中定義自動化發佈程序，您將建立管道以做為描述軟體變更如何經過發佈程序的工作流程架構。管道是由您設定的階段與動作所組成。

> **Note**
>
> 當您新增「建置」、「部署」、「測試」或「叫用」階段時，除了提供的預設選項之外 CodePipeline，您還可以選擇已建立的自訂動作，以便與管道搭配使用。自訂動作可用於例如執行內部開發建置程序或測試套件等任務。包含的版本識別符，可協助您區分供應商清單中的不同版本自訂動作。如需詳細資訊，請參閱在 CodePipeline 中建立和新增自訂動作。

在您可建立管道前，您必須先完成 開始使用 CodePipeline 中的步驟。

如需有關管道的詳細資訊，請參閱 CodePipeline 概念 和 CodePipeline 教程；而如果您想要使用 AWS CLI 來建立管道，請參閱 CodePipeline 配管結構參照。若要檢視管道清單，請參閱 (在中檢視管線和詳細資訊 CodePipeline)。

主題

- 在中啟動管道 CodePipeline
- 停止管線執行 CodePipeline
- 在中建立管線 CodePipeline
- 在中編輯配管 CodePipeline
- 在中檢視管線和詳細資訊 CodePipeline
- 在 CodePipeline 中刪除管道
- 在 CodePipeline 中建立使用來自另一個 AWS 帳戶資源的管道
- 移轉輪詢管道以使用事件型變更偵測
- 建立 CodePipeline 服務角色
- 在 CodePipeline 中標記管道
- 建立通知規則

在中啟動管道 CodePipeline

每個管線執行都可以根據不同的觸發程序來啟動。視管線的啟動方式而定，每個管線執行都可以有不同類型的觸發器。每個執行的觸發器類型都會顯示在管線的執行歷史記錄中。觸發類型可以取決於來源動作提供者，如下所示：

> **Note**
>
> 您無法為每個來源動作指定一個以上的觸發器。

- 管線建立：建立管線時，管線會自動開始執行。這是執行歷史記錄中的`CreatePipeline`觸發器類型。
- 修訂物件的變更：此類別代表「執行」歷史記錄中的`PutActionRevision`觸發程式類型。
- 更改分支上的檢測並提交代碼推送：此類別代表執行歷史記錄中的`CloudWatchEvent`觸發器類型。當偵測到來源儲存庫中的來源提交和分支的變更時，您的管道就會啟動。此觸發類型使用自動變更偵測。使用此觸發器類型的來源動作提供者為 S3 和 CodeCommit. 此類型也可用於啟動管線的排程。請參閱 按照排程啟動配管。
- 輪詢來源變更：此類別代表「執行」歷史記錄中的`PollForSourceChanges`觸發程式類型。當透過輪詢偵測到來源認可和來源儲存庫中的變更時，您的管道會啟動。不建議使用此觸發器類型，因此應移轉為使用自動變更偵測。使用此觸發器類型的來源動作提供者為 S3 和 CodeCommit.
- 第三方來源的 Webhook 事件：此類別代表「執行」歷史記錄中的`Webhook`觸發器類型。當 webhook 事件偵測到變更時，您的管線會啟動。此觸發類型使用自動變更偵測。使用此觸發器類型的來源動作提供者是針對程式碼推送 (Bitbucket Cloud、 GitHub 企業伺服器 GitHub、 GitLab .com 和 GitLab自我管理) 設定的連線。
- 第三方來源的 WebHookv2 事件：此類別代表「執行」歷史記錄中的`WebhookV2`觸發程序類型。此類型適用於根據管線定義中定義的觸發程序觸發的執行。當偵測到具有指定 Git 標籤的發行版本時，您的管線會啟動。您可以使用 Git 標籤以名稱或其他識別符來標記遞交，以協助其他儲存庫使用者了解其重要性。您也可以使用 Git 標籤來識別儲存庫歷史記錄中的特定遞交。此觸發類型會停用自動變更偵測。使用此觸發器類型的來源動作提供者是針對 Git 標籤 (Bitbucket 雲端 GitHub、 GitHub 企業伺服器和 GitLab .com) 設定的連線。
- 手動啟動管線：此類別代表「執行」歷史記錄中的`StartPipelineExecution`觸發器類型。您可以使用控制台或手動啟動管線。 AWS CLI 如需相關資訊，請參閱 手動啟動管道。

當您將使用自動變更偵測觸發器類型的管道新增來源動作時，這些動作會與其他資源搭配使用。由於這些用於變更偵測的額外資源，因此建立每個來源動作會在不同的章節中詳述。如需有關每個來源提供者以及自動變更偵測所需的變更偵測方法的詳細資訊，請參閱 來源動作和變更偵測方法。

主題

來源動作和變更偵測方法

當您將來源動作新增至管線時，動作會與表格中所述的其他資源搭配使用。

> **Note**
>
> CodeCommit 和 S3 來源動作需要設定的變更偵測資源（EventBridge 規則），或使用選項輪詢存放庫以進行來源變更。對於具有 Bitbucket GitHub、或 GitHub 企業伺服器來源動作的管線，您不需要設定 Webhook 或預設進行輪詢。連線動作會為您管理變更偵測。

來源	是否使用其他資源？	步驟
Amazon S3	此來源動作會使用其他資源。當您使用 CLI 或 CloudFormation 建立此動作時，您也會建立和管理這些資源。	請參閱在中建立管線 CodePipeline和 Amazon S3 來源動作 EventBridge 與 AWS CloudTrail
Bitbucket Cloud	此來源動作會使用連線資源。	請參閱 比特桶雲連接
AWS CodeCommit	Amazon EventBridge （推薦）。這是用於在主控台中建立或編輯並擁有 CodeCommit 來源之管道的預設值。	請參閱在中建立管線 CodePipeline和 CodeCommit 來源動作和 EventBridge

來源	是否使用其他資源？	步驟
Amazon ECR	Amazon EventBridge。這是由精靈針對管道建立或在主控台中建立或編輯的 Amazon ECR 來源所建立。	請參閱 在中建立管線 CodePipeline 和 Amazon ECR 來源動作和 EventBridge 資源。
GitHub 或 GitHub 企業雲	此來源動作會使用連線資源。	請參閱 GitHub 連接
GitHub 企業伺服器	此來源動作使用連線資源和主機資源。	請參閱 GitHub 企業伺服器連線
GitLab.com	此來源動作會使用連線資源。	請參閱 GitLab.com 連線
GitLab 自我管理	此來源動作使用連線資源和主機資源。	請參閱 GitLab 自我管理的連線

如果您有使用輪詢的管道，您可以更新管道以使用建議的偵測方法。如需詳細資訊，請參閱 將輪詢管道更新至建議的變更偵測方法。

如果您要關閉使用連線的來源動作的變更偵測，請參閱CodeStarSourceConnection 適用於 Bitbucket 雲端 GitHub、 GitHub 企業伺服器、 GitLab .com 和 GitLab 自我管理動作。

手動啟動管道

根據預設，建立管道時，以及隨時在來源儲存庫中進行變更時，管道就會自動啟動。不過，您可能會想要再次透過管道，重新執行最新的修訂版本。您可以使用 CodePipeline 控制台或 AWS CLI and start-pipeline-execution 命令，透過管道手動重新執行最新的修訂版本。

主題

- 手動啟動管道 (主控台)
- 手動啟動管道 (CLI)

手動啟動管道 (主控台)

手動啟動管道，並透過管道執行最新的修訂版本

1. 請登入 AWS Management Console 並開啟 CodePipeline 主控台，網址為 http://console.aws.amazon.com/codesuite/codepipeline/home。
2. 在 Name (名稱) 中，選擇您想啟動的管道名稱。
3. 在配管詳細資訊頁面上，選擇「發行變更」。如果配管配置為傳遞參數 (管線變數)，則選擇「發行變更」(Release change) 會開啟「發行變更」視窗。在配管層級變數的一或多個變數欄位中，輸入要為此管線執行傳遞的一或多個值。如需詳細資訊，請參閱 Variables。

   這將會啟動各來源位置的最新可用修訂版本；這些來源位置透過管道的來源動作指定。

手動啟動管道 (CLI)

手動啟動管道，並透過管道執行最近的成品版本

1. 開啟終端機 (Linux、macOS 或 Unix) 或命令提示字元 (Windows)，然後使用 AWS CLI 來執行指start-pipeline-execution令，並指定您要啟動的管線名稱。例如，若要透過名為 *MyFirstPipeline* 的管道開始執行最後的變更：

   ```
   aws codepipeline start-pipeline-execution --name MyFirstPipeline
   ```

   若要啟動在管線層級設定變數的管線，請使用具有選用--variables引數的start-pipeline-execution指令來啟動管線，並新增將在執行中使用的變數。例如，若要新增值為var1的變數1，請使用下列命令：

   ```
   aws codepipeline start-pipeline-execution --name MyFirstPipeline --variables
    name=var1,value=1
   ```

2. 請檢視回傳的物件以驗證是否成功。此命令會傳回如下的執行 ID：

   ```
   {
       "pipelineExecutionId": "c53dbd42-This-Is-An-Example"
   }
   ```

> **Note**
>
> 啟動管線之後，您可以在 CodePipeline 主控台中或執行get-pipeline-state指令來監視其進度。如需詳細資訊，請參閱 檢視管線 (主控台) 及 檢視管道詳細資訊與歷程記錄 (CLI)。

按照排程啟動配管

您可以在中設定規則， EventBridge 以按照排程啟動配管。

建立排程管線啟動的 EventBridge 規則 (主控台)

建立以排程作為事件來源的 EventBridge 規則

1. 在以下位置打開 Amazon EventBridge 控制台 https://console.aws.amazon.com/events/。
2. 在導覽窗格中，選擇規則。
3. 選擇 [建立規則]，然後在 [規則詳細資訊] 下選擇 [排程]
4. 使用固定頻率或運算式來設定排程。如需詳細資訊，請參閱適用於規則的排程運算式。
5. 在 [目標] 中，選擇CodePipeline。
6. 針對此排程輸入配管執行的管線 ARN。

> **Note**
>
> 您可以在控制台的「設置」 下找到管道 ARN。請參閱 檢視管線 ARN 和服務角色 ARN (主控台)。

7. 選擇下列其中一個選項以建立或指定 IAM 服務角色，該角色 EventBridge 授予許可以叫用與 EventBridge 規則關聯的目標 (在此情況下，目標為 CodePipeline)。
   - 選擇 [為此特定資源建立新角色]，以建立授與啟動管線執行之 EventBridge權限的服務角色。
   - 選擇 [使用現有角色] 以輸入授與啟動管線執行之 EventBridge 權限的服務角色。
8. 選擇 Configure details (設定詳細資訊)。
9. 在 Configure rule details (設定規則詳細資訊) 頁面上，輸入規則的名稱和描述，然後選擇 State (狀態) 啟用規則。
10. 如果您對此規則感到滿意，請選擇 Create rule (建立規則)。

建立 EventBridge 排程管線以啟動 (CLI) 的規則

若要使用 AWS CLI 建立規則，請呼叫指put-rule令，並指定：

- 可唯一識別您所建立規則的名稱。在與 AWS 帳戶 CodePipeline 相關聯的所有管道中，此名稱必須是唯一的。
- 適用於規則的排程運算式。

建立以排程作為事件來源的 EventBridge 規則

1. 呼叫 put-rule 命令，並包含 `--name` 和 `--schedule-expression` 參數。

   範例：

   下列範例指令用--schedule-expression來建立名為依排MyRule2程篩選 EventBridge 的規則。

   ```
   aws events put-rule --schedule-expression 'cron(15 10 ? * 6L 2002-2005)' --name
    MyRule2
   ```

2. 授與用 EventBridge CodePipeline 於呼叫規則的權限。如需詳細資訊，請參閱在 Amazon EventBridge 使用以資源為基礎的政策。

   a. 使用以下範本來建立信任政策，以允許 EventBridge 擔任服務角色。將其命名為 `trustpolicyforEB.json`。

   ```
   {
       "Version": "2012-10-17",
       "Statement": [
           {
               "Effect": "Allow",
               "Principal": {
                   "Service": "events.amazonaws.com"
               },
               "Action": "sts:AssumeRole"
           }
       ]
   }
   ```

   b. 使用下列命令來建立 `Role-for-MyRule` 角色，並連接信任政策。

```
aws iam create-role --role-name Role-for-MyRule --assume-role-policy-document
 file://trustpolicyforEB.json
```

c. 為名為 MyFirstPipeline 的管道建立許可政策 JSON，如這個範例所示。將許可政策命名為 permissionspolicyforEB.json。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codepipeline:StartPipelineExecution"
            ],
            "Resource": [
                "arn:aws:codepipeline:us-west-2:80398EXAMPLE:MyFirstPipeline"
            ]
        }
    ]
}
```

d. 使用執行以下命令，將新的 CodePipeline-Permissions-Policy-for-EB 許可政策連接到您所建立的 Role-for-MyRule 角色。

```
aws iam put-role-policy --role-name Role-for-MyRule --policy-name CodePipeline-
Permissions-Policy-For-EB --policy-document file://permissionspolicyforCWE.json
```

以來源修訂版本取代啟動管線

您可以使用取代，使用您為管線執行提供的特定來源修訂 ID 來啟動管線。例如，如果您想要啟動一個管道來處理 CodeCommit 來源的特定提交 ID，則可以在啟動管道時將提交 ID 新增為覆寫。

「版本修訂類型」有三種類型的來源版本修訂：

- 提交識別碼
- 影像摘要
- S3_ 對象版本 ID

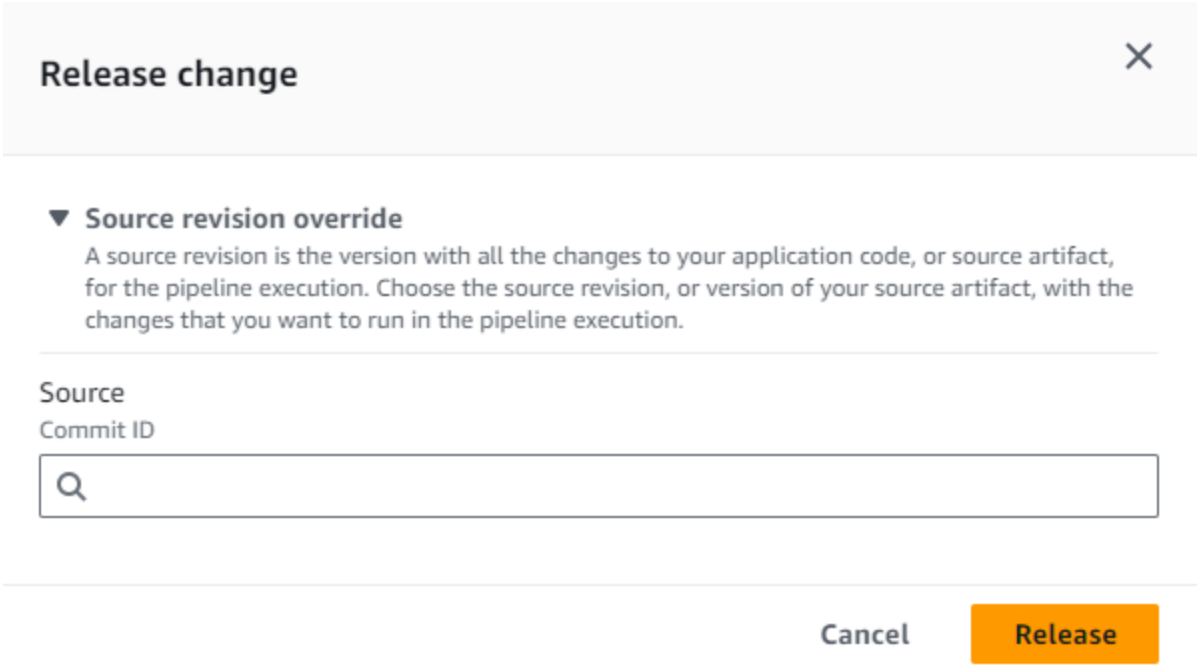
> **Note**
>
> 對於來源修訂版本的 COMMER_ID 和 IMAGE_DIGEST 類型，來源修訂 ID 會套用至存放庫中的所有內容，跨所有分支。

主題

使用來源修訂覆寫 (主控台) 啟動管線

手動啟動管道，並透過管道執行最新的修訂版本

1. 請登入 AWS Management Console 並開啟 CodePipeline 主控台，網址為 http://console.aws.amazon.com/codesuite/codepipeline/home。
2. 在 Name (名稱) 中，選擇您想啟動的管道名稱。
3. 在配管詳細資訊頁面上，選擇「發行變更」。選擇「核發變更」可開啟「核發變更」視窗 對於來源修訂取代，請選擇箭頭以展開欄位。在來源中，輸入來源版次識別碼。例如，如果您的管道具有 CodeCommit 來源，請從您要使用的欄位中選擇提交 ID。



使用來源修訂版取代 (CLI) 啟動管線

若要透過管線手動啟動管線，並針對人工因素執行指定的來源修訂版本 ID

1. 開啟終端機 (Linux、macOS 或 Unix) 或命令提示字元 (Windows)，然後使用 AWS CLI 來執行指start-pipeline-execution令，並指定您要啟動的管線名稱。您也可以使用--source-revisions引數來提供來源修訂 ID。來源版本修訂由「actionName」、「版本修訂類型」和「版本修訂值」組成。有效的「修訂類型」值為。`COMMIT_ID | IMAGE_DIGEST | S3_OBJECT_VERSION_ID`

   在下列範例中，若要透過名為的管線開始執行指定的變更codecommit-pipeline，下列命令會對來源動作名稱進行種類，修訂版本類型為`COMMIT_ID`，以及的提交 ID `78a25c18755ccac3f2a9eec099dEXAMPLE`。

   ```
   aws codepipeline start-pipeline-execution --name codecommit-pipeline --source-
   revisions
    actionName=Source,revisionType=COMMIT_ID,revisionValue=78a25c18755ccac3f2a9eec099dEXAMPLE
    --region us-west-1
   ```

2. 請檢視回傳的物件以驗證是否成功。此命令會傳回如下的執行 ID：

   ```
   {
       "pipelineExecutionId": "c53dbd42-This-Is-An-Example"
   }
   ```

   > **Note**
   >
   > 啟動管線之後，您可以在 CodePipeline 主控台中或執行get-pipeline-state指令來監視其進度。如需更多詳細資訊，請參閱 檢視管線 (主控台) 及 檢視管道詳細資訊與歷程記錄 (CLI)。

停止管線執行 CodePipeline

當管道執行開始透過管道執行時，其一次會進入一個階段，並在階段中的所有動作執行正在執行時鎖定階段。這些進行中的動作必須以某種方式處理，以便在管道執行停止時，允許完成或捨棄動作。

有兩種方式可以停止管道執行：

- 停止等待：AWS CodePipeline 會等待停止執行，直到所有進行中的動作都完成為止 (也就是動作具有 Succeeded 或 Failed 狀態)。此選項會保留進行中的動作。執行會處於 Stopping 狀態，直到進行中的動作完成為止。然後執行會處於 Stopped 狀態。動作完成之後，階段就會解除鎖定。

  如果您選擇停止並等待，而且在執行仍處於 Stopping 狀態時改變主意，您可以選擇捨棄。

- 停止並捨棄：AWS CodePipeline 停止執行，而不等待進行中的動作完成。當進行中的動作被捨棄時，執行會有非常短的時間處於 Stopping 狀態。在執行停止之後，當管道執行處於 Abandoned 狀態時，動作執行會處於 Stopped 狀態。階段會解除鎖定。

  對於處於 Stopped 狀態的管道執行，您可重試執行停止所在階段中的動作。

> **Warning**
>
> 此選項可能會導致工作失敗或工作失序。

主題

停止管道執行 (主控台)

您可以使用控制台停止管道執行。選擇執行，然後選擇停止管線執行的方法。

> **Note**
>
> 您也可以停止作為輸入執行的管道執行。若要深入瞭解如何停止輸入執行，請參閱停止輸入執行 (主控台)。

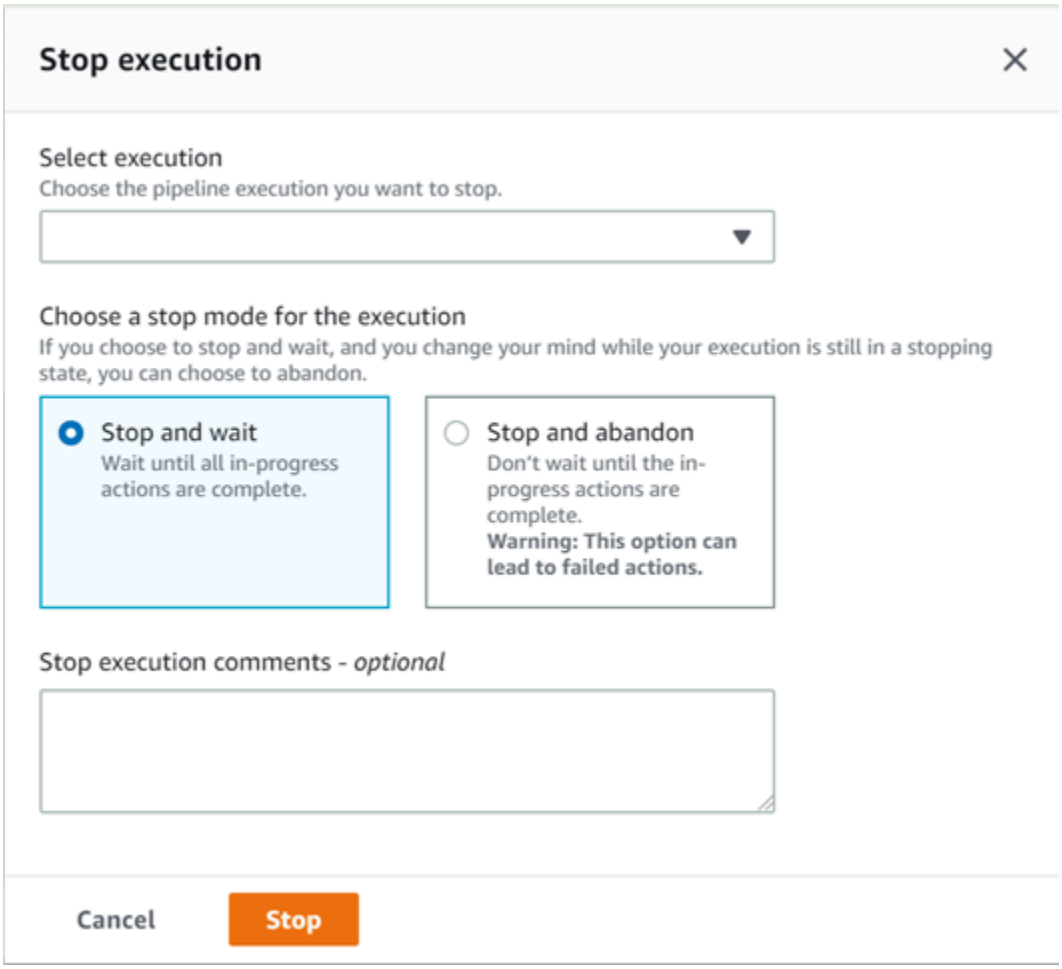
1. 請登入AWS Management Console並開啟 CodePipeline 主控台，網址為 http://console.aws.amazon.com/codesuite/codepipeline/home。
2. 執行下列任意一項：

> **Note**
>
> 在您停止執行之前，我們建議您停用階段前面的轉換。如此一來，當階段由於停止執行而解除鎖定時，階段就不會接受後續的管道執行。

- 在 Name (名稱) 中，選擇您想停止執行的管道名稱。在管道詳細資訊頁面上，選擇 Stop execution (停止執行)。
- 選擇 View history (檢視歷程記錄)。在歷程記錄頁面上，選擇 Stop execution (停止執行)。

3. 在 Stop execution (停止執行) 頁面的 Select execution (選取執行) 之下，選擇您要停止的執行。

> **Note**
>
> 只有仍在進行中的執行才會顯示。不會顯示已經完成的執行。



4. 在 Select an action to apply to execution (選取要套用至執行的動作) 之下，選擇下列其中一項：

   - 若要確保在所有進行中的動作都完成前，執行不會停止，請選擇 Stop and wait (停止並等待)。

     > **Note**
     >
     > 如果執行已處於 Stopping (停止中) 狀態，您就無法選擇停止並等待，但您可以選擇停止並捨棄。

   - 若要停止而不等待進行中的動作完成，請選擇 Stop and abandon (停止並捨棄)。

     > **Warning**
     >
     > 此選項可能會導致工作失敗或工作失序。

5. (選擇性) 輸入註解。這些註解以及執行狀態會顯示在執行的歷程記錄頁面上。

6. 選擇 Stop (停止)。

> **Important**
>
> 這個操作無法復原。

7. 在管道視覺效果中檢視執行狀態，如下所示：

   - 如果您選擇停止並等待，則選取的執行會繼續進行，直到進行中的動作完成為止。
     - 成功橫幅訊息會顯示在主控台的頂端。
     - 在目前階段中，進行中的動作會以 InProgress 狀態繼續進行。當動作正在進行時，管道執行會處於 Stopping 狀態。

       動作完成 (也就是動作失敗或成功) 後，管道執行會變更為 Stopped 狀態，而動作會變更為 Failed 或 Succeeded 狀態。您也可以在執行詳細資訊頁面上檢視動作狀態。您可以在執行歷程記錄頁面或執行詳細資訊頁面上檢視執行狀態。
     - 管道執行會短暫變更為 Stopping 狀態，然後變更為 Stopped 狀態。您可以在執行歷程記錄頁面或執行詳細資訊頁面上檢視執行狀態。
   - 如果您選擇停止並捨棄，則執行不會等待進行中的動作完成。
     - 成功橫幅訊息會顯示在主控台的頂端。
     - 在目前階段中，進行中的動作會變更為 Abandoned 狀態。您也可以在執行詳細資訊頁面上檢視動作狀態。
     - 管道執行會短暫變更為 Stopping 狀態，然後變更為 Stopped 狀態。您可以在執行歷程記錄頁面或執行詳細資訊頁面上檢視執行狀態。

   您可以在執行歷程記錄檢視和詳細歷程記錄檢視中檢視管道執行狀態。

停止輸入執行 (主控台)

您可以使用控制台停止輸入執行。輸入執行是一種管線執行，正在等待進入停用轉換的階段。啟用轉換時，會InProgress繼續進入階段的輸入執行。未進入階段Stopped的輸入執行。

> **Note**
>
> 入站執行停止後，無法重試。

如果您沒有看到輸入執行，則在停用的階段轉換中沒有擱置的執行。

1. 請登入AWS Management Console並開啟 CodePipeline 主控台，網址為 http://console.aws.amazon.com/codesuite/codepipeline/home。

   將會顯示所有與您 AWS 帳戶建立關聯的管道名稱。

2. 選擇您要停止輸入執行的管道名稱，請執行下列其中一個動作：

   - 在「管線」檢視中，選擇輸入執行 ID，然後選擇停止執行。
   - 選擇管線，然後選擇檢視歷史記錄。在執行歷史記錄中，選擇輸入執行 ID，然後選擇停止執行。

3. 在停止執行模式中，請按照上述部分中的步驟選取執行 ID 並指定 stop 方法。

   使用get-pipeline-state命令檢視輸入執行的狀態。

停止管道執行 (CLI)

若要使用 AWS CLI 手動停止管道，請使用 stop-pipeline-execution 命令搭配下列參數：

- 執行 ID (必要)
- 註解 (選擇性)
- 管道名稱 (必要)
- 捨棄旗標 (選擇性，預設值為 false)

命令格式：

```
aws codepipeline stop-pipeline-execution --pipeline-name Pipeline_Name --pipeline-execution-id Execution_ID [--abandon | --no-abandon] [--reason STOP_EXECUTION_REASON]
```

1. 開啟終端機 (Linux、macOS 或 Unix) 或命令提示 (Windows)。
2. 若要停止管道執行，請選擇下列其中一項：

   - 若要確保在所有進行中的動作都完成前，執行不會停止，請選擇停止並等待。您可以納入 no-abandon 參數來完成此操作。如果您未指定參數，則命令會預設為停止並等待。使用 AWS CLI 執行 stop-pipeline-execution 命令，並指定管道名稱和執行 ID。例如，若要以指定的停止並等待選項來停止名為 *MyFirstPipeline* 的管道：

```
aws codepipeline stop-pipeline-execution --pipeline-name MyFirstPipeline --
pipeline-execution-id d-EXAMPLE --no-abandon
```

例如，若要停止名為 *MyFirstPipeline* 的管道、預設為停止並等待選項，並選擇包含註解：

```
aws codepipeline stop-pipeline-execution --pipeline-name MyFirstPipeline --
pipeline-execution-id d-EXAMPLE --reason "Stopping execution after the build
 action is done"
```

> **Note**
>
> 如果執行已處於 Stopping (停止中) 狀態，您就無法選擇停止並等待。您可以選擇停止並捨棄已處於 Stopping (停止中) 狀態的執行。

- 若要停止而不等待進行中的動作完成，請選擇停止並捨棄。納入 abandon 參數。使用 AWS CLI 執行 stop-pipeline-execution 命令，並指定管道名稱和執行 ID。

  例如，若要停止名為 *MyFirstPipeline* 的管道、指定捨棄選項，並選擇包含註解：

  ```
  aws codepipeline stop-pipeline-execution --pipeline-name MyFirstPipeline --
  pipeline-execution-id d-EXAMPLE --abandon --reason "Stopping execution for a bug
   fix"
  ```

停止輸入執行 (CLI)

您可以使用 CLI 停止輸入執行。輸入執行是一種管線執行，正在等待進入停用轉換的階段。啟用轉換時，會InProgress繼續進入階段的輸入執行。未進入階段Stopped的輸入執行。

> **Note**
>
> 入站執行停止後，無法重試。

如果您沒有看到輸入執行，則在停用的階段轉換中沒有擱置的執行。

若要使用AWS CLI手動停止輸入執行，請使用具有下列參數的stop-pipeline-execution命令：

- 輸入執行 ID (必要)

- 註解 (選擇性)
- 管道名稱 (必要)
- 捨棄旗標 (選擇性，預設值為 false)

命令格式：

```
aws codepipeline stop-pipeline-execution --pipeline-name Pipeline_Name --
pipeline-execution-id Inbound_Execution_ID [--abandon | --no-abandon] [--
reason STOP_EXECUTION_REASON]
```

按照上述程序中的步驟輸入命令並指定 stop 方法。

使用get-pipeline-state命令檢視輸入執行的狀態。

在中建立管線 CodePipeline

您可以使用 AWS CodePipeline 控制台或 AWS CLI 建立管線。管道必須擁有至少兩個階段。管道的第一階段必須是來源階段。管道必須至少有一個其他階段是建置或部署階段。

您可以將動作新增至管道位於與管道不同的 AWS 區域中的管道。「跨區域」動作是指動作的提供者，而動作類型或提供者類型位於與管線不同的「 AWS 區域」中。 AWS 服務 如需詳細資訊，請參閱 在 CodePipeline 中新增跨區域動作。

您也可以使用 Amazon ECS 做為部署供應商，建立管道來建置和部署容器型應用程式。在建立使用 Amazon ECS 部署容器型應用程式的管道之前，您必須建立映像定義檔，如中所述。映像定義檔案參考

CodePipeline 推送原始程式碼變更時，會使用變更偵測方法來啟動管線。這些偵測方法視原始碼類型而定：

- CodePipeline 使用 Amazon CloudWatch 事件偵測 CodeCommit 來源儲存庫和分支或 S3 來源儲存貯體中的變更。

> **Note**
>
> 當您使用主控台建立或編輯管道時，將為您建立變更偵測資源。如果您使用 AWS CLI 建立管線，則必須自行建立其他資源。如需詳細資訊，請參閱 CodeCommit 來源動作和 EventBridge。

主題

建立管道 (主控台)

若要在主控台中建立管道，您必須提供原始檔案位置，以及您將用於動作的提供者相關資訊。

當您使用主控台來建立管道時，必須加入一個原始碼階段以及下方其中之一或兩者：

- 建置階段。
- 部署階段。

當您使用管線精靈時， CodePipeline 會建立階段的名稱 (來源、組建、暫存)。這些名稱無法變更。您可以在稍後新增的階段中使用更具體的名稱 (例如， BuildToGamma 或 DeployToProd)。

步驟 1：建立並命名管道

1. 請登入 AWS Management Console 並開啟 CodePipeline 主控台，網址為 http://console.aws.amazon.com/codesuite/codepipeline/home。
2. 在 Welcome (歡迎使用) 頁面上，選擇 Create pipeline (建立管道)。

   如果這是您第一次使用 CodePipeline，請選擇「開始使用」。
3. 在 Step 1: Choose pipeline settings (步驟 1：選擇管道設定) 頁面的 Pipeline name (管道名稱) 中輸入管道的名稱。

在單一 AWS 帳戶中，您在「AWS 區域」中建立的每個管線都必須具有唯一的名稱。名稱可以重複用於不同區域中的管道。

> **Note**
>
> 在您建立管道後，便無法更改其名稱。如需其他限制的相關資訊，請參閱 AWS CodePipeline 中的配額。

4. 在配管類型中，選擇下列其中一個選項。管道類型的特性和價格不同。如需詳細資訊，請參閱 管線類型。

   - V1 類型管道具有包含標準管道、階段和動作層級參數的 JSON 結構。
   - V2 類型管線與 V1 類型具有相同的結構，以及額外的參數支援，例如 Git 標籤上的觸發程序和管線層級變數。

5. 在 Service role (服務角色) 中，執行下列其中一項作業：

   - 選擇 [新增服務角色] CodePipeline 以允許在 IAM 中建立新的服務角色。
   - 選擇 Existing service role (現有服務角色) 以使用已在 IAM 中建立的服務角色。在 Role ARN (角色 ARN) 中，從清單選擇您的服務角色 ARN。

> **Note**
>
> 視建立服務角色的時間而定，您可能需要更新其權限以支援其他權限 AWS 服務。如需相關資訊，請參閱 將許可新增至 CodePipeline 服務角色。

如需服務角色與其政策陳述式的詳細資訊，請參閱 管理服 CodePipeline 務角色。

6. (選擇性) 在變數下，選擇新增變數以在管線層級新增變數。

如需管線層級變數的詳細資訊，請參閱Variables。如需在管線執行時傳送的管線層級變數的教學課程，請參閱。教學課程：使用管線層級變數

> **Note**
>
> 雖然在管線層級新增變數是可選的，但對於在沒有提供值的管線層級使用變數指定的配管，管線執行將會失敗。

7. (選用) 展開 Advanced settings (進階設定)。
8. 在 Artifact store (成品存放區) 中，執行下列其中一項操作：

   a. 選擇預設位置以針對您為管道選取的管道使用預設成品存放區，例如指定為預設值的 AWS 區域 S3 成品儲存貯體。

   b. 在與您的管道相同的區域中，若您已有成品存放區 (例如 S3 成品儲存貯體)，請選擇 Custom location (自訂位置)。在 Bucket (儲存貯體) 中，選擇儲存貯體名稱。

> **Note**
>
> 這不是原始碼的來源儲存貯體。這是管道的成品存放區。每個管道都需要有個別成品存放區，例如 S3 儲存貯體。建立或編輯管道時，管線區域中必須有一個人工因素儲存貯體，而每個執行動作的「 AWS 區域」都必須有一個成品儲存貯體。
> 如需詳細資訊，請參閱 輸入和輸出成品 及 CodePipeline 配管結構參照。

9. 在 Encryption key (加密金鑰) 中，執行下列其中一項操作：

   a. 若要使用 CodePipeline 預設值 AWS KMS key 來加密管道人工因素存放區 (S3 儲存貯體) 中的資料，請選擇預設 AWS 受管金鑰。

   b. 若要使用客戶受管金鑰加密管道成品存放區 (S3 儲存貯體) 中的資料，請選擇客戶受管金鑰。選擇金鑰識別碼、金鑰 ARN 或別名 ARN。

10. 選擇下一步。

步驟 2：建立原始碼階段

- 在 Step 2: Add source stage (步驟 2：新增來源階段) 頁面的 Source provider (來源供應商) 下拉式選單中，選擇您的原始碼所存放的儲存庫類型、指定其必要選項，然後選擇 Next step (下一步)。
  - 對於比特桶雲 GitHub（版本 2），GitHub 企業服務器，GitLab.com 或 GitLab自我管理：
    1. 在「連線」下，選擇現有的連線或建立新的連線。若要建立或管理來 GitHub 源動作的連線，請參閱GitHub 連接。
    2. 選擇您要用作管線來源位置的存放庫。

選擇在觸發器類型上新增觸發器或篩選器，以啟動管線。如需使用觸發器的詳細資訊，請參閱篩選程式碼推送或提取要求的觸發程序。如需使用 glob 模式篩選的詳細資訊，請參閱在語法中使用 glob 模式。

3. 在 [輸出成品格式] 中，選擇成品的格式。
   - 若要使用預設方法儲存 GitHub 動作的輸出成品，請選擇CodePipeline預設值。此動作會從存 GitHub 放庫存取檔案，並將人工因素儲存在管線人工因素存放區中的 ZIP 檔案中。
   - 若要存放包含儲存庫 URL 參考的 JSON 檔案，以便下游動作可以直接執行 Git 命令，請選擇 Full clone (完整複製)。此選項只能由 CodeBuild 下游動作使用。

     如果您選擇此選項，則需要更新 CodeBuild 專案服務角色的權限，如中所示疑難排 CodePipeline。如需說明如何使用「完整複製」選項的教學課程，請參閱教學課程：搭 GitHub 配管線來源使用完整複製。

- 對於 Amazon S3：

1. 在 Amazon S3 location (Amazon S3 位置) 中，提供 S3 儲存貯體名稱與連結到儲存貯體中物件的路徑，並啟用版本控制。儲存貯體名稱與路徑的格式類似下列內容：

```
s3://bucketName/folderName/objectName
```

> **Note**
>
> 當 Amazon S3 是管道的來源供應商時，您可以將一或多個來源檔案壓縮為單一 .zip，然後將 .zip 上傳到來源儲存貯體。您也可以上傳單一解壓縮檔案；不過，預期 .zip 檔案的下游動作會失敗。

2. 選擇 S3 來源儲存貯體後， CodePipeline 建立 Amazon CloudWatch 事件規則和要為此管道建立的 AWS CloudTrail 追蹤。接受 Change detection options (變更偵測選項) 下的預設設定： 這可 CodePipeline 讓您使用 Amazon CloudWatch 事件 AWS CloudTrail 並偵測新管道的變更。選擇下一步。

- 在AWS CodeCommit中：
  - 在存放庫名稱中，選擇要用作管線來源位置的 CodeCommit 存放庫名稱。在 Branch name (分支名稱)，從下拉式清單中選擇您想要使用的分支。
  - 在 [輸出成品格式] 中，選擇成品的格式。

- 若要使用預設方法儲存 CodeCommit 動作的輸出成品，請選擇CodePipeline預設值。此動作會從存 CodeCommit 放庫存取檔案，並將人工因素儲存在管線人工因素存放區中的 ZIP 檔案中。
- 若要存放包含儲存庫 URL 參考的 JSON 檔案，以便下游動作可以直接執行 Git 命令，請選擇 Full clone (完整複製)。此選項只能由 CodeBuild 下游動作使用。

  如果您選擇此選項，則需要將`codecommit:GitPull`權限新增至您的 CodeBuild 服務角色，如中所示新增 CodeCommit來源動作的 CodeBuild GitClone 權限。您還需要將`codecommit:GetRepository`權限添加到 CodePipeline 服務角色，如中所示將許可新增至 CodePipeline 服務角色。如需說明如何使用「完整複製」選項的教學課程，請參閱教學課程：搭 GitHub 配管線來源使用完整複製。

- 選擇 CodeCommit 儲存庫名稱和分支後，變更偵測選項中會顯示一則訊息，顯示要為此管道建立的 Amazon E CloudWatch vents 規則。接受 Change detection options (變更偵測選項) 下的預設設定： 這可 CodePipeline 讓您使用 Amazon CloudWatch 事件偵測新管道的變更。

- 對於 Amazon ECR：
  - 在儲存庫名稱中，選擇 Amazon ECR 儲存庫的名稱。
  - 在 Image tag (映像標籤) 中，指定映像名稱和版本，如果與最新不同的話。
  - 在輸出人工因素中，選擇輸出人工因素預設值 MyApp，例如，其中包含您要下一個階段使用的影像名稱和存放庫 URI 資訊。

    如需使用 CodeDeploy藍綠色部署 (包括 Amazon ECR 來源階段) 為 Amazon ECS 建立管道的教學課程，請參閱。教學課程：使用 Amazon ECR 來源和 ECS 到部署建立管道 CodeDeploy

  當您在管道中包含 Amazon ECR 來源階段時，當您提交變更時，來源動作會產生`imageDetail.json`檔案做為輸出成品。如需 `imageDetail.json` 詳細資訊，請參閱 適用於 Amazon ECS 藍色/綠色部署動作的 imageDetail.json 檔案。

> **Note**
>
> 物件和檔案類型必須與您計劃使用的部署系統相容 (例如，Elastic Beanstalk 或 CodeDeploy)。支援的檔案類型必須包含 .zip、.tar 以及 .tgz 檔案。如需 Elastic Beanstalk 支援容器類型的詳細資訊，請參閱自訂和設定 Elastic Beanstalk 環境和支援的平台。如需使用部署修訂版本的詳細資訊 CodeDeploy，請參閱上傳應用程式修訂版本和準備修訂版本。

步驟 3：建立建置階段

如果您打算建立部署階段，此步驟為選用。

- 在 Step 3: Add build stage (步驟 3：新增建置階段) 頁面上，執行下列其中一項動作，然後選擇 Next (下一步)：
  - 如果您打算建立部署階段，請選擇 Skip build stage (跳過建置階段)。
  - 從 Build provider (建置提供者) 中選擇建置服務的自訂動作提供者，並提供該提供者的組態詳細資訊。如需如何將 Jenkins 新增為建置提供者的範例，請參閱 教學：建立四階段管道。
  - 從 Build provider (建置供應商)，選擇 AWS CodeBuild。

    在區域中，選擇資源所在的 AWS 區域。[Re gion] 欄位會指定針對此動作類型和提供者類型建立 AWS 資源的位置。只有動作提供者為的動作才會顯示此欄位 AWS 服務。「區域」(Region) 欄位預設為與管線相同的「 AWS 區域」。

    在 Project name (專案名稱) 中，選擇您的建置專案。如果您已在中建立組建專案 CodeBuild，請選擇它。或者，您可以在中創建一個構建項目， CodeBuild 然後返回到此任務。請遵循「使用者指南」中的「建立使用的管線」 CodeBuild 中的CodeBuild指示進行。

    在環境變數中，若要將 CodeBuild環境變數新增至您的建置動作，請選擇 [新增環境變數]。每個變數都是由三個項目組成：
    - 在 Name (名稱) 中輸入環境變數的名稱或索引鍵。
    - 在 Value (值) 中輸入環境變數的值。如果您為變數類型選擇「參數」，請確定此值是您已儲存在「 AWS Systems Manager 參數存放區」中的參數名稱。

      > **Note**
      >
      > 我們強烈建議使用環境變數來儲存敏感值，尤其是 AWS 憑證。使用 CodeBuild 主控台或 AWS CLI 時，環境變數會以純文字顯示。對於敏感值，建議您改用 Parameter (參數) 類型。

    - (選擇性) 在 Type (類型) 中輸入環境變數的類型。有效值為 Plaintext (純文字) 或 Parameter (參數)。預設值為 Plaintext (純文字)。

    (選擇性) 在 [建置類型] 中，選擇下列其中一項：
    - 若要在單一建置動作執行中執行每個組建，請選擇 [單一組建]。
    - 若要在同一個建置動作執行中執行多個組建，請選擇「Batch 建置」。

(選擇性) 如果您選擇執行批次組建，您可以選擇將批次中的所有人工因素合併到單一位置，將所有組建成品置入單一輸出成品中。

步驟 4：建立部署階段

如果您已建立建置階段，此步驟為選用。

- 在 Step 4: Add deploy stage (步驟 4：新增部署階段) 頁面上，執行下列其中一項動作，然後選擇 Next (下一步)：
  - 如果您已在前一個步驟中建立建置階段，選擇 Skip deploy stage (跳過部署階段)。

    > **Note**
    >
    > 如果您已跳過建置階段，此選項不會出現。

  - 在 Deploy provider (部署提供者) 中，選擇您已為部署提供者建立的自訂動作。

    在「區域」中，僅針對「跨區域」作業，選擇要在其中建立資源的「 AWS 區域」。[Re gion] 欄位會指定針對此動作類型和提供者類型建立 AWS 資源的位置。此欄位只會針對動作提供者為的動作顯示 AWS 服務。「區域」(Region) 欄位預設為與管線相同的「 AWS 區域」。
  - 在 Deploy provider (部署供應商) 中，可用於預設供應商的欄位如下所示：
    - CodeDeploy

      在應用程式名稱中，輸入或選擇現有 CodeDeploy 應用程式的名稱。在 Deployment group (部署群組) 中，輸入該應用程式的部署群組名稱。選擇下一步。您也可以在 CodeDeploy 主控台中建立應用程式、部署群組或兩者。
    - AWS Elastic Beanstalk

      在應用程式名稱中，輸入或選擇現有 Elastic Beanstalk 應用程式的名稱。在 Environment name (環境名稱) 中，輸入應用程式的環境。選擇下一步。您也可以在 Elastic Beanstalk 主控台中建立應用程式、環境或兩者。
    - AWS OpsWorks Stacks

      在 Stack (堆疊) 中，輸入或選擇您想要使用的堆疊名稱。在 Layer (分層) 中，選擇您的目標執行個體隸屬的分層。在 App (應用程式) 中，選擇您想要更新與部署的應用程式。如果您需要建立應用程式，請在中選擇 [建立新應用程式] AWS OpsWorks。

如需將應用程式新增至堆疊和圖層的詳細資訊 AWS OpsWorks，請參閱《AWS OpsWorks 使用手冊》中的「新增應用程式」。

如需如何使用中的簡單管線 CodePipeline 做為在 AWS OpsWorks 圖層上執行之程式碼原始碼的 end-to-end 範例，請參閱使 CodePipeline 用 and AWS OpsWorks Stacks。

- AWS CloudFormation

  執行以下任意一項：

  - 在 [動作] 模式中，選擇 [建立或更新堆疊]、輸入堆疊名稱和範本檔案名稱，然後選擇 AWS CloudFormation 要承擔的角色名稱。選擇性地輸入組態檔案的名稱，然後選擇 IAM 功能選項。
  - 在「動作」模式中，選擇「建立或取代變更集」，輸入堆疊名稱和變更集名稱，然後選擇 AWS CloudFormation 要承擔的角色名稱。選擇性地輸入組態檔案的名稱，然後選擇 IAM 功能選項。

  有關將 AWS CloudFormation 功能整合到中的管道的詳細資訊 CodePipeline，請參閱AWS CloudFormation 使用者指南 CodePipeline中的持續交付。

- Amazon ECS

  在叢集名稱中，輸入或選擇現有 Amazon ECS 叢集的名稱。在 Service name (服務名稱) 中，輸入或選擇在叢集上執行的服務名稱。您也可以建立叢集和服務。在 Image filename (映像檔案名稱) 中，輸入說明服務容器與映像之映像定義檔案的名稱。

  > **Note**
  >
  > Amazon ECS 部署動作需要一個`imagedefinitions.json`檔案作為部署動作的輸入。檔案預設名稱為 imagedefinitions.json。如果您選用不同的名稱，必須在建立管道部署階段時提供名稱。如需詳細資訊，請參閱 適用於 Amazon ECS 標準部署動作的 imagedefinitions.json 檔案。

選擇下一步。

> **Note**
>
> 請確定您的 Amazon ECS 叢集已設定有兩個或更多個執行個體。Amazon ECS 叢集必須至少包含兩個執行個體，以便將一個作為主執行個體進行維護，另一個則用於容納新部署。

如需關於使用管道部署以容器為基礎的應用程式教學，請參閱教學：使用 CodePipeline 進行持續部署。

- Amazon ECS (Blue/Green) (Amazon ECS (藍/綠))

  輸入 CodeDeploy 應用程式和部署群組、Amazon ECS 任務定義和 AppSpec 檔案資訊，然後選擇 [下一步]。

  > **Note**
  >
  > Amazon ECS (Blue/Green) 動作需要 imageDetail.json 檔案做為部署動作的出入成品。由於 Amazon ECR 來源動作會建立此檔案，因此使用 Amazon ECR 來源動作的管道不需要提供檔案。`imageDetail.json`如需詳細資訊，請參閱 適用於 Amazon ECS 藍色/綠色部署動作的 imageDetail.json 檔案。

  如需使用建立管道以將藍綠色部署到 Amazon ECS 叢集的教學課程 CodeDeploy，請參閱。教學課程：使用 Amazon ECR 來源和 ECS 到部署建立管道 CodeDeploy

- AWS Service Catalog

  如果您想要使用主控台中的欄位來指定您的組態，請選擇 Enter deployment configuration (輸入部署組態)，或如果您擁有另一個組態檔案，請選擇 Configuration file (組態檔案)。輸入產品和組態資訊，然後選擇 Next (下一步)。

  如需使用管線將產品變更部署至 Service Catalog 的教學課程，請參閱教學課程：建立部署至 Service Catalog 的管線。

- Alexa Skills Kit

  在 Alexa Skill ID (Alexa 技能 ID) 中，輸入您 Alexa 技能的技能 ID。在 Client ID (用戶端 ID) 和 Client secret (用戶端密碼) 中，輸入使用 Login with Amazon (LWA) 安全性描述檔產生的

登入資料。在 Refresh token (重新整理字符) 中， 輸入您使用 ASK CLI 命令 (用於擷取重新整理字符) 產生的重新整理字符。選擇下一步。

如需有關使用您的管道部署 Alexa 技能和產生 LWA 登入資料的教學課程，請參閱教學：建立部署 Amazon Alexa 技能的管道。

- Amazon Simple Storage Service (Amazon S3)

  在 Bucket (儲存貯體) 中，輸入您要使用的 S3 儲存貯體名稱。如果您部署階段的輸入成品是 ZIP 檔案，請選擇 Extract file before deploy (部署前解壓縮檔案)。如果選取 Extract file before deploy (部署前解壓縮檔案)，您可以選擇 ZIP 檔案將要解壓縮至的 Deployment path (部署路徑)。如果未選取此選項，您需要在 S3 object key (S3 物件金鑰) 中輸入一個值。

  > **Note**
  >
  > 大多數來源和建置階段輸出成品都將被壓縮。Amazon S3 以外的所有管道來源供應商都會壓縮您的來源檔案，然後再將它們提供為下一個動作的輸入成品。

  (選擇性) 在固定 ACL 中，輸入要套用至部署到 Amazon S3 之物件的固定 ACL。

  > **Note**
  >
  > 套用固定的 ACL 時會覆寫任何套用到物件的現有 ACL。

  (選用) 在 Cache control (快取控制) 中，為從儲存貯體下載物件的請求指定快取控制參數。如需有效值的清單，請參閱 HTTP 操作的 `Cache-Control` 標頭欄位。若要在 Cache control (快取控制) 中輸入多個值，請在各值之間使用逗號。如此範例所示，您可以在每個逗號後面加上空格 (選用)。

  前面的範例項目會顯示在 CLI 中，如下所示：

  ```
  "CacheControl": "public, max-age=0, no-transform"
  ```

  選擇下一步。

如需使用 Amazon S3 部署動作提供者建立管道的教學課程，請參閱教學課程：建立使用 Amazon S3 做為部署供應商的管道。

步驟 5：檢閱管道

- 在 Step 5: Review (步驟 5：檢閱) 頁面上，檢閱您的管道組態，然後選擇 Create pipeline (建立管道) 以建立管道或 Previous (上一步) 以返回並編輯您的選擇。請選擇 Cancel (取消) 以放棄建立管道並離開精靈。

您現在已建立管道，並可在主控台中檢視。管道會在您建立後即開始執行。如需詳細資訊，請參閱 在中檢視管線和詳細資訊 CodePipeline。如需有關變更管道的詳細資訊，請參閱 在中編輯配管 CodePipeline。

建立管道 (CLI)

若要使用 AWS CLI 建立管線，請建立 JSON 檔案來定義管線結構，然後使用`--cli-input-json`參數執行create-pipeline命令。

> **Important**
>
> 您無法使用建 AWS CLI 立包含合作夥伴動作的管道。您必須改用 CodePipeline 主控台。

如需管線結構的詳細資訊，請參閱 API 參考中的CodePipeline 配管結構參照並建立管線。 CodePipeline

如要建立 JSON 檔案，請使用範本管道 JSON 檔案並進行編輯，然後在執行 create-pipeline 命令時呼叫該檔案。

先決條件：

您需要在 CodePipeline 中開始使用 CodePipeline建立的服務角色的 ARN。當您執行create-pipeline命令時，您可以在管線 JSON 檔案中使用 CodePipeline 服務角色 ARN。如需建立服務角色的詳細資訊，請參閱建立 CodePipeline 服務角色。與主控台不同的是，在中執行create-pipeline命令 AWS CLI 並沒有為您建立 CodePipeline 服務角色的選項。服務角色必須已存在。

您需要存放管道成品之 S3 儲存貯體的名稱。此儲存貯體必須與管道位於相同的區域。當執行 create-pipeline 命令時，您在管道 JSON 檔案中使用儲存貯體名稱。與主控台不同，在中執行create-pipeline命令 AWS CLI 並不會建立用於存放成品的 S3 儲存貯體。該儲存貯體必須已經存在。

> **Note**
>
> 您也可使用 get-pipeline 命令以取得該管道的 JSON 結構複本，然後以純文字編輯器修改該結構。

建立 JSON 檔案

1. 在終端機 (Linux、macOS 或 Unix) 或命令提示字元 (Windows) 上，在本機目錄中建立新的文字檔案。
2. (選用) 您可以在配管層級新增一或多個變數。您可以在 CodePipeline 動作的組態中參照此值。您可以在建立配管時新增變數名稱和值，也可以選擇在主控台中啟動管線時指派值。

   > **Note**
   >
   > 雖然在管線層級新增變數是可選的，但對於在沒有提供值的管線層級使用變數指定的配管，管線執行將會失敗。

   管線層級的變數會在管線執行時解析。所有變量都是不可變的，這意味著在賦值後它們無法更新。具有已解析值的管線層級變數將顯示在每次執行的歷史記錄中。

   您可以使用配管結構中的變數屬性在管線層級提供變數。在下列範例中，`Variable1`變數的值為`Value1`。

   ```
   "variables": [
        {
            "name": "Timeout",
            "defaultValue": "1000",
            "description": "description"
        }
    ]
   ```

   將此結構新增至管線 JSON，或在下一個步驟中新增至範例 JSON。如需有關變數的詳細資訊，包括命名空間資訊，請參閱Variables。
3. 在純文字編輯器中開啟檔案並編輯值，以反映您想建立的結構。您必須至少變更管道名稱。您也應考慮是否變更：

   - 此管道成品要存放的 S3 儲存貯體。

- 您程式碼的來源位置。
- 部署供應商。
- 您想如何部署程式碼？
- 您的管道標籤。

以下兩個階段的範本管道結構，反白了您應考慮變更的管道值。您的管道可能包含超過兩個以上的階段：

```
{
    "pipeline": {
        "roleArn": "arn:aws:iam::80398EXAMPLE::role/AWS-CodePipeline-Service",
        "stages": [
            {
                "name": "Source",
                "actions": [
                    {
                        "inputArtifacts": [],
                        "name": "Source",
                        "actionTypeId": {
                            "category": "Source",
                            "owner": "AWS",
                            "version": "1",
                            "provider": "S3"
                        },
                        "outputArtifacts": [
                            {
                                "name": "MyApp"
                            }
                        ],
                        "configuration": {
                            "S3Bucket": "awscodepipeline-demobucket-example-date",
                            "S3ObjectKey": "ExampleCodePipelineSampleBundle.zip",
                            "PollForSourceChanges": "false"
                        },
                        "runOrder": 1
                    }
                ]
            },
            {
                "name": "Staging",
                "actions": [
```

```
                {
                    "inputArtifacts": [
                        {
                            "name": "MyApp"
                        }
                    ],
                    "name": "Deploy-CodeDeploy-Application",
                    "actionTypeId": {
                        "category": "Deploy",
                        "owner": "AWS",
                        "version": "1",
                        "provider": "CodeDeploy"
                    },
                    "outputArtifacts": [],
                    "configuration": {
                        "ApplicationName": "CodePipelineDemoApplication",
                        "DeploymentGroupName": "CodePipelineDemoFleet"
                    },
                    "runOrder": 1
                }
            ]
        }
    ],
    "artifactStore": {
        "type": "S3",
        "location": "codepipeline-us-east-2-250656481468"
    },
    "name": "MyFirstPipeline",
    "version": 1,
    "variables": [
        {
            "name": "Timeout",
            "defaultValue": "1000",
            "description": "description"
        }
      ]
    },
    "triggers": [
        {
            "providerType": "CodeStarSourceConnection",
            "gitConfiguration": {
                "sourceActionName": "Source",
                "push": [
                    {
```

```
                            "tags": {
                                "includes": [
                                    "v1"
                                ],
                                "excludes": [
                                    "v2"
                                ]
                            }
                        }
                    ]
                }
            }
        ]
    "metadata": {
        "pipelineArn": "arn:aws:codepipeline:us-
east-2:80398EXAMPLE:MyFirstPipeline",
        "updated": 1501626591.112,
        "created": 1501626591.112
    },
    "tags": [{
      "key": "Project",
      "value": "ProjectA"
    }]
}
```

此範例透過將管道的 `Project` 標籤鍵和 `ProjectA` 值，新增標籤到管道。如需有關在中標記資源的更多資訊 CodePipeline，請參閱[標記 資源]。

請確認您 JSON 檔案中的 `PollForSourceChanges` 參數已如下所示進行設定：

```
                "PollForSourceChanges": "false",
```

CodePipeline 使用 Amazon CloudWatch 事件偵測 CodeCommit 來源儲存庫和分支或 S3 來源儲存貯體中的變更。下一步驟包含了為您的管道手動建立這些資源的說明。在您使用建議的變更偵測方法時，因為無須使用定期檢查，故可將旗標設為 `false` 以將其停用。

4. 若要在與您的管道不同的區域中建立建置、測試或部署動作，您必須將以下項目新增到管道結構。如需說明，請參閱[在 CodePipeline 中新增跨區域動作]。

   - 將 `Region` 參數新增到動作的管道結構。
   - 使用此`artifactStores`參數可為您執行動作的每個「 AWS 區域」指定成品值區。

5. 當您對其結構滿意時，使用像是 **pipeline.json** 的名稱儲存您的檔案。

建立管道

1. 執行 create-pipeline 命令，並用 `--cli-input-json` 參數以指定您之前建立的 JSON 檔案。

   若要建立以名*MySecondPipeline*為「管線 .json」的 JSON 檔案命名的管線，該檔案包含名稱為 *MySecondPipeline*"" 作為 JSON name 中的值，您的命令看起來會如下所示：

   ```
   aws codepipeline create-pipeline --cli-input-json file://pipeline.json
   ```

   > **Important**
   >
   > 請確認在檔案名稱之前包含 `file://`。這是此命令必要項目。

   此命令會傳回您所建立的所有管道結構。

2. 若要檢視管線，請開啟 CodePipeline 控制台並從管線清單中選擇它，或使用get-pipeline-state指令。如需詳細資訊，請參閱 在中檢視管線和詳細資訊 CodePipeline。

3. 若您使用 CLI 來建立管道，您必須為管道手動建立建議的變更偵測資源：

   - 對於具有 CodeCommit 存放庫的管線，您必須手動建立 CloudWatch 事件規則，如中所述建立 CodeCommit 來源 (CLI) 的 EventBridge 規則。
   - 對於具有 Amazon S3 來源的管道，您必須手動建立 CloudWatch 事件規則和 AWS CloudTrail 追蹤，如中所述Amazon S3 來源動作 EventBridge 與 AWS CloudTrail。

Amazon ECR 來源動作和 EventBridge 資源

若要在中新增 Amazon ECR 來源動作 CodePipeline，您可以選擇以下任一項作業：

- 使用主 CodePipeline 控台建立管道精靈 (建立管道 (主控台)) 或編輯動作頁面來選擇 Amazon ECR 提供者選項。控制台創建一個 EventBridge 規則，該規則在源更改時啟動管道。
- 使用 CLI 新增動作的動作組態，並建立其他資源，如下所示：ECR
  - 使用中的ECR範例動作配置Amazon ECR來建立您的動作，如中所示建立管道 (CLI)。
  - 變更偵測方法預設為透過輪詢來源來啟動管線。您應該停用定期檢查，並手動建立變更偵測規則。使用下列其中一種方法：為 Amazon ECR 來源 (主控台) 建立 EventBridge 規則為 Amazon ECR

來源 (CLI) 建立 EventBridge 規則、或為 Amazon ECR 來源 (AWS CloudFormation 範本) 建立 EventBridge 規則 。

主題

- 為 Amazon ECR 來源 (主控台) 建立 EventBridge 規則
- 為 Amazon ECR 來源 (CLI) 建立 EventBridge 規則
- 為 Amazon ECR 來源 (AWS CloudFormation 範本) 建立 EventBridge 規則

為 Amazon ECR 來源 (主控台) 建立 EventBridge 規則

若要建立在 CodePipeline 作業中使用的 EventBridge 規則 (Amazon ECR 來源)

1. 在以下位置打開 Amazon EventBridge 控制台 https://console.aws.amazon.com/events/。
2. 在導覽窗格中，選擇 Events (事件)。
3. 選擇 [建立規則]，然後在 [事件來源] 下的 [服務名稱] 中選擇 [彈性容器登錄 (ECR)]。
4. 在 Event Source (事件來源) 中，選擇 Event Pattern (事件模式)。

   選擇 Edit (編輯)，然後在 Event Source (事件來源) 視窗中為 `eb-test` 儲存庫貼上以下範例事件模式，並加上 `cli-testing` 的映像標籤：

```
{
    "detail-type": [
        "ECR Image Action"
    ],
    "source": [
        "aws.ecr"
    ],
    "detail": {
        "action-type": [
            "PUSH"
        ],
        "image-tag": [
            "latest"
        ],
        "repository-name": [
            "eb-test"
        ],
        "result": [
            "SUCCESS"
```

```
        ]
    }
}
```

> **Note**
>
> 若要檢視 Amazon ECR 事件支援的完整事件模式，請參閱 Amazon ECR 事件和 EventBridge或 Amazon 彈性容器登錄事件。

5. 選擇儲存。

   在 Event Pattern Preview (事件模式預覽) 窗格中，檢視規則。

6. 在 [目標] 中，選擇CodePipeline。

7. 輸入要由此規則啟動之管線的配管 ARN。

> **Note**
>
> 在您執行 get-pipeline 命令之後，即可在中繼資料輸出中找到管道 ARN。管道 ARN 是以下列格式建構：
>
> *AR: aws: #####:##:##:####*
>
> 範例管道 ARN：
>
> `arn:aws:codepipeline:us-east-2:80398EXAMPLE:MyFirstPipeline`

8. 建立或指定 IAM 服務角色，以授與呼叫 EventBridge 規則關聯之目標的 EventBridge 許可 (在此情況下，目標為 CodePipeline)。

   - 選擇 [為此特定資源建立新角色] 以建立服務角色，以提供啟動管線執行的 EventBridge權限的服務角色。
   - 選擇 [使用現有角色] 以輸入授與啟動管線執行之 EventBridge 權限的服務角色。

9. 檢閱您的規則設定以確定其符合您的要求。

10. 選擇 Configure details (設定詳細資訊)。

11. 在 Configure rule details (設定規則詳細資訊) 頁面上，輸入規則的名稱和描述，然後選擇 State (狀態) 啟用規則。

12. 如果您對此規則感到滿意，請選擇 Create rule (建立規則)。

為 Amazon ECR 來源 (CLI) 建立 EventBridge 規則

呼叫 put-rule 命令，並指定：

- 可唯一識別您所建立規則的名稱。在與 AWS 帳戶 CodePipeline 相關聯的所有管道中，此名稱必須是唯一的。
- 規則所使用來源和詳細資訊欄位的事件模式。如需詳細資訊，請參閱 Amazon EventBridge 和事件模式。

若要使用 Amazon ECR 做為事件來源並 CodePipeline 做為目標建立 EventBridge 規則

1. 新增用 EventBridge CodePipeline 於呼叫規則的權限。如需詳細資訊，請參閱在 Amazon EventBridge 使用以資源為基礎的政策。

   a. 使用下列範例來建立信任政策，以允許 EventBridge 擔任服務角色。將信任政策命名為 `trustpolicyforEB.json`。

   ```
   {
       "Version": "2012-10-17",
       "Statement": [
           {
               "Effect": "Allow",
               "Principal": {
                   "Service": "events.amazonaws.com"
               },
               "Action": "sts:AssumeRole"
           }
       ]
   }
   ```

   b. 使用下列命令來建立 `Role-for-MyRule` 角色，並連接信任政策。

   ```
   aws iam create-role --role-name Role-for-MyRule --assume-role-policy-document
    file://trustpolicyforEB.json
   ```

   c. 為名為 `MyFirstPipeline` 的管道建立許可政策 JSON，如這個範例所示。將許可政策命名為 `permissionspolicyforEB.json`。

   ```
   {
       "Version": "2012-10-17",
       "Statement": [
   ```

```
        {
            "Effect": "Allow",
            "Action": [
                "codepipeline:StartPipelineExecution"
            ],
            "Resource": [
                "arn:aws:codepipeline:us-west-2:80398EXAMPLE:MyFirstPipeline"
            ]
        }
    ]
}
```

d. 使用執行以下命令，將 CodePipeline-Permissions-Policy-for-EB 許可政策連接到 Role-for-MyRule 角色。

為什麼我會做出此變更？ 將此原則新增至角色會建立的權限 EventBridge。

```
aws iam put-role-policy --role-name Role-for-MyRule --policy-name CodePipeline-
Permissions-Policy-For-EB --policy-document file://permissionspolicyforEB.json
```

2. 呼叫 put-rule 命令，並包含 --name、--event-pattern 和 --role-arn 參數。

為什麼我會做出此變更？ 您必須使用規則來建立事件，該規則指定必須如何進行映像推送，以及命名要由事件啟動的管道的目標。

以下範例命令會建立名為 MyECRRepoRule 的規則。

```
aws events put-rule --name "MyECRRepoRule" --event-pattern "{\"detail-type\":[\"ECR
 Image Action\"],\"source\":[\"aws.ecr\"],\"detail\":{\"action-type\":[\"PUSH\"],
\"image-tag\":[\"latest\"],\"repository-name\":[\"eb-test\"],\"result\":[\"SUCCESS
\"]}}" --role-arn "arn:aws:iam::ACCOUNT_ID:role/Role-for-MyRule"
```

> **Note**
>
> 若要檢視 Amazon ECR 事件支援的完整事件模式，請參閱 Amazon ECR 事件和/EventBridge或 Amazon 彈性容器登錄事件。

3. 若要新增 CodePipeline 為目標，請呼叫指put-targets令並包含下列參數：

- --rule 參數與您使用 put-rule 所建立的 rule_name 搭配使用。
- --targets 參數與目標清單中目標的清單 Id 和目標管道的 ARN 搭配使用。

以下命令範例指定名為 MyECRRepoRule 的規則，該目標 Id 是由數字 1 組成，指出在規則的目標清單中，這是目標 1。這個範例命令也會指定管線範例 Arn，以及規則範例 RoleArn。儲存庫中若發生變更，管道就會啟動。

```
aws events put-targets --rule MyECRRepoRule --targets
 Id=1,Arn=arn:aws:codepipeline:us-
west-2:80398EXAMPLE:TestPipeline,RoleArn=arn:aws:iam::80398EXAMPLE:role/Role-for-
MyRule
```

為 Amazon ECR 來源 (AWS CloudFormation 範本) 建立 EventBridge 規則

若要用 AWS CloudFormation 來建立規則，請使用範本程式碼片段，如下所示。

更新管線 AWS CloudFormation 範本並建立 EventBridge 規則

1. 在範本中Resources，使用資AWS::IAM::Role AWS CloudFormation 源來設定允許事件啟動管道的 IAM 角色。此項目會建立一個使用兩個政策的角色：

   - 第一個政策允許要承擔的角色。
   - 第二個政策提供啟動管道的許可。

   為什麼我會做出此變更？ 您必須創建一個可以假定的角色，以 EventBridge 便在我們的管道中開始執行。

   YAML

   ```
   EventRole:
     Type: AWS::IAM::Role
     Properties:
       AssumeRolePolicyDocument:
         Version: 2012-10-17
         Statement:
           -
             Effect: Allow
             Principal:
               Service:
                 - events.amazonaws.com
             Action: sts:AssumeRole
   ```

```
      Path: /
      Policies:
        -
          PolicyName: eb-pipeline-execution
          PolicyDocument:
            Version: 2012-10-17
            Statement:
              -
                Effect: Allow
                Action: codepipeline:StartPipelineExecution
                Resource: !Sub arn:aws:codepipeline:${AWS::Region}:
${AWS::AccountId}:${AppPipeline}
```

JSON

```
{
    "EventRole": {
        "Type": "AWS::IAM::Role",
        "Properties": {
            "AssumeRolePolicyDocument": {
                "Version": "2012-10-17",
                "Statement": [
                    {
                        "Effect": "Allow",
                        "Principal": {
                            "Service": [
                                "events.amazonaws.com"
                            ]
                        },
                        "Action": "sts:AssumeRole"
                    }
                ]
            },
            "Path": "/",
            "Policies": [
                {
                    "PolicyName": "eb-pipeline-execution",
                    "PolicyDocument": {
                        "Version": "2012-10-17",
                        "Statement": [
                            {
                                "Effect": "Allow",
                                "Action": "codepipeline:StartPipelineExecution",
```

```
                                "Resource": {
                                    "Fn::Sub": "arn:aws:codepipeline:
${AWS::Region}:${AWS::AccountId}:${AppPipeline}"
                                }
                            }
                        ]
                    }
                }
            ]
        }
    }
}
...
```

2. 在範本的下Resources，使用AWS::Events::Rule AWS CloudFormation 資源為 Amazon ECR 來源新增 EventBridge 規則。此事件模式會建立一個事件，用於監視對儲存庫的提交。EventBridge 偵測到存放庫狀態變更時，規則會StartPipelineExecution在您的目標管線上叫用。

   我為什麼要進行這項變更？ 您必須使用規則來建立事件，該規則指定必須如何進行映像推送，以及命名要由事件啟動的管道的目標。

   此程式碼片段使用名為 eb-test 的映像，並具有 latest 標籤。

   YAML

```
EventRule:
  Type: 'AWS::Events::Rule'
  Properties:
    EventPattern:
      detail:
        action-type: [PUSH]
        image-tag: [latest]
        repository-name: [eb-test]
        result: [SUCCESS]
      detail-type: [ECR Image Action]
      source: [aws.ecr]
    Targets:
      - Arn: !Sub arn:aws:codepipeline:${AWS::Region}:${AWS::AccountId}:
${AppPipeline}
        RoleArn: !GetAtt
          - EventRole
```

```
          - Arn
        Id: codepipeline-AppPipeline
```

JSON

```
{
    "EventRule": {
        "Type": "AWS::Events::Rule",
        "Properties": {
            "EventPattern": {
                "detail": {
                    "action-type": [
                        "PUSH"
                    ],
                    "image-tag": [
                        "latest"
                    ],
                    "repository-name": [
                        "eb-test"
                    ],
                    "result": [
                        "SUCCESS"
                    ]
                },
                "detail-type": [
                    "ECR Image Action"
                ],
                "source": [
                    "aws.ecr"
                ]
            },
            "Targets": [
                {
                    "Arn": {
                        "Fn::Sub": "arn:aws:codepipeline:${AWS::Region}:
${AWS::AccountId}:${AppPipeline}"
                    },
                    "RoleArn": {
                        "Fn::GetAtt": [
                            "EventRole",
                            "Arn"
                        ]
                    },
```

```
                    "Id": "codepipeline-AppPipeline"
                }
            ]
        }
    }
},
```

> **Note**
>
> 若要檢視 Amazon ECR 事件支援的完整事件模式，請參閱 Amazon ECR 事件和 EventBridge或 Amazon 彈性容器登錄事件。

3. 將更新的範本儲存到本機電腦，然後開啟 AWS CloudFormation 主控台。
4. 選擇您的堆疊，然後選擇 Create Change Set for Current Stack (建立目前堆疊的變更集)。
5. 上傳範本，然後檢視 AWS CloudFormation中所列的變更。這些是會針對堆疊進行的變更。您應該會在清單中看到新資源。
6. 選擇 Execute (執行)。

Amazon S3 來源動作 EventBridge 與 AWS CloudTrail

若要在中新增 Amazon S3 來源動作 CodePipeline，您可以選擇以下任一項作業：

- 使用主 CodePipeline 控台建立管道精靈 (建立管道 (主控台)) 或編輯動作頁面來選擇 S3 提供者選項。主控台會建立 EventBridge 規則和 CloudTrail 追蹤，以便在來源變更時啟動管道。
- 使用新 AWS CLI 增動作的動作組態，並建立其他資源，如下所示：S3
  - 使用中的S3範例動作配置Amazon S3 來源動作來建立您的動作，如中所示建立管道 (CLI)。
  - 變更偵測方法預設為透過輪詢來源來啟動管線。您應該停用定期檢查，並手動建立變更偵測規則和追蹤。使用下列其中一種方法：為 Amazon S3 來源 (主控台) 建立 EventBridge 規則為 Amazon S3 來源 (CLI) 建立 EventBridge 規則、或為 Amazon S3 來源 (AWS CloudFormation 範本) 建立 EventBridge 規則 。

AWS CloudTrail 這是一項服務，用於記錄和篩選 Amazon S3 來源儲存貯體上的事件。追蹤會將已篩選的來源變更傳送至 EventBridge 規則。 EventBridge 規則會偵測來源變更，然後啟動管線。

使用要求：

- 如果您不建立追蹤，請使用現有 AWS CloudTrail 追蹤記錄 Amazon S3 來源儲存貯體中的事件，並將篩選的事件傳送至 EventBridge規則。
- 建立或使用現有的 S3 儲存貯體， AWS CloudTrail 以存放其日誌檔。 AWS CloudTrail 必須具有將日誌檔交付到 Amazon S3 儲存貯體所需的許可。此儲存貯體無法設定為申請者付款儲存貯體。當您在主控台中建立或更新追蹤時建立 Amazon S3 儲存貯體時，請為您 AWS CloudTrail 附加必要的許可到儲存貯體。如需詳細資訊，請參閱的 Amazon S3 儲存貯體政策 CloudTrail。

為 Amazon S3 來源 (主控台) 建立 EventBridge 規則

在中設置規則之前 EventBridge，您必須先建立 AWS CloudTrail 軌跡。如需詳細資訊，請參閱在主控台中建立線索。

> **Important**
>
> 如果您使用主控台建立或編輯管線， EventBridge 則會為您建立規則和 AWS CloudTrail 追蹤。

若要建立追蹤記錄

1. 開啟主 AWS CloudTrail 控台。
2. 在導覽窗格中，選擇 Trails (追蹤記錄)。
3. 選擇 Create trail (建立追蹤)。對於 Trail name (線索名稱)，輸入您的線索名稱。
4. 在 Storage location (儲存位置) 下，建立或指定要用來存放日誌檔案的儲存貯體。根據預設，所有 Amazon S3 儲存貯體和物件皆為私有。只有資源擁有者 (建立值區的 AWS 帳號) 可以存取值區及其物件。值區必須具有允許存取值區中物件的 AWS CloudTrail 權限的資源策略。
5. 在追蹤記錄儲存貯體和資料夾下，指定 Amazon S3 儲存貯體和物件前置詞 (資料夾名稱)，以記錄資料夾中所有物件的資料事件。對於每個追蹤，您最多可以新增 250 個 Amazon S3 物件。完成所需的加密金鑰資訊，然後選擇「下一步」
6. 針對事件類型，選擇管理事件。
7. 針對管理事件，選擇寫入。追蹤會在指定儲存貯體和前置詞上記錄 Amazon S3 物件層級 API 活動 (例如`GetObject`和`PutObject`)。
8. 選擇 Write (寫入)。
9. 如果您對路線感到滿意，請選擇「建立路線」。

若要使用 Amazon S3 來源建立以管道為目標的 EventBridge 規則

1. 在以下位置打開 Amazon EventBridge 控制台 https://console.aws.amazon.com/events/。
2. 在導覽窗格中，選擇規則。保持選取預設匯流排，或選擇活動匯流排。選擇建立規則。
3. 在名稱中，輸入規則的名稱。
4. 在「規則類型」下，選擇「具有事件模式的規則」。選擇下一步。
5. 在事件來源下，選擇AWS 事件或 EventBridge 合作夥伴事件。
6. 在 [範例事件類型] 下方，選擇AWS 事件。
7. 在範例事件中，輸入 S3 做為要篩選的關鍵字。通過選擇 AWS API 調用 CloudTrail。
8. 在 [建立方法] 下方，選擇 [客戶模式 (JSON 編輯器)]。

   粘貼下面提供的事件模式。請務必新增儲存貯體名稱和 S3 物件金鑰 (或金鑰名稱)，這些金鑰可唯一識別儲存貯體中的物件`requestParameters`。在此範例中，會為名為的值區`my-bucket`和物件索引鍵建立規則`my-files.zip`。當您使用 Edit (編輯) 視窗指定資源時，您的規則將更新為使用自訂事件模式。

   以下是用於複製貼上的範例事件模式：

```
{
    "source": [
        "aws.s3"
    ],
    "detail-type": [
        "AWS API Call via CloudTrail"
    ],
    "detail": {
        "eventSource": [
            "s3.amazonaws.com"
        ],
        "eventName": [
            "CopyObject",
            "CompleteMultipartUpload",
            "PutObject"
        ],
        "requestParameters": {
            "bucketName": [
                "my-bucket"
            ],
            "key": [
                "my-files.zip"
```

```
            ]
        }
    }
}
```

9. 選擇下一步。
10. 在目標類型中，選擇AWS 服務。
11. 在 [選取目標] 中，選擇CodePipeline。在管線 ARN 中，輸入管線 ARN，讓此規則啟動的管線。

> **Note**
>
> 若要取得管道 ARN，請執行 get-pipeline 命令。管道 ARN 會出現在輸出中。其建構格式如下：
>
> *AR: aws: #####:##:##:####*
>
> 範例管道 ARN：
>
> ARN：AW：代碼流水線：美國東部-2：80398 示例：MyFirstPipeline

12. 若要建立或指定 IAM 服務角色，以授與呼叫 EventBridge 規則關聯之目標的 EventBridge 權限 (在本例中，目標為 CodePipeline)：
    - 選擇 [為此特定資源建立新角色] 以建立服務角色，以提供啟動管線執行的 EventBridge權限的服務角色。
    - 選擇 [使用現有角色] 以輸入授與啟動管線執行之 EventBridge 權限的服務角色。
13. 選擇下一步。
14. 在 [標籤] 頁面上，選擇 [下一步]。
15. 在 [檢閱並建立] 頁面上，檢閱規則組態。如果您對此規則感到滿意，請選擇 Create rule (建立規則)。

為 Amazon S3 來源 (CLI) 建立 EventBridge 規則

若要建立 AWS CloudTrail 追蹤並啟用記錄

若要使用 AWS CLI 建立軌跡，請呼叫指create-trail令，並指定：

- 線索名稱。
- 您已套用 AWS CloudTrail 儲存貯體政策的儲存貯體。

若要取得更多資訊，請參閱〈使用 AWS 指令行介面建立系統線〉。

1. 呼叫 create-trail 命令，並包含 --name 和 --s3-bucket-name 參數。

   為什麼我會做出此變更？ 這會為您的 S3 來源儲存貯體建立所需的 CloudTrail 線索。

   以下命令使用 --name 和 --s3-bucket-name，來建立名為 my-trail 的線索，以及名為 myBucket 的儲存貯體。

   ```
   aws cloudtrail create-trail --name my-trail --s3-bucket-name myBucket
   ```

2. 呼叫 start-logging 命令並加入 --name 參數。

   我為什麼要進行這項變更？ 此命令會啟動來源值區的 CloudTrail 記錄，並將事件傳送至 EventBridge。

   範例：

   以下命令範例會使用了 --name，以在名為 my-trail 的線索上啟動日誌記錄。

   ```
   aws cloudtrail start-logging --name my-trail
   ```

3. 呼叫 put-event-selectors 命令，並包含 --trail-name 和 --event-selectors 參數。使用事件選取器來指定您希望追蹤記錄來源儲存貯體的資料事件，並將事件傳送至 EventBridge 規則。

   我為什麼要進行這項變更？ 此命令會篩選事件。

   範例：

   以下命令範例使用 --trail-name 與 --event-selectors，來指定來源儲存貯體和字首的資料事件，名為 myBucket/myFolder。

   ```
   aws cloudtrail put-event-selectors --trail-name my-trail --event-selectors
    '[{ "ReadWriteType": "WriteOnly", "IncludeManagementEvents":false,
    "DataResources": [{ "Type": "AWS::S3::Object", "Values": ["arn:aws:s3:::myBucket/
   myFolder/file.zip"] }] }]'
   ```

使用 Amazon S3 做為事件來源和目標建立 EventBridge 規則 CodePipeline ，並套用許可政策

1. 授與用 EventBridge CodePipeline 於呼叫規則的權限。如需詳細資訊，請參閱在 Amazon EventBridge 使用以資源為基礎的政策。

a. 使用下列範例建立信任原則，以 EventBridge 允許擔任服務角色。將其命名為 `trustpolicyforEB.json`。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "events.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

b. 使用下列命令來建立 `Role-for-MyRule` 角色，並連接信任政策。

為什麼我會做出此變更？ 將此信任原則新增至角色會建立的權限 EventBridge。

```
aws iam create-role --role-name Role-for-MyRule --assume-role-policy-document
 file://trustpolicyforEB.json
```

c. 針對名為 `MyFirstPipeline` 的管道建立許可政策 JSON，如下所示。將許可政策命名為 `permissionspolicyforEB.json`。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codepipeline:StartPipelineExecution"
            ],
            "Resource": [
                "arn:aws:codepipeline:us-west-2:80398EXAMPLE:MyFirstPipeline"
            ]
        }
    ]
}
```

d. 使用執行以下命令，將新的 CodePipeline-Permissions-Policy-for-EB 許可政策連接到您所建立的 Role-for-MyRule 角色。

```
aws iam put-role-policy --role-name Role-for-MyRule --policy-name CodePipeline-
Permissions-Policy-For-EB --policy-document file://permissionspolicyforEB.json
```

2. 呼叫 put-rule 命令，並包含 --name、--event-pattern 和 --role-arn 參數。

以下範例命令會建立名為 MyS3SourceRule 的規則。

```
aws events put-rule --name "MyS3SourceRule" --event-pattern "{\"source\":
[\"aws.s3\"],\"detail-type\":[\"AWS API Call via CloudTrail\"],\"detail\":
{\"eventSource\":[\"s3.amazonaws.com\"],\"eventName\":[\"CopyObject\",\"PutObject
\",\"CompleteMultipartUpload\"],\"requestParameters\":{\"bucketName\":[\"my-bucket
\"],\"key\":[\"my-key\"]}}}
 --role-arn "arn:aws:iam::ACCOUNT_ID:role/Role-for-MyRule"
```

3. 若要新增 CodePipeline 為目標，請呼叫put-targets指令並包含--rule和--targets參數。

以下命令指定名為 MyS3SourceRule 的規則，該目標 Id 是由數字 1 組成，指出在規則的目標清單中，這是目標 1。此命令也會指定管道的範例 ARN。儲存庫中若發生變更，管道就會啟動。

```
aws events put-targets --rule MyS3SourceRule --targets
 Id=1,Arn=arn:aws:codepipeline:us-west-2:80398EXAMPLE:TestPipeline
```

若要編輯管線的 PollForSourceChanges參數

> **Important**
>
> 當您使用這個方法建立管道時，如果沒有明確設為 false，則 PollForSourceChanges 參數會預設為 true。當新增基於事件的變更偵測時，您必須將該參數新增到輸出，並將其設為 false 以停用輪詢。否則，您的管道會針對單一來源變更啟動兩次。如需詳細資訊，請參閱 PollForSourceChanges 參數的預設設定。

1. 執行 get-pipeline 命令，將管道結構複製到 JSON 檔案。例如，針對名為 MyFirstPipeline 的管道，執行下列命令：

```
aws codepipeline get-pipeline --name MyFirstPipeline >pipeline.json
```

此命令不會傳回任何內容，但您建立的檔案應該會顯示在您執行命令的目錄中。

2. 在任何純文字編輯器中開啟 JSON 檔案，然後將名為 `storage-bucket` 之儲存貯體的 `PollForSourceChanges` 參數變更為 `false`，來編輯來源階段，如這個範例所示。

   為什麼我會做出此變更？ 將此參數設為 `false` 會關閉定期檢查，因此您只能使用事件型變更偵測。

   ```
   "configuration": {
       "S3Bucket": "storage-bucket",
       "PollForSourceChanges": "false",
       "S3ObjectKey": "index.zip"
   },
   ```

3. 如果您使用的是使用 get-pipeline 命令擷取的管道結構，則必須從 JSON 檔案中移除 `metadata` 行。否則，update-pipeline 命令無法使用它。移除 `"metadata": { }` 行，以及 `"created"`、`"pipelineARN"` 和 `"updated"` 欄位。

   例如，從結構中移除下列幾行：

   ```
   "metadata": {
       "pipelineArn": "arn:aws:codepipeline:region:account-ID:pipeline-name",
       "created": "date",
       "updated": "date"
   },
   ```

   儲存檔案。

4. 若要套用您的變更，請執行 update-pipeline 命令、指定管道 JSON 檔案：

   > **Important**
   >
   > 請確認在檔案名稱之前包含 `file://`。這是此命令必要項目。

   ```
   aws codepipeline update-pipeline --cli-input-json file://pipeline.json
   ```

   此命令會傳回所編輯管道的整個結構。

> **Note**
>
> update-pipeline 命令將終止管道。若在您執行 update-pipeline 命令時有修訂正在透過管道執行，該執行將停止。您必須手動啟動管道，以透過更新的管道執行該修訂。使用 start-pipeline-execution 命令來手動啟動您的管道。

為 Amazon S3 來源 (AWS CloudFormation 範本) 建立 EventBridge 規則

若要用 AWS CloudFormation 來建立規則，請更新範本，如下所示。

使用 Amazon S3 做為事件來源和目標建立 EventBridge 規則 CodePipeline ，並套用許可政策

1. 在範本中`Resources`，使用資`AWS::IAM::Role` AWS CloudFormation 源來設定允許事件啟動管道的 IAM 角色。此項目會建立一個使用兩個政策的角色：

   - 第一個政策允許要承擔的角色。
   - 第二個政策提供啟動管道的許可。

   為什麼我會做出此變更？ 新增`AWS::IAM::Role`資源可 AWS CloudFormation 建立的權限 EventBridge。此資源會新增至您的 AWS CloudFormation 堆疊。

   YAML

```
EventRole:
  Type: AWS::IAM::Role
  Properties:
    AssumeRolePolicyDocument:
      Version: 2012-10-17
      Statement:
        -
          Effect: Allow
          Principal:
            Service:
              - events.amazonaws.com
          Action: sts:AssumeRole
    Path: /
    Policies:
      -
        PolicyName: eb-pipeline-execution
```

```
          PolicyDocument:
            Version: 2012-10-17
            Statement:
              -
                Effect: Allow
                Action: codepipeline:StartPipelineExecution
                Resource: !Join [ '', [ 'arn:aws:codepipeline:', !Ref
 'AWS::Region', ':', !Ref 'AWS::AccountId', ':', !Ref AppPipeline ] ]


...
```

JSON

```
 "EventRole": {
   "Type": "AWS::IAM::Role",
   "Properties": {
     "AssumeRolePolicyDocument": {
       "Version": "2012-10-17",
       "Statement": [
         {
           "Effect": "Allow",
           "Principal": {
             "Service": [
               "events.amazonaws.com"
             ]
           },
           "Action": "sts:AssumeRole"
         }
       ]
     },
     "Path": "/",
     "Policies": [
       {
         "PolicyName": "eb-pipeline-execution",
         "PolicyDocument": {
           "Version": "2012-10-17",
           "Statement": [
             {
               "Effect": "Allow",
               "Action": "codepipeline:StartPipelineExecution",
               "Resource": {
                 "Fn::Join": [
```

```
                    "",
                    [
                      "arn:aws:codepipeline:",
                      {
                        "Ref": "AWS::Region"
                      },
                      ":",
                      {
                        "Ref": "AWS::AccountId"
                      },
                      ":",
                      {
                        "Ref": "AppPipeline"
                      }
                    ]
                  ]

...
```

2. 使用AWS::Events::Rule AWS CloudFormation 資源新增 EventBridge 規則。此事件模式會在您的 Amazon S3 來源儲存貯體CompleteMultipartUpload上建立監控CopyObject事件。PutObject此外，會包含您管道的目標。當 CopyObject、PutObject 或 CompleteMultipartUpload 發生時，此規則會在目標管道上呼叫 StartPipelineExecution。

   為什麼我會做出此變更？ 新增資AWS::Events::Rule源可 AWS CloudFormation 建立事件。此資源會新增至您的 AWS CloudFormation 堆疊。

   YAML

```
  EventRule:
    Type: AWS::Events::Rule
    Properties:
      EventPattern:
        source:
          - aws.s3
        detail-type:
          - 'AWS API Call via CloudTrail'
        detail:
          eventSource:
            - s3.amazonaws.com
          eventName:
            - CopyObject
```

```
            - PutObject
            - CompleteMultipartUpload
          requestParameters:
            bucketName:
              - !Ref SourceBucket
            key:
              - !Ref SourceObjectKey
      Targets:
        -
          Arn:
            !Join [ '', [ 'arn:aws:codepipeline:', !Ref 'AWS::Region', ':', !Ref
 'AWS::AccountId', ':', !Ref AppPipeline ] ]
          RoleArn: !GetAtt EventRole.Arn
          Id: codepipeline-AppPipeline


...
```

JSON

```
 "EventRule": {
   "Type": "AWS::Events::Rule",
   "Properties": {
     "EventPattern": {
       "source": [
         "aws.s3"
       ],
       "detail-type": [
         "AWS API Call via CloudTrail"
       ],
       "detail": {
         "eventSource": [
           "s3.amazonaws.com"
         ],
         "eventName": [
           "CopyObject",
           "PutObject",
           "CompleteMultipartUpload"
         ],
         "requestParameters": {
           "bucketName": [
             {
```

```
                "Ref": "SourceBucket"
              }
            ],
            "key": [
              {
                "Ref": "SourceObjectKey"
              }
            ]
          }
        }
      },
      "Targets": [
        {
          "Arn": {
            "Fn::Join": [
              "",
              [
                "arn:aws:codepipeline:",
                {
                  "Ref": "AWS::Region"
                },
                ":",
                {
                  "Ref": "AWS::AccountId"
                },
                ":",
                {
                  "Ref": "AppPipeline"
                }
              ]
            ]
          },
          "RoleArn": {
            "Fn::GetAtt": [
              "EventRole",
              "Arn"
            ]
          },
          "Id": "codepipeline-AppPipeline"
        }
      ]
    }
  }
},
```

```
...
```

3. 將此片段新增到您的第一個範本，以允許跨堆疊功能：

   YAML

   ```
   Outputs:
     SourceBucketARN:
       Description: "S3 bucket ARN that Cloudtrail will use"
       Value: !GetAtt SourceBucket.Arn
       Export:
         Name: SourceBucketARN
   ```

   JSON

   ```
     "Outputs" : {
       "SourceBucketARN" : {
         "Description" : "S3 bucket ARN that Cloudtrail will use",
         "Value" : { "Fn::GetAtt": ["SourceBucket", "Arn"] },
         "Export" : {
           "Name" : "SourceBucketARN"
         }
       }

   ...
   ```

4. 將更新的範本儲存到本機電腦，然後開啟主 AWS CloudFormation 控台。
5. 選擇您的堆疊，然後選擇 Create Change Set for Current Stack (建立目前堆疊的變更集)。
6. 上傳您的更新範本，然後檢視中 AWS CloudFormation所列的變更。這些是會針對堆疊進行的變更。您應該會在清單中看到新資源。
7. 選擇 Execute (執行)。

若要編輯管線的 PollForSourceChanges參數

> **Important**
>
> 當您使用這個方法建立管道時，如果沒有明確設為 false，則 `PollForSourceChanges` 參數會預設為 true。當新增基於事件的變更偵測時，您必須將該參數新增到輸出，並將其設為

false 以停用輪詢。否則，您的管道會針對單一來源變更啟動兩次。如需詳細資訊，請參閱 PollForSourceChanges 參數的預設設定。

- 在範本中，將 PollForSourceChanges 變更為 false。如果您並未在管道定義中包含 PollForSourceChanges，請新增它，並將其設為 false。

  為什麼我會做出此變更？ 將 PollForSourceChanges變更為 false 會關閉定期檢查，因此您只能使用事件型變更偵測。

  YAML

```
        Name: Source
        Actions:
          -
            Name: SourceAction
            ActionTypeId:
              Category: Source
              Owner: AWS
              Version: 1
              Provider: S3
            OutputArtifacts:
              - Name: SourceOutput
            Configuration:
              S3Bucket: !Ref SourceBucket
              S3ObjectKey: !Ref SourceObjectKey
              PollForSourceChanges: false
            RunOrder: 1
```

  JSON

```
{
   "Name": "SourceAction",
   "ActionTypeId": {
     "Category": "Source",
     "Owner": "AWS",
     "Version": 1,
     "Provider": "S3"
   },
   "OutputArtifacts": [
     {
       "Name": "SourceOutput"
```

```
      }
    ],
    "Configuration": {
      "S3Bucket": {
        "Ref": "SourceBucket"
      },
      "S3ObjectKey": {
        "Ref": "SourceObjectKey"
      },
      "PollForSourceChanges": false
    },
    "RunOrder": 1
  }
```

若要為 Amazon S3 管道的 CloudTrail 資源建立第二個範本

- 在單獨的範本中`Resources`，使用`AWS::S3::Bucket``AWS::S3::BucketPolicy`、和`AWS::CloudTrail::Trail` AWS CloudFormation 資源來提供簡單的值區定義和追蹤 CloudTrail。

  我為什麼要進行這項變更？ 鑑於每個帳戶目前有五個 CloudTrail 追蹤的限制，必須個別建立和管理追蹤。(請參閱中的限制 AWS CloudTrail。) 不過，您可以在單一追蹤中包含多個 Amazon S3 儲存貯體，因此您可以建立一次追蹤，然後視需要為其他管道新增 Amazon S3 儲存貯體。將下列內容貼至您的第二個範例範本檔案中。

  YAML

```
###############################################################################
# Prerequisites:
#   - S3 SourceBucket and SourceObjectKey must exist
###############################################################################

Parameters:
  SourceObjectKey:
    Description: 'S3 source artifact'
    Type: String
    Default: SampleApp_Linux.zip

Resources:
  AWSCloudTrailBucketPolicy:
```

```
    Type: AWS::S3::BucketPolicy
    Properties:
      Bucket: !Ref AWSCloudTrailBucket
      PolicyDocument:
        Version: 2012-10-17
        Statement:
          -
            Sid: AWSCloudTrailAclCheck
            Effect: Allow
            Principal:
              Service:
                - cloudtrail.amazonaws.com
            Action: s3:GetBucketAcl
            Resource: !GetAtt AWSCloudTrailBucket.Arn
          -
            Sid: AWSCloudTrailWrite
            Effect: Allow
            Principal:
              Service:
                - cloudtrail.amazonaws.com
            Action: s3:PutObject
            Resource: !Join [ '', [ !GetAtt AWSCloudTrailBucket.Arn, '/
AWSLogs/', !Ref 'AWS::AccountId', '/*' ] ]
            Condition:
              StringEquals:
                s3:x-amz-acl: bucket-owner-full-control
  AWSCloudTrailBucket:
    Type: AWS::S3::Bucket
    DeletionPolicy: Retain
  AwsCloudTrail:
    DependsOn:
      - AWSCloudTrailBucketPolicy
    Type: AWS::CloudTrail::Trail
    Properties:
      S3BucketName: !Ref AWSCloudTrailBucket
      EventSelectors:
        -
          DataResources:
            -
              Type: AWS::S3::Object
              Values:
                - !Join [ '', [ !ImportValue SourceBucketARN, '/', !Ref
 SourceObjectKey ] ]
          ReadWriteType: WriteOnly
```

```
          IncludeManagementEvents: false
      IncludeGlobalServiceEvents: true
      IsLogging: true
      IsMultiRegionTrail: true


...
```

JSON

```
{
  "Parameters": {
    "SourceObjectKey": {
      "Description": "S3 source artifact",
      "Type": "String",
      "Default": "SampleApp_Linux.zip"
    }
  },
  "Resources": {
    "AWSCloudTrailBucket": {
      "Type": "AWS::S3::Bucket",
        "DeletionPolicy": "Retain"
    },
    "AWSCloudTrailBucketPolicy": {
      "Type": "AWS::S3::BucketPolicy",
      "Properties": {
        "Bucket": {
          "Ref": "AWSCloudTrailBucket"
        },
        "PolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [
            {
              "Sid": "AWSCloudTrailAclCheck",
              "Effect": "Allow",
              "Principal": {
                "Service": [
                  "cloudtrail.amazonaws.com"
                ]
              },
              "Action": "s3:GetBucketAcl",
              "Resource": {
                "Fn::GetAtt": [
```

```
                "AWSCloudTrailBucket",
                "Arn"
              ]
            }
          },
          {
            "Sid": "AWSCloudTrailWrite",
            "Effect": "Allow",
            "Principal": {
              "Service": [
                "cloudtrail.amazonaws.com"
              ]
            },
            "Action": "s3:PutObject",
            "Resource": {
              "Fn::Join": [
                "",
                [
                  {
                    "Fn::GetAtt": [
                      "AWSCloudTrailBucket",
                      "Arn"
                    ]
                  },
                  "/AWSLogs/",
                  {
                    "Ref": "AWS::AccountId"
                  },
                  "/*"
                ]
              ]
            },
            "Condition": {
              "StringEquals": {
                "s3:x-amz-acl": "bucket-owner-full-control"
              }
            }
          }
        ]
      }
    }
  },
  "AwsCloudTrail": {
    "DependsOn": [
```

```
        "AWSCloudTrailBucketPolicy"
      ],
      "Type": "AWS::CloudTrail::Trail",
      "Properties": {
        "S3BucketName": {
          "Ref": "AWSCloudTrailBucket"
        },
        "EventSelectors": [
          {
            "DataResources": [
              {
                "Type": "AWS::S3::Object",
                "Values": [
                  {
                    "Fn::Join": [
                      "",
                      [
                        {
                          "Fn::ImportValue": "SourceBucketARN"
                        },
                        "/",
                        {
                          "Ref": "SourceObjectKey"
                        }
                      ]
                    ]
                  }
                ]
              }
            ],
            "ReadWriteType": "WriteOnly",
            "IncludeManagementEvents": false
          }
        ],
        "IncludeGlobalServiceEvents": true,
        "IsLogging": true,
        "IsMultiRegionTrail": true
      }
    }
  }
}

...
```

比特桶雲連接

連線可讓您授權並建立將第三方供應商與 AWS 資源相關聯的組態。若要將第三方存放庫與管道的來源相關聯，請使用連線。

> **Note**
>
> 此功能不適用於亞太區域 (香港)、亞太區域 (海德拉巴)、亞太區域 (雅加達)、亞太區域 (墨爾本)、亞太區域 (大阪)、非洲 (開普敦)、中東 (巴林)、中東 (阿聯酋)、歐洲 (西班牙)、歐洲 (蘇黎世)、以色列 (特拉維夫) 或 AWS GovCloud (美國西部) 區域。若要參考其他可用動作，請參閱產品與服務整合 CodePipeline。如需在歐洲 (米蘭) 區域進行此動作的考量，請參閱中的附註CodeStarSourceConnection 適用於 Bitbucket 雲端 GitHub、 GitHub 企業伺服器、 GitLab .com 和 GitLab 自我管理動作。

若要在中新增 Bitbucket 雲端來源動作 CodePipeline，您可以選擇：

- 使用 CodePipeline 主控台「建立管線精靈」或 「編輯」動作頁面來選擇 Bitbucket 提供者選項。請參創建到比特桶雲（控制台）的連接閱以新增動作。主控台可協助您建立連線資源。

  > **Note**
  >
  > 您可以建立連至 Bitbucket Cloud 儲存庫的連線。不支援安裝式 Bitbucket 供應商類型，例如 Bitbucket 伺服器。

- 使用 CLI 為`Bitbucket`提供者的動作新增`CreateSourceConnection`動作組態，如下所示：
  - 若要建立連線資源，請參創建到比特桶雲（CLI）的連接閱使用 CLI 建立連線資源。
  - 使用中的`CreateSourceConnection`範例動作配置CodeStarSourceConnection 適用於 Bitbucket 雲端 GitHub、 GitHub 企業伺服器、 GitLab .com 和 GitLab 自我管理動作來新增您的動作，如中所示建立管道 (CLI)。

> **Note**
>
> 您也可以使用「設定」下的「開發人員工具」主控台建立連線。請參閱建立連線。

開始之前：

- 您必須使用第三方存儲庫的提供商（例如 Bitbucket 雲）創建了一個帳戶。
- 您必須已經建立了第三方程式碼儲存庫，例如 Bitbucket 雲端儲存庫。

> **Note**
>
> Bitbucket 雲端連線僅提供對用來建立連線之 Bitbucket 雲端帳戶所擁有的儲存庫的存取權。如果應用程式安裝在 Bitbucket Cloud 工作區中，您需要管理工作區權限。否則，安裝該應用程式的選項將不會顯示。

主題

創建到比特桶雲（控制台）的連接

使用這些步驟來使用 CodePipeline 主控台為您的 Bitbucket 儲存庫新增連線動作。

> **Note**
>
> 您可以建立連至 Bitbucket Cloud 儲存庫的連線。不支援安裝式 Bitbucket 供應商類型，例如 Bitbucket 伺服器。

步驟 1：建立或編輯管道

若要建立或編輯管線

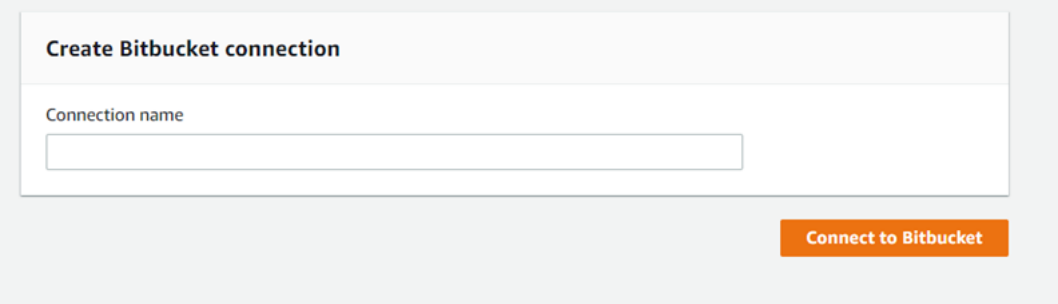
1. 登入 CodePipeline 主控台。
2. 選擇下列其中一項。
   - 選擇建立配管。按照「建立管道」中的步驟完成第一個畫面，然後選擇「下一步」。在 [來源] 頁面的 [來源提供者] 下，選擇 [Bitbucket]。
   - 選擇編輯現有配管。選擇 [編輯]，然後選擇 [編輯階段]。選擇新增或編輯來源動作。在 [編輯動作] 頁面上的 [動作名稱] 下，輸入動作的名稱。在 [動作提供者] 中，選擇 [Bitbucket]。
3. 執行以下任意一項：

- 在 [連線] 下，如果您尚未建立與提供者的連線，請選擇 [Connect 線至 Bitbucket]。繼續執行步驟 2：建立與 Bitbucket 的連線。
- 在 [連線] 下方，如果您已建立與提供者的連線，請選擇連線。繼續執行步驟 3：儲存連線的來源動作。

步驟 2：創建到比特桶雲的連接

若要建立至 Bitbucket 雲端的連線

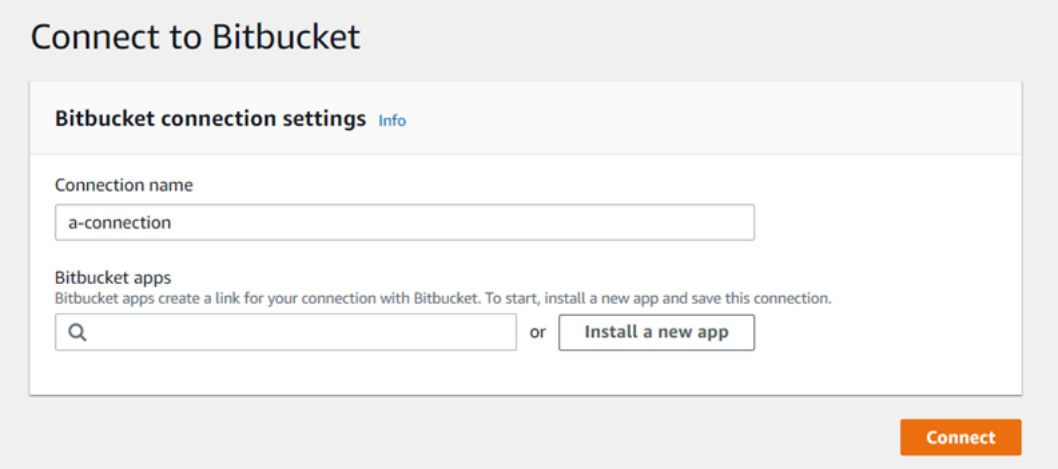
1. 在 [Connect 線至 Bitbucket 設定] 頁面上，輸入您的連線名稱，然後選擇 [Connect 線至 Bitbucket]。



便會出現「Bitbucket 應用程式」欄位。

2. 在 Bitbucket apps (Bitbucket 應用程式) 底下，選擇應用程式安裝，或選擇 Install a new app (安裝新應用程式) 以建立安裝。

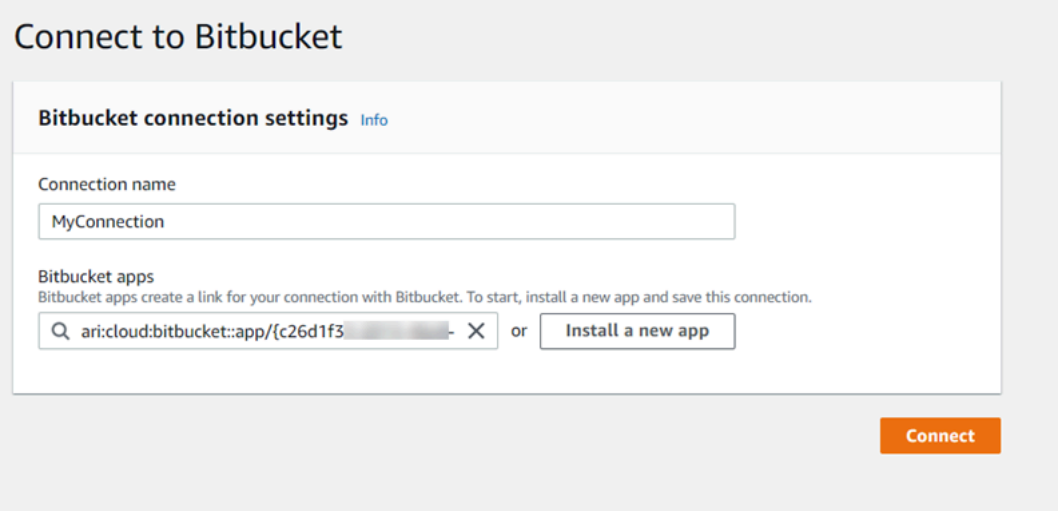
> **Note**
>
> 您只能為每個 Bitbucket 雲端工作區或帳戶安裝一次應用程式。如果您已經安裝了 Bitbucket 應用程序，請選擇它並轉到步驟 4。



3. 如果顯示 Bitbucket Cloud 的登入頁面，請使用您的認證登入，然後選擇繼續。
4. 在應用程式安裝頁面上，會有訊息顯示 AWS CodeStar 應用程式正在嘗試連線至您的 Bitbucket 帳戶。

   若您使用的是 Bitbucket 工作區，請將 Authorize for (授權) 選項變更為工作區。僅會顯示您具有管理員存取權的工作區。

   選擇 Grant access (授與存取權)。
5. 在 Bitbucket apps (Bitbucket 應用程式) 中，會顯示新安裝的連線 ID。選擇 Connect (連線)。建立的連線會顯示在連線清單中。



步驟 3：儲存您的 Bitbucket 雲端來源動作

使用精靈或 「編輯」動作頁面上的這些步驟，將來源動作與連線資訊一起儲存。

透過連線完成並儲存來源動作

1. 在 Repository name (儲存庫名稱) 中，選擇第三方儲存庫的名稱。
2. 如果您的動作是動作，您可以在管線觸發器下新增觸發 CodeConnections 器。若要設定管線觸發器組態並選擇性地使用觸發程式進行篩選，請參閱中的詳細資訊篩選程式碼推送或提取要求的觸發程序。
3. 在 Output artifact format (輸出成品格式) 中，您必須選擇成品的格式。
   - 若要使用預設方法儲存來自 Bitbucket Cloud 動作的輸出成品，請選擇CodePipeline 預設值。此動作會從 Bitbucket Cloud 存放庫存取檔案，並將成品儲存在管線成品存放區中的 ZIP 檔案中。
   - 若要存放包含儲存庫 URL 參考的 JSON 檔案，以便下游動作可以直接執行 Git 命令，請選擇 Full clone (完整複製)。此選項只能由 CodeBuild 下游動作使用。

     如果您選擇此選項，則需要更新 CodeBuild 專案服務角色的權限，如中所示新增連線至 Bitbucket GitHub、 GitHub 企業伺服器或 GitLab .com 的 CodeBuild GitClone 權限。
4. 在精靈中選擇 [下一步]，或在 [編輯動作] 頁面上選擇 [儲

創建到比特桶雲（CLI）的連接

您可以使用 AWS Command Line Interface (AWS CLI) 建立連線。

> **Note**
>
> 您可以建立連至 Bitbucket Cloud 儲存庫的連線。不支援安裝式 Bitbucket 供應商類型，例如 Bitbucket 伺服器。

若要這麼做，請使用 create-connection 命令。

> **Important**
>
> 依預設，透過 AWS CLI 或建立 AWS CloudFormation 的連線處於`PENDING`狀態。建立與 CLI 的連線之後 AWS CloudFormation，或使用主控台編輯連線以顯示其狀態`AVAILABLE`。

建立連線

1. 開啟終端機 (Linux、macOS 或 Unix) 或命令提示 (Windows)。使用 AWS CLI 來執行指create-connection令，--connection-name為您的連線指定--provider-type和。在此範例中，第三方供應商名稱為 Bitbucket，而指定的連線名稱為 MyConnection。

```
aws codestar-connections create-connection --provider-type Bitbucket --connection-name MyConnection
```

如果成功，此命令會傳回類似下列內容的連線 ARN 資訊。

```
{
    "ConnectionArn": "arn:aws:codestar-connections:us-west-2:account_id:connection/aEXAMPLE-8aad-4d5d-8878-dfcab0bc441f"
}
```

2. 使用主控台完成連線。如需詳細資訊，請參閱更新擱置中的連線。
3. 管線預設會偵測程式碼推送至連線來源儲存庫時的變更。若要針對手動發行或 Git 標籤設定管線觸發程序組態，請執行下列其中一個動作：

- 若要將管線觸發器組態設定為僅從手動發行版本開始，請將下列行新增至組態：

```
"DetectChanges": "false",
```

- 若要設定管線觸發器組態以篩選觸發器，請參閱中的詳細資訊篩選程式碼推送或提取要求的觸發程序。例如，以下內容會將 Git 標籤新增至管線 JSON 定義的管線層級。在這個範例中，release-v0和release-v1是要包含的 Git 標籤，而且release-v2是要排除的 Git 標籤。

```
"triggers": [
            {
                "providerType": "CodeStarSourceConnection",
                "gitConfiguration": {
                    "sourceActionName": "Source",
                    "push": [
                        {
                            "tags": {
                                "includes": [
                                    "release-v0", "release-v1"
                                ],
                                "excludes": [
```

```
                                "release-v2"
                            ]
                        }
                    }
                ]
            }
        }
    ]
```

CodeCommit 來源動作和 EventBridge

若要在中新增 CodeCommit 來源動作 CodePipeline，您可以選擇：

- 使用主 CodePipeline 控台「建立管線精靈」(建立管道 (主控台)) 或 「編輯」動作頁面來選擇CodeCommit提供者選項。控制台創建一個 EventBridge規則，該規則在源更改時啟動管道。
- 使用新 AWS CLI 增動作的動作組態，並建立其他資源，如下所示：`CodeCommit`
  - 使用中的`CodeCommit`範例動作配置CodeCommit來建立您的動作，如中所示建立管道 (CLI)。
  - 變更偵測方法預設為透過輪詢來源來啟動管線。您應該停用定期檢查，並手動建立變更偵測規則。使用下列其中一種方法：建立 CodeCommit 來源 (主控台) 的 EventBridge 規則建立 CodeCommit 來源 (CLI) 的 EventBridge 規則、或建立 CodeCommit 來源 (AWS CloudFormation 範本) 的 EventBridge 規則 。

主題

- 建立 CodeCommit 來源 (主控台) 的 EventBridge 規則
- 建立 CodeCommit 來源 (CLI) 的 EventBridge 規則
- 建立 CodeCommit 來源 (AWS CloudFormation 範本) 的 EventBridge 規則

建立 CodeCommit 來源 (主控台) 的 EventBridge 規則

> **Important**
>
> 如果您使用主控台建立或編輯管線， EventBridge 則會為您建立規則。

若要建立用於 CodePipeline 作業的 EventBridge 規則

1. 在以下位置打開 Amazon EventBridge 控制台 https://console.aws.amazon.com/events/。
2. 在導覽窗格中，選擇規則。保持選取預設匯流排，或選擇活動匯流排。選擇建立規則。
3. 在名稱中，輸入規則的名稱。
4. 在「規則類型」下，選擇「具有事件模式的規則」。選擇下一步。
5. 在事件來源下，選擇AWS 事件或 EventBridge 合作夥伴事件。
6. 在 [範例事件類型] 下方，選擇AWS 事件。
7. 在範例事件中，輸入 CodeCommit 要篩選的關鍵字。選擇CodeCommit 儲存庫狀態變更。
8. 在 [建立方法] 下方，選擇 [客戶模式 (JSON 編輯器)]。

   粘貼下面提供的事件模式。以下是「事件」(CodeCommitEvent) 視窗中具有分支名稱的`MyTestRepo`存放庫的範例事件模式`main`：

```
{
  "source": [
    "aws.codecommit"
  ],
  "detail-type": [
    "CodeCommit Repository State Change"
  ],
  "resources": [
    "arn:aws:codecommit:us-west-2:80398EXAMPLE:MyTestRepo"
  ],
  "detail": {
    "referenceType": [
      "branch"
    ],
    "referenceName": [
      "main"
    ]
  }
}
```

9. 在 [目標] 中，選擇CodePipeline。
10. 輸入要由此規則啟動之管線的配管 ARN。

> **Note**
>
> 在您執行 get-pipeline 命令之後，即可在中繼資料輸出中找到管道 ARN。管道 ARN 是以下列格式建構：
>
> *AR: aws: #####:##:##:####*
>
> 範例管道 ARN：
>
> `arn:aws:codepipeline:us-east-2:80398EXAMPLE:MyFirstPipeline`

11. 若要建立或指定 IAM 服務角色，以授與呼叫 EventBridge 規則關聯之目標的 EventBridge 權限 (在本例中，目標為 CodePipeline)：
    - 選擇 [為此特定資源建立新角色] 以建立服務角色，以提供啟動管線執行的 EventBridge權限的服務角色。
    - 選擇 [使用現有角色] 以輸入授與啟動管線執行之 EventBridge 權限的服務角色。
12. 選擇下一步。
13. 在 [標籤] 頁面上，選擇 [下一步]。
14. 在 [檢閱並建立] 頁面上，檢閱規則組態。如果您對此規則感到滿意，請選擇 Create rule (建立規則)。

建立 CodeCommit 來源 (CLI) 的 EventBridge 規則

呼叫 put-rule 命令，並指定：

- 可唯一識別您所建立規則的名稱。在與 AWS 帳戶 CodePipeline 相關聯的所有管道中，此名稱必須是唯一的。
- 規則所使用來源和詳細資訊欄位的事件模式。如需詳細資訊，請參閱 Amazon EventBridge 和事件模式。

若要以作 CodeCommit 為事件來源與 CodePipeline 目標建立 EventBridge 規則

1. 新增用 EventBridge CodePipeline 於呼叫規則的權限。如需詳細資訊，請參閱在 Amazon EventBridge 使用以資源為基礎的政策。
    a. 使用下列範例建立允許擔任服務角色 EventBridge 的信任原則。將信任政策命名為 `trustpolicyforEB.json`。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "events.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

b. 使用下列命令來建立 Role-for-MyRule 角色，並連接信任政策。

```
aws iam create-role --role-name Role-for-MyRule --assume-role-policy-document
 file://trustpolicyforEB.json
```

c. 為名為 MyFirstPipeline 的管道建立許可政策 JSON，如這個範例所示。將許可政策命名為 permissionspolicyforEB.json。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codepipeline:StartPipelineExecution"
            ],
            "Resource": [
                "arn:aws:codepipeline:us-west-2:80398EXAMPLE:MyFirstPipeline"
            ]
        }
    ]
}
```

d. 使用執行以下命令，將 CodePipeline-Permissions-Policy-for-EB 許可政策連接到 Role-for-MyRule 角色。

為什麼我會做出此變更？ 將此原則新增至角色會建立的權限 EventBridge。

```
aws iam put-role-policy --role-name Role-for-MyRule --policy-name CodePipeline-
Permissions-Policy-For-EB --policy-document file://permissionspolicyforEB.json
```

2. 呼叫 put-rule 命令，並包含 `--name`、`--event-pattern` 和 `--role-arn` 參數。

   為什麼我會做出此變更？ 此命令可讓 AWS CloudFormation 建立事件。

   以下範例命令會建立名為 `MyCodeCommitRepoRule` 的規則。

```
aws events put-rule --name "MyCodeCommitRepoRule" --event-pattern "{\"source\":
[\"aws.codecommit\"],\"detail-type\":[\"CodeCommit Repository State Change\"],
\"resources\":[\"repository-ARN\"],\"detail\":{\"referenceType\":[\"branch\"],
\"referenceName\":[\"main\"]}}" --role-arn "arn:aws:iam::ACCOUNT_ID:role/Role-for-
MyRule"
```

3. 若要新增 CodePipeline 為目標，請呼叫指put-targets令並包含下列參數：

   - `--rule` 參數與您使用 put-rule 所建立的 `rule_name` 搭配使用。
   - `--targets` 參數與目標清單中目標的清單 `Id` 和目標管道的 ARN 搭配使用。

   以下命令範例指定名為 `MyCodeCommitRepoRule` 的規則，該目標 `Id` 是由數字 1 組成，指出在規則的目標清單中，這是目標 1。此範例命令也會指定管道的範例 ARN。儲存庫中若發生變更，管道就會啟動。

```
aws events put-targets --rule MyCodeCommitRepoRule --targets
 Id=1,Arn=arn:aws:codepipeline:us-west-2:80398EXAMPLE:TestPipeline
```

若要編輯管線的 PollForSourceChanges參數

> **Important**
>
> 當您使用這個方法建立管道時，如果沒有明確設為 false，則 `PollForSourceChanges` 參數會預設為 true。當新增基於事件的變更偵測時，您必須將該參數新增到輸出，並將其設為 false 以停用輪詢。否則，您的管道會針對單一來源變更啟動兩次。如需詳細資訊，請參閱 PollForSourceChanges 參數的預設設定。

1. 執行 get-pipeline 命令，將管道結構複製到 JSON 檔案。例如，針對名為 MyFirstPipeline 的管道，執行下列命令：

```
aws codepipeline get-pipeline --name MyFirstPipeline >pipeline.json
```

此命令不會傳回任何內容，但您建立的檔案應該會顯示在您執行命令的目錄中。

2. 在任何純文字編輯器中開啟 JSON 檔案，然後將 PollForSourceChanges 參數變更為 false，來編輯來源階段，如這個範例所示。

為什麼我會做出此變更？ 將此參數變更為 false 會關閉定期檢查，因此您只能使用事件型變更偵測。

```
"configuration": {
    "PollForSourceChanges": "false",
    "BranchName": "main",
    "RepositoryName": "MyTestRepo"
},
```

3. 如果您使用的是使用 get-pipeline 命令擷取的管道結構，請從 JSON 檔案中移除 metadata 行。否則，update-pipeline 命令無法使用它。移除 "metadata": { } 行，以及 "created"、"pipelineARN" 和 "updated" 欄位。

例如，從結構中移除下列幾行：

```
"metadata": {
    "pipelineArn": "arn:aws:codepipeline:region:account-ID:pipeline-name",
    "created": "date",
    "updated": "date"
},
```

儲存檔案。

4. 若要套用您的變更，請執行 update-pipeline 命令、指定管道 JSON 檔案：

> **Important**
>
> 請確認在檔案名稱之前包含 file://。這是此命令必要項目。

```
aws codepipeline update-pipeline --cli-input-json file://pipeline.json
```

此命令會傳回所編輯管道的整個結構。

> **Note**
>
> update-pipeline 命令將終止管道。若在您執行 update-pipeline 命令時有修訂正在透過管道執行，該執行將停止。您必須手動啟動管道，以透過更新的管道執行該修訂。使用 **start-pipeline-execution** 命令來手動啟動您的管道。

建立 CodeCommit 來源 (AWS CloudFormation 範本) 的 EventBridge 規則

若要用 AWS CloudFormation 來建立規則，請更新範本，如下所示。

更新管線 AWS CloudFormation 範本並建立 EventBridge 規則

1. 在範本中`Resources`，使用資`AWS::IAM::Role` AWS CloudFormation 源來設定允許事件啟動管道的 IAM 角色。此項目會建立一個使用兩個政策的角色：

   - 第一個政策允許要承擔的角色。
   - 第二個政策提供啟動管道的許可。

   為什麼我會做出此變更？ 新增資`AWS::IAM::Role`源可 AWS CloudFormation 建立的權限 EventBridge。此資源會新增至您的 AWS CloudFormation 堆疊。

   YAML

   ```
   EventRole:
     Type: AWS::IAM::Role
     Properties:
       AssumeRolePolicyDocument:
         Version: 2012-10-17
         Statement:
           -
             Effect: Allow
             Principal:
   ```

```
              Service:
                - events.amazonaws.com
            Action: sts:AssumeRole
      Path: /
      Policies:
        -
          PolicyName: eb-pipeline-execution
          PolicyDocument:
            Version: 2012-10-17
            Statement:
              -
                Effect: Allow
                Action: codepipeline:StartPipelineExecution
                Resource: !Join [ '', [ 'arn:aws:codepipeline:', !Ref
 'AWS::Region', ':', !Ref 'AWS::AccountId', ':', !Ref AppPipeline ] ]
```

JSON

```
"EventRole": {
  "Type": "AWS::IAM::Role",
  "Properties": {
    "AssumeRolePolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "Service": [
              "events.amazonaws.com"
            ]
          },
          "Action": "sts:AssumeRole"
        }
      ]
    },
    "Path": "/",
    "Policies": [
      {
        "PolicyName": "eb-pipeline-execution",
        "PolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [
            {
```

```
              "Effect": "Allow",
              "Action": "codepipeline:StartPipelineExecution",
              "Resource": {
                "Fn::Join": [
                  "",
                  [
                    "arn:aws:codepipeline:",
                    {
                      "Ref": "AWS::Region"
                    },
                    ":",
                    {
                      "Ref": "AWS::AccountId"
                    },
                    ":",
                    {
                      "Ref": "AppPipeline"
                    }
                  ]

...
```

2. 在範本的下Resources，使用資AWS::Events::Rule AWS CloudFormation 源新增 EventBridge 規則。此事件模式會建立一個事件，以監視存放庫的推送變更。 EventBridge 偵測到存放庫狀態變更時，規則會StartPipelineExecution在您的目標管線上叫用。

   我為什麼要進行這項變更？ 新增資AWS::Events::Rule源可 AWS CloudFormation 建立事件。此資源會新增至您的 AWS CloudFormation 堆疊。

   YAML

```
  EventRule:
    Type: AWS::Events::Rule
    Properties:
      EventPattern:
        source:
          - aws.codecommit
        detail-type:
          - 'CodeCommit Repository State Change'
        resources:
          - !Join [ '', [ 'arn:aws:codecommit:', !Ref 'AWS::Region', ':', !Ref
'AWS::AccountId', ':', !Ref RepositoryName ] ]
        detail:
```

```
          event:
            - referenceCreated
            - referenceUpdated
          referenceType:
            - branch
          referenceName:
            - main
      Targets:
        -
          Arn:
            !Join [ '', [ 'arn:aws:codepipeline:', !Ref 'AWS::Region', ':', !Ref
 'AWS::AccountId', ':', !Ref AppPipeline ] ]
          RoleArn: !GetAtt EventRole.Arn
          Id: codepipeline-AppPipeline
```

JSON

```
"EventRule": {
  "Type": "AWS::Events::Rule",
  "Properties": {
    "EventPattern": {
      "source": [
        "aws.codecommit"
      ],
      "detail-type": [
        "CodeCommit Repository State Change"
      ],
      "resources": [
        {
          "Fn::Join": [
            "",
            [
              "arn:aws:codecommit:",
              {
                "Ref": "AWS::Region"
              },
              ":",
              {
                "Ref": "AWS::AccountId"
              },
              ":",
              {
```

```
                "Ref": "RepositoryName"
              }
            ]
          ]
        }
      ],
      "detail": {
        "event": [
          "referenceCreated",
          "referenceUpdated"
        ],
        "referenceType": [
          "branch"
        ],
        "referenceName": [
          "main"
        ]
      }
    },
    "Targets": [
      {
        "Arn": {
          "Fn::Join": [
            "",
            [
              "arn:aws:codepipeline:",
              {
                "Ref": "AWS::Region"
              },
              ":",
              {
                "Ref": "AWS::AccountId"
              },
              ":",
              {
                "Ref": "AppPipeline"
              }
            ]
          ]
        },
        "RoleArn": {
          "Fn::GetAtt": [
            "EventRole",
            "Arn"
```

```
          ]
        },
        "Id": "codepipeline-AppPipeline"
      }
    ]
  }
},
```

3. 將更新的範本儲存到本機電腦，然後開啟 AWS CloudFormation 主控台。
4. 選擇您的堆疊，然後選擇 Create Change Set for Current Stack (建立目前堆疊的變更集)。
5. 上傳範本，然後檢視 AWS CloudFormation中所列的變更。這些是會針對堆疊進行的變更。您應該會在清單中看到新資源。
6. 選擇 Execute (執行)。

若要編輯管線的 PollForSourceChanges參數

> **Important**
>
> 在許多情況下，當您建立管道時，`PollForSourceChanges` 參數會預設為 true。當新增基於事件的變更偵測時，您必須將該參數新增到輸出，並將其設為 false 以停用輪詢。否則，您的管道會針對單一來源變更啟動兩次。如需詳細資訊，請參閱 PollForSourceChanges 參數的預設設定。

- 在範本中，將 `PollForSourceChanges` 變更為 `false`。如果您並未在管道定義中包含 `PollForSourceChanges`，請新增它，並將其設為 `false`。

  為什麼我會做出此變更？ 將此參數變更為 `false` 會關閉定期檢查，因此您只能使用事件型變更偵測。

  YAML

```
      Name: Source
      Actions:
        -
          Name: SourceAction
          ActionTypeId:
            Category: Source
            Owner: AWS
            Version: 1
```

```
            Provider: CodeCommit
          OutputArtifacts:
            - Name: SourceOutput
          Configuration:
            BranchName: !Ref BranchName
            RepositoryName: !Ref RepositoryName
            PollForSourceChanges: false
          RunOrder: 1
```

JSON

```
{
  "Name": "Source",
  "Actions": [
    {
      "Name": "SourceAction",
      "ActionTypeId": {
        "Category": "Source",
        "Owner": "AWS",
        "Version": 1,
        "Provider": "CodeCommit"
      },
      "OutputArtifacts": [
        {
          "Name": "SourceOutput"
        }
      ],
      "Configuration": {
        "BranchName": {
          "Ref": "BranchName"
        },
        "RepositoryName": {
          "Ref": "RepositoryName"
        },
        "PollForSourceChanges": false
      },
      "RunOrder": 1
    }
  ]
},
```

GitHub 連接

您可以使用連線來授權和建立將您的第三方供應商與您的AWS資源相關聯的組態。

> **Note**
>
> 此功能不適用於亞太區域 (香港)、亞太區域 (海德拉巴)、亞太區域 (雅加達)、亞太區域 (墨爾本)、亞太區域 (大阪)、非洲 (開普敦)、中東 (巴林)、中東 (阿聯酋)、歐洲 (西班牙)、歐洲 (蘇黎世)、以色列 (特拉維夫) 或 AWS GovCloud (美國西部) 區域。若要參考其他可用動作，請參閱產品與服務整合 CodePipeline。如需在歐洲 (米蘭) 區域進行此動作的考量，請參閱中的附註CodeStarSourceConnection 適用於 Bitbucket 雲端 GitHub、 GitHub 企業伺服器、GitLab .com 和 GitLab 自我管理動作。

若要在中新增您的 GitHub 或 GitHub 企業雲端儲存庫的來源動作 CodePipeline，您可以選擇：

- 使用主 CodePipeline 控台「建立管線精靈」或 「編輯」動作頁面來選擇 GitHub (版本 2) 提供者選項。請參建立與 GitHub 企業伺服器 (主控台) 的連線閱以新增動作。主控台可協助您建立連線資源。

  > **Note**
  >
  > 如需逐步引導您如何在管線中新增 GitHub 連線及使用「完整複製」選項的教學課程，請參閱教學課程：搭 GitHub 配管線來源使用完整複製。

- 使用 CLI 為`GitHub`提供者的動作新增`CodeStarSourceConnection`動作組態，並使用中所示的 CLI 步驟建立管道 (CLI)。

> **Note**
>
> 您也可以使用「設定」下的「開發人員工具」主控台建立連線。請參閱建立連線。

開始之前：

- 您必須已在建立一個帳戶 GitHub。
- 您必須已經建立 GitHub 程式碼儲存庫。

- 如果您的 CodePipeline 服務角色是在 2019 年 12 月 18 日之前建立的，您可能需要更新其權限才能用`codestar-connections:UseConnection`於AWS CodeStar連線。如需說明，請參閱將許可新增至 CodePipeline 服務角色。

> **Note**
>
> 若要建立連線，您必須是 GitHub 組織擁有者。對於不在組織下的儲存庫，您必須是儲存庫擁有者。

主題

建立連線至 GitHub (主控台)

使用這些步驟來使用 CodePipeline 主控台為您 GitHub或 GitHub 企業雲端儲存庫新增連線動作。

> **Note**
>
> 在這些步驟中，您可以在存放庫存取下選取特定的儲存庫。任何未選取的儲存庫將無法存取或顯示 CodePipeline。

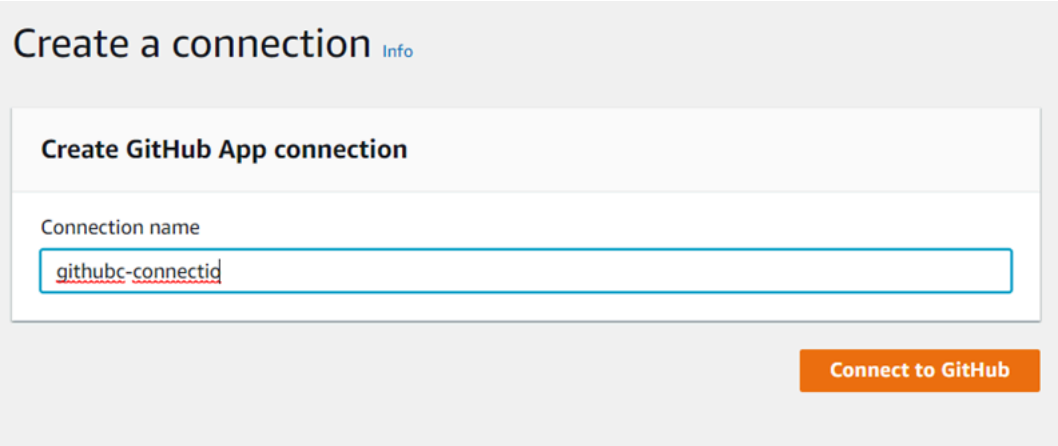
步驟 1：建立或編輯管道

1. 登入 CodePipeline 主控台。
2. 選擇下列其中一項。
   - 選擇建立配管。按照「建立管道」中的步驟完成第一個畫面，然後選擇「下一步」。在 [來源] 頁面的 [來源提供者] 下，選擇 GitHub [版本 2]。
   - 選擇編輯現有配管。選擇 [編輯]，然後選擇 [編輯階段]。選擇新增或編輯來源動作。在 [編輯動作] 頁面上的 [動作名稱] 下，輸入動作的名稱。在 [動作提供者] 中，選擇 [GitHub 版本 2]。
3. 執行以下任意一項：
   - 在 [連線] 下方，如果您尚未建立與提供者的連線，請選擇 [Connect 線至] GitHub。繼續執行「步驟 2：建立連線至 GitHub」。

- 在 [連線] 下方，如果您已建立與提供者的連線，請選擇連線。繼續執行步驟 3：儲存連線的來源動作。

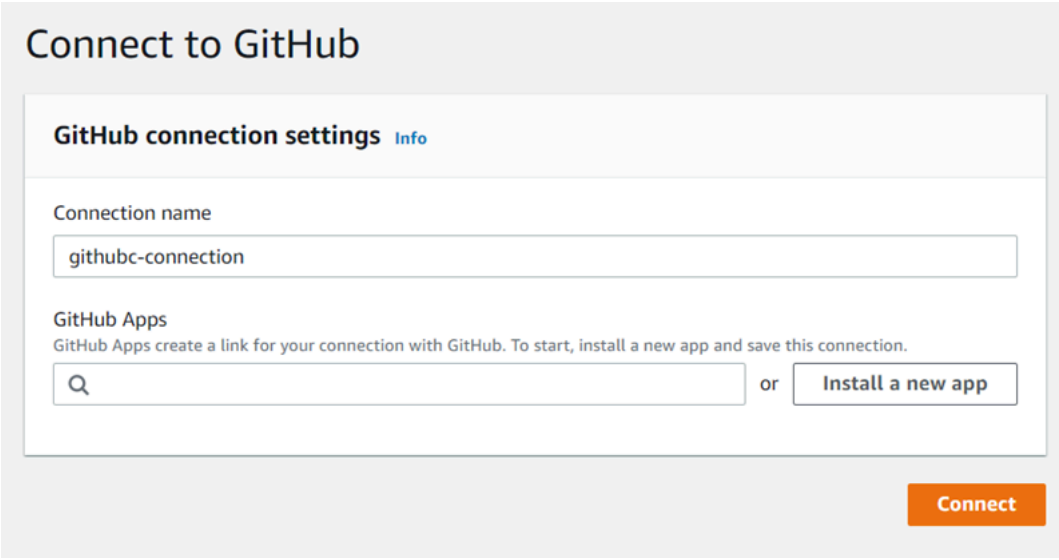
步驟 2：建立連線 GitHub

選擇建立連線之後，會出現 [連線至 GitHub] 頁面。



若要建立連線 GitHub

1. 在GitHub 連線設定下，您的連線名稱會出現在連線名稱中。選擇「Connect 至」 GitHub。隨即會顯示存取請求頁面。
2. 選擇 [授權AWS連接器] GitHub。連線頁面隨即顯示並顯示 [GitHub 應用程式] 欄位。



3. 在 [GitHub 應用程式] 下方，選擇應用程式安裝，或選擇 [安裝新的應用程式] 來建立

> **Note**
>
> 您可以為您連至特定供應商的所有連線安裝一個應用程式。如果您已經安裝 GitHub應用程式的AWS連接器，請選擇該連接器並略過此步驟。

4. 在 [安裝AWS連接器 GitHub] 頁面上，選擇您要安裝應用程式的帳戶。

> **Note**
>
> 您只能為每個 GitHub 帳戶安裝一次該應用程序。如果您先前已安裝應用程式，可以選擇 Configure (設定)，繼續前往應用程式安裝的修改頁面，或者您可以使用上一步按鈕返回主控台。

5. 在 [安裝AWS連接器 GitHub] 頁面上，保留預設值，然後選擇 [安裝]。
6. 在 [Connect 線至 GitHub] 頁面上，新安裝的連線 ID 會出現在GitHub 應用程式中。選擇連線。

步驟 3：儲存來 GitHub 源動作

使用 「編輯」動作頁面上的這些步驟，將來源動作與連線資訊一起儲存。

若要儲存 GitHub 來源動作

1. 在 Repository name (儲存庫名稱) 中，選擇第三方儲存庫的名稱。
2. 如果您的動作是動作，您可以在管線觸發器下新增觸發CodeConnections器。若要設定管線觸發器組態並選擇性地使用觸發程式進行篩選，請參閱中的詳細資訊篩選程式碼推送或提取要求的觸發程序。
3. 在 Output artifact format (輸出成品格式) 中，您必須選擇成品的格式。

   - 若要使用預設方法儲存 GitHub 動作的輸出成品，請選擇CodePipeline 預設值。此動作會從存 GitHub 放庫存取檔案，並將人工因素儲存在管線人工因素存放區中的 ZIP 檔案中。
   - 若要存放包含儲存庫 URL 參考的 JSON 檔案，以便下游動作可以直接執行 Git 命令，請選擇 Full clone (完整複製)。此選項只能由 CodeBuild 下游動作使用。

     如果您選擇此選項，則需要更新 CodeBuild 專案服務角色的權限，如中所示新增連線至 Bitbucket GitHub、 GitHub 企業伺服器或 GitLab .com 的 CodeBuild GitClone 權限。如需說明如何使用「完整複製」選項的教學課程，請參閱教學課程：搭 GitHub 配管線來源使用完整複製。

4. 在精靈中選擇 [下一步]，或在 [編輯動作] 頁面上選擇 [儲

建立與 GitHub (CLI) 的連線

您可以使用 AWS Command Line Interface (AWS CLI) 刪除連線。

若要這麼做，請使用 create-connection 命令。

> **Important**
>
> 依預設，透過 AWS CLI 或 AWS CloudFormation 建立的連線會處於 PENDING 狀態。建立連至 CLI 或 AWS CloudFormation 的連線後，請使用主控台編輯連線，將其狀態設為 AVAILABLE。

建立連線

1. 開啟終端機 (Linux、macOS 或 Unix) 或命令提示 (Windows)。使用 AWS CLI 來執行 create-connection 命令，為連線指定 `--connection-name` 和 `--provider-type`。在此範例中，第三方供應商名稱為 `GitHub`，而指定的連線名稱為 `MyConnection`。

   ```
   aws codestar-connections create-connection --provider-type GitHub --connection-name
    MyConnection
   ```

   如果成功，此命令會傳回類似下列內容的連線 ARN 資訊。

   ```
   {
       "ConnectionArn": "arn:aws:codestar-connections:us-west-2:account_id:connection/
   aEXAMPLE-8aad-4d5d-8878-dfcab0bc441f"
   }
   ```

2. 使用主控台完成連線。如需詳細資訊，請參閱更新擱置中的連線。
3. 管線預設會偵測程式碼推送至連線來源儲存庫時的變更。若要針對手動發行或 Git 標籤設定管線觸發程序組態，請執行下列其中一個動作：

   - 若要將管線觸發器組態設定為僅從手動發行版本開始，請將下列行新增至組態：

     ```
     "DetectChanges": "false",
     ```

- 若要設定管線觸發器組態以篩選觸發器，請參閱中的詳細資訊篩選程式碼推送或提取要求的觸發程序。例如，以下內容會新增至管線 JSON 定義的管線層級。在這個範例中，release-v0和release-v1是要包含的 Git 標籤，而且release-v2是要排除的 Git 標籤。

```
"triggers": [
            {
                "providerType": "CodeStarSourceConnection",
                "gitConfiguration": {
                    "sourceActionName": "Source",
                    "push": [
                        {
                            "tags": {
                                "includes": [
                                    "release-v0", "release-v1"
                                ],
                                "excludes": [
                                    "release-v2"
                                ]
                            }
                        }
                    ]
                }
            }
        ]
```

GitHub 企業伺服器連線

連線可讓您授權並建立將第三方供應商與AWS資源相關聯的組態。若要將第三方存放庫與管道的來源相關聯，請使用連線。

> **Note**
>
> 此功能不適用於亞太區域 (香港)、亞太區域 (海德拉巴)、亞太區域 (雅加達)、亞太區域 (墨爾本)、亞太區域 (大阪)、非洲 (開普敦)、中東 (巴林)、中東 (阿聯酋)、歐洲 (西班牙)、歐洲 (蘇黎世)、以色列 (特拉維夫) 或 AWS GovCloud (美國西部) 區域。若要參考其他可用動作，請參閱產品與服務整合 CodePipeline。如需在歐洲 (米蘭) 區域進行此動作的考量，請參閱中的附註CodeStarSourceConnection 適用於 Bitbucket 雲端 GitHub、 GitHub 企業伺服器、GitLab .com 和 GitLab 自我管理動作。

若要在中新增 GitHub 企業伺服器來源動作 CodePipeline，您可以選擇：

- 使用 CodePipeline 主控台「建立管線精靈」或 「編輯」動作頁面選擇「GitHub 企業伺服器提供者」選項。請參建立與 GitHub 企業伺服器 (主控台) 的連線閱以新增動作。主控台可協助您建立主機資源和連線資源。
- 使用 CLI 為`GitHubEnterpriseServer`提供者的動作新增`CreateSourceConnection`動作組態，並建立您的資源：
  - 若要建立連線資源，請參建立主機並連線至 GitHub 企業伺服器 (CLI)閱使用 CLI 建立主機資源和連線資源。
  - 使用中的`CreateSourceConnection`範例動作配置CodeStarSourceConnection 適用於 Bitbucket 雲端 GitHub、 GitHub 企業伺服器、 GitLab .com 和 GitLab 自我管理動作來新增您的動作，如中所示建立管道 (CLI)。

> **Note**
>
> 您也可以使用「設定」下的「開發人員工具」主控台建立連線。請參閱建立連線。

開始之前：

- 您必須已建立 GitHub 企業伺服器帳戶，並在您的基礎結構上安裝 GitHub 企業伺服器執行個體。

  > **Note**
  >
  > 每個 VPC 一次只能與一台主機 (GitHub 企業伺服器執行個體) 產生關聯。

- 您必須已經使用 GitHub 企業伺服器建立程式碼儲存庫。

主題

建立與 GitHub 企業伺服器 (主控台) 的連線

使用下列步驟可使用 CodePipeline 主控台新增 GitHub企業伺服器儲存區域的連線動作。

> **Note**
>
> GitHub 企業伺服器連線只能存取用來建立連線之 GitHub 企業伺服器帳戶所擁有的儲存庫。

在您開始之前：

對於 GitHub 企業伺服器的主機連線，您必須完成為連線建立主機資源的步驟。請參閱管理主機以取得連線。

步驟 1：建立或編輯管道

若要建立或編輯管線

1. 登入 CodePipeline 主控台。
2. 選擇下列其中一項。
   - 選擇建立配管。按照「建立管道」中的步驟完成第一個畫面，然後選擇「下一步」。在 [來源] 頁面的 [來源提供者] 下，選擇 [GitHub 企業伺服器]。
   - 選擇編輯現有配管。選擇 [編輯]，然後選擇 [編輯階段]。選擇新增或編輯來源動作。在 [編輯動作] 頁面的 [動作名稱] 下，輸入動作的名稱。在動作提供者中，選擇GitHub 企業伺服器。
3. 執行以下任意一項：
   - 在 [連線] 底下，如果您尚未建立與提供者的連線，請選擇 [Connect 線到 GitHub 企業伺服器]。繼續執行步驟 2：建立與 GitHub 企業伺服器的連線。
   - 在 [連線] 下方，如果您已建立與提供者的連線，請選擇連線。繼續執行步驟 3：儲存連線的來源動作。

建立與 GitHub企業伺服器的連線

選擇建立連線之後，便會顯示「連線到 GitHub企業伺服器」頁面。

> **Important**
>
> AWS CodeStar 連線不支援 GitHub 企業伺服器 2.22.0 版，因為發行版本中存在已知問題。若要連線，請升級至 2.22.1 版或最新的可用版本。

連線至 GitHub 企業伺服器

1. 針對 Connection name (連線名稱)，請輸入連線的名稱。
2. 在 URL 中，輸入伺服器的端點。

> **Note**
>
> 如果提供的 URL 已用於為連線設定 GitHub企業伺服器，系統會提示您選擇先前為該端點建立的主機資源 ARN。

3. 如果您已將伺服器啟動到 Amazon VPC 中，且想與 VPC 連線，請選擇 Use a VPC (使用 VPC)，然後完成以下操作。

   a. 在 VPC ID 底下，選擇您的 VPC ID。請務必針對安裝 GitHub 企業伺服器執行個體的基礎結構選擇 VPC，或是可透過 VPN 或直 Connect 存取 GitHub企業伺服器執行個體的 VPC。

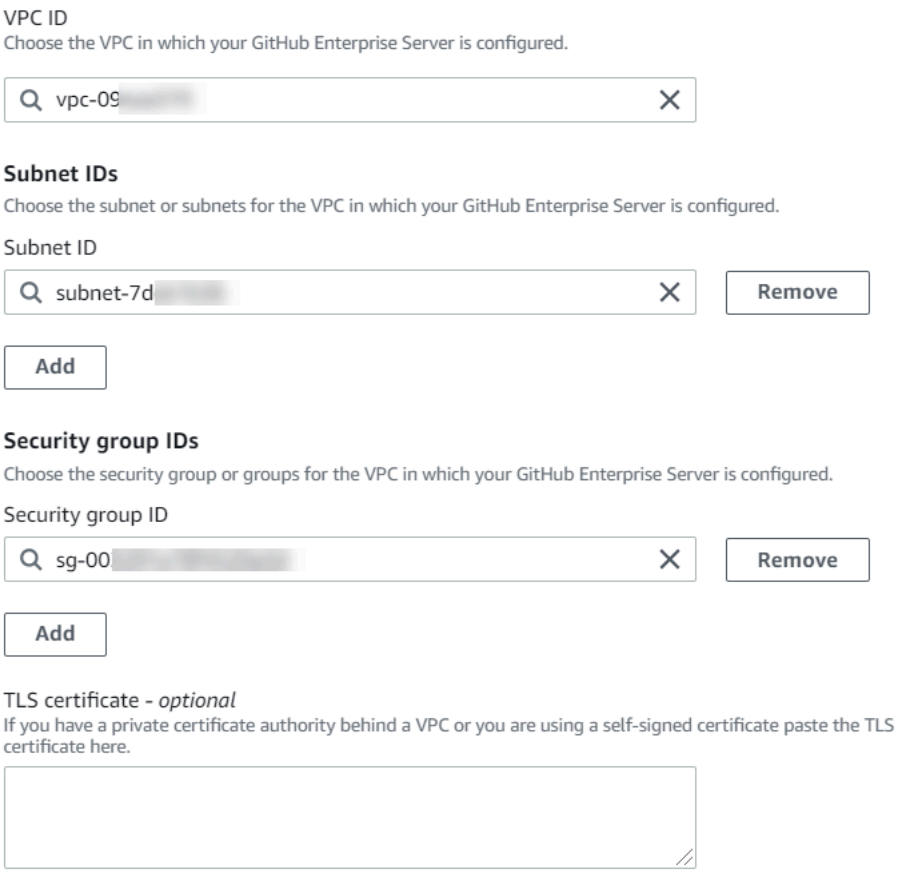
   b. 在 Subnet ID (子網路 ID) 底下，選擇 Add (新增)。在欄位中，選擇您要用於主機的子網路 ID。您最多可選擇 10 個子網路。

      請務必針對安裝 GitHub Enterprise Server 執行個體的基礎結構選擇子網路，或是可透過 VPN 或直 Connect 存取已安裝 GitHub 企業伺服器執行個體的子網路。

   c. 在 Security group IDs (安全群組 ID) 底下，選擇 Add (新增)。在欄位中，選擇您要用於主機的安全群組。您最多可以選擇 10 個安全群組。

      請務必針對安裝 GitHub Enterprise Server 執行個體的基礎結構選擇安全性群組，或可透過 VPN 或直 Connect 存取已安裝 GitHub 企業伺服器執行個體的安全性群組。

   d. 如果您已設定私有 VPC，並且已將 GitHub 企業伺服器執行個體設定為使用非公用憑證授權單位執行 TLS 驗證，請在 TLS 憑證中輸入您的憑證 ID。TLS 憑證值應該是憑證的公有金鑰。



4. 選擇 [Connect 到 GitHub 企業伺服器]。建立的連線會顯示 Pending (待定) 狀態。系統會利用您提供的伺服器資訊為連線建立主機資源。主機名稱會使用 URL。
5. 選擇 Update pending connection (更新待定連線)。
6. 如果出現提示，請在 GitHub 企業登入頁面上，使用您的 GitHub企業認證登入。
7. 在「建立 GitHub 應用程式」頁面上，選擇應用程式的名稱。
8. 在 GitHub 授權頁面上，選擇授權<app-name>。
9. 在應用程式安裝頁面上，會出現訊息，顯示AWS CodeStar連接器應用程式已準備好安裝。如果您有多個組織，系統可能會提示您選擇要安裝應用程式的組織。

   選擇您要安裝應用程式的儲存庫設定。選擇 Install (安裝)。
10. 連線頁面會顯示建立的連線處於 Available (可用) 狀態。

步驟 3：儲存您的 GitHub 企業伺服器來源動作

使用精靈或 「編輯」動作頁面上的這些步驟，將來源動作與連線資訊一起儲存。

透過連線完成並儲存來源動作

1. 在 Repository name (儲存庫名稱) 中，選擇第三方儲存庫的名稱。
2. 如果您的動作是動作，您可以在管線觸發器下新增觸發CodeConnections器。若要設定管線觸發器組態並選擇性地使用觸發程式進行篩選，請參閱中的詳細資訊篩選程式碼推送或提取要求的觸發程序。
3. 在 Output artifact format (輸出成品格式) 中，您必須選擇成品的格式。
   - 若要使用預設方法儲存「 GitHub 企業伺服器」動作的輸出人工因素，請選擇CodePipeline預設值。此動作會存取 GitHub 企業伺服器存放庫中的檔案，並將人工因素儲存在管線人工因素存放區中的 ZIP 檔案中。
   - 若要存放包含儲存庫 URL 參考的 JSON 檔案，以便下游動作可以直接執行 Git 命令，請選擇 Full clone (完整複製)。此選項只能由 CodeBuild 下游動作使用。
4. 在精靈中選擇 [下一步]，或在 [編輯動作] 頁面上選擇 [儲

建立主機並連線至 GitHub 企業伺服器 (CLI)

您可以使用 AWS Command Line Interface (AWS CLI) 刪除連線。

若要這麼做，請使用 create-connection 命令。

> **Important**
>
> 依預設，透過 AWS CLI 或 AWS CloudFormation 建立的連線會處於 `PENDING` 狀態。建立連至 CLI 或 AWS CloudFormation 的連線後，請使用主控台編輯連線，將其狀態設為 `AVAILABLE`。

您可以使用 AWS Command Line Interface (AWS CLI) 為已安裝的連線建立主機。

> **Note**
>
> 每個 GitHub 企業伺服器帳戶只能建立一個主機一次。您與特定 GitHub 企業伺服器帳戶的所有連線都會使用相同的主機。

您可以使用主機來代表安裝第三方供應商的基礎設施之端點。使用 CLI 完成主機建立後，主機處於 [擱置中] 狀態。然後，您可以設定或註冊主機，將其移至「可用」狀態。主機變為可用後，便可完成建立連線的步驟。

若要這麼做，請使用 create-host 命令。

> **Important**
>
> 根據預設，透過 AWS CLI 建立的主機會處於 Pending 狀態。使用 CLI 建立主機後，請使用主控台或 CLI 設定主機以顯示其狀態Available。

建立主機

1. 開啟終端機 (Linux、macOS 或 Unix) 或命令提示 (Windows)。使用 AWS CLI 來執行 create-host 命令，為連線指定 --name、--provider-type 和 --provider-endpoint。在此範例中，第三方供應商名稱為 GitHubEnterpriseServer，而端點為 my-instance.dev。

   ```
   aws codestar-connections create-host --name MyHost --provider-type
    GitHubEnterpriseServer --provider-endpoint "https://my-instance.dev"
   ```

   如果成功，此命令會傳回類似下列內容的主機 Amazon Resource Name (ARN) 資訊。

   ```
   {
       "HostArn": "arn:aws:codestar-connections:us-west-2:account_id:host/My-
   Host-28aef605"
   }
   ```

   完成此步驟後，主機會處於 PENDING 狀態。
2. 使用主控台完成主機設定，並將主機變為 Available 狀態。

建立與 GitHub 企業伺服器的連線

1. 開啟終端機 (Linux、macOS 或 Unix) 或命令提示 (Windows)。使用 AWS CLI 來執行 create-connection 命令，為連線指定 --connection-name 和 --host-arn。

```
aws codestar-connections create-connection --host-arn arn:aws:codestar-
connections:us-west-2:account_id:host/MyHost-234EXAMPLE --connection-name
 MyConnection
```

如果成功，此命令會傳回類似下列內容的連線 ARN 資訊。

```
{
    "ConnectionArn": "arn:aws:codestar-connections:us-west-2:account_id:connection/
aEXAMPLE-8aad"
}
```

2. 使用主控台來設定待定連線。
3. 管線預設會偵測程式碼推送至連線來源儲存庫時的變更。若要針對手動發行或 Git 標籤設定管線觸發程序組態，請執行下列其中一個動作：

   - 若要將管線觸發器組態設定為僅從手動發行版本開始，請將下列行新增至組態：

     ```
     "DetectChanges": "false",
     ```

   - 若要設定管線觸發器組態以篩選觸發器，請參閱中的詳細資訊篩選程式碼推送或提取要求的觸發程序。例如，以下內容會新增至管線 JSON 定義的管線層級。在這個範例中，`release-v0`和`release-v1`是要包含的 Git 標籤，而且`release-v2`是要排除的 Git 標籤。

     ```
     "triggers": [
                {
                    "providerType": "CodeStarSourceConnection",
                    "gitConfiguration": {
                        "sourceActionName": "Source",
                        "push": [
                            {
                                "tags": {
                                    "includes": [
                                        "release-v0", "release-v1"
                                    ],
                                    "excludes": [
                                        "release-v2"
                                    ]
                                }
                            }
                        ]
     ```

```
                }
            }
        ]
```

GitLab.com 連線

連線可讓您授權並建立將第三方供應商與AWS資源相關聯的組態。若要將您的第三方存放庫與管道的來源相關聯，請使用連線。

> **Note**
>
> 亞太區域 (香港)、亞太區域 (海德拉巴)、亞太區域 (雅加達)、亞太區域 (墨爾本)、亞太區域 (大阪)、非洲 (開普敦)、中東 (巴林)、中東 (阿聯酋)、歐洲 (西班牙)、歐洲 (蘇黎世)、以色列 (特拉維夫) 或 AWS GovCloud (美國西部) 區域不提供此功能。若要參考其他可用動作，請參閱產品與服務整合 CodePipeline。如需在歐洲 (米蘭) 區域進行此動作的考量，請參閱中的附註CodeStarSourceConnection 適用於 Bitbucket 雲端 GitHub、 GitHub 企業伺服器、GitLab .com 和 GitLab 自我管理動作。

若要在中新增 GitLab .com 來源動作 CodePipeline，您可以選擇：

- 使用 CodePipeline 主控台「建立管線精靈」或 「編輯」動作頁面來選擇GitLab提供者選項。請參建立與 GitLab .com（主控台）的連線閱以新增動作。主控台可協助您建立連線資源。
- 使用 CLI 為`GitLab`提供者的動作新增`CreateSourceConnection`動作組態，如下所示：
  - 若要建立連線資源，請參建立與 GitLab .com (CLI) 的連線閱使用 CLI 建立連線資源。
  - 使用中的`CreateSourceConnection`範例動作配置CodeStarSourceConnection 適用於 Bitbucket 雲端 GitHub、 GitHub 企業伺服器、 GitLab .com 和 GitLab 自我管理動作來新增您的動作，如中所示建立管道 (CLI)。

> **Note**
>
> 您也可以使用「設定」下的「開發人員工具」主控台建立連線。請參閱建立連線。

> **Note**
>
> 通過在 GitLab .com 中授權此連接安裝，您授予我們的服務許可以通過訪問您的帳戶來處理您的數據，並且可以通過卸載該應用程序隨時撤消許可。

開始之前：

- 您必須已經在 GitLab .com 創建了一個帳戶。

  > **Note**
  >
  > 連線只能存取用於建立和授權連線之帳戶擁有的儲存庫。

  > **Note**
  >
  > 您可以建立與具有 Owner 角色之儲存庫的連線 GitLab，然後連線可與具有諸如資源的存放庫一起使用 CodePipeline。如果是群組中的儲存庫，您不需要為群組擁有者。

- 要為管道指定源，您必須已經在 gitlab.com 上創建了一個存儲庫。

主題

建立與 GitLab .com（主控台）的連線

使用下列步驟可使用 CodePipeline 主控台在中為您的專案 (存放庫) 新增連線動作 GitLab。

若要建立或編輯管線

1. 登入 CodePipeline 主控台。
2. 選擇下列其中一項。
   - 選擇建立配管。按照「建立管道」中的步驟完成第一個畫面，然後選擇「下一步」。在 [來源] 頁面的 [來源提供者] 下，選擇GitLab。

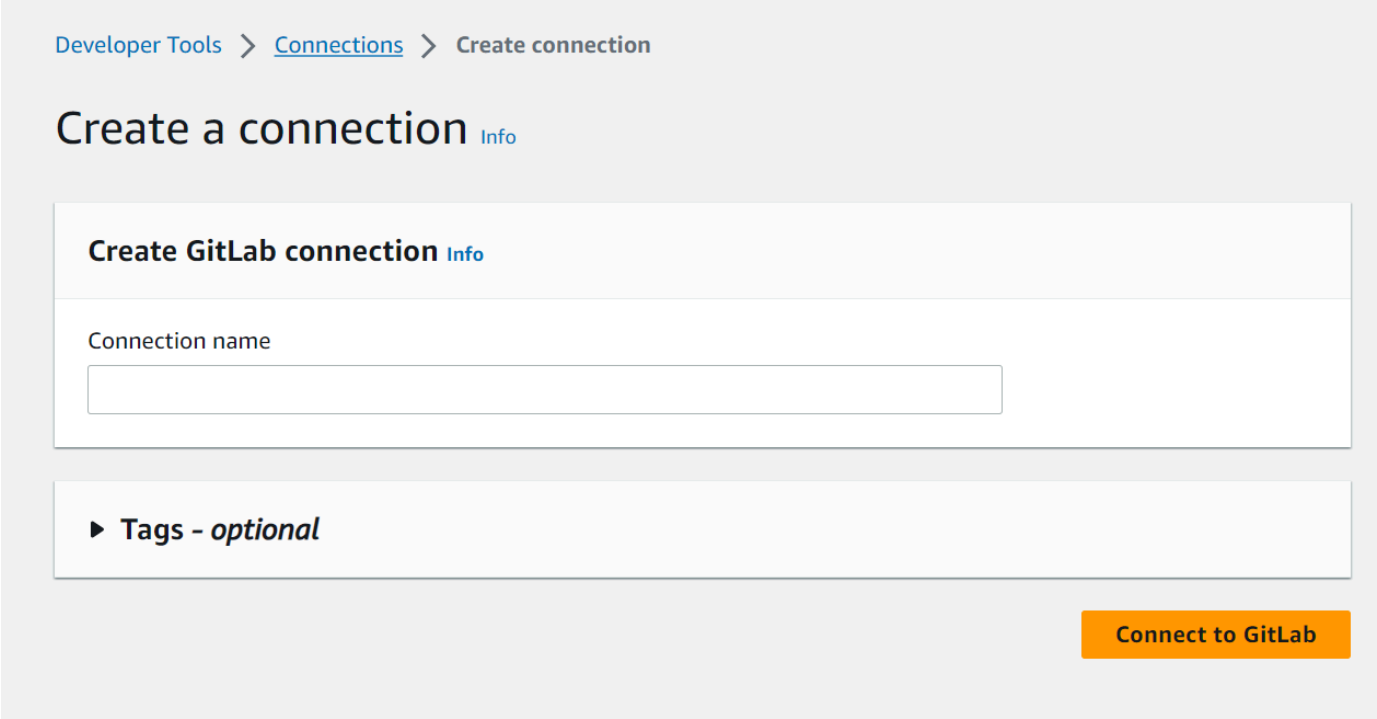
- 選擇編輯現有配管。選擇 [編輯]，然後選擇 [編輯階段]。選擇新增或編輯來源動作。在 [編輯動作] 頁面上的 [動作名稱] 下，輸入動作的名稱。在 [動作提供者] 中，選擇GitLab。

3. 執行以下任意一項：

   - 在 [連線] 下方，如果您尚未建立與提供者的連線，請選擇 [Connect 線至] GitLab。繼續執行步驟 4 以建立連線。
   - 在 [連線] 下方，如果您已建立與提供者的連線，請選擇連線。繼續執行步驟 9。

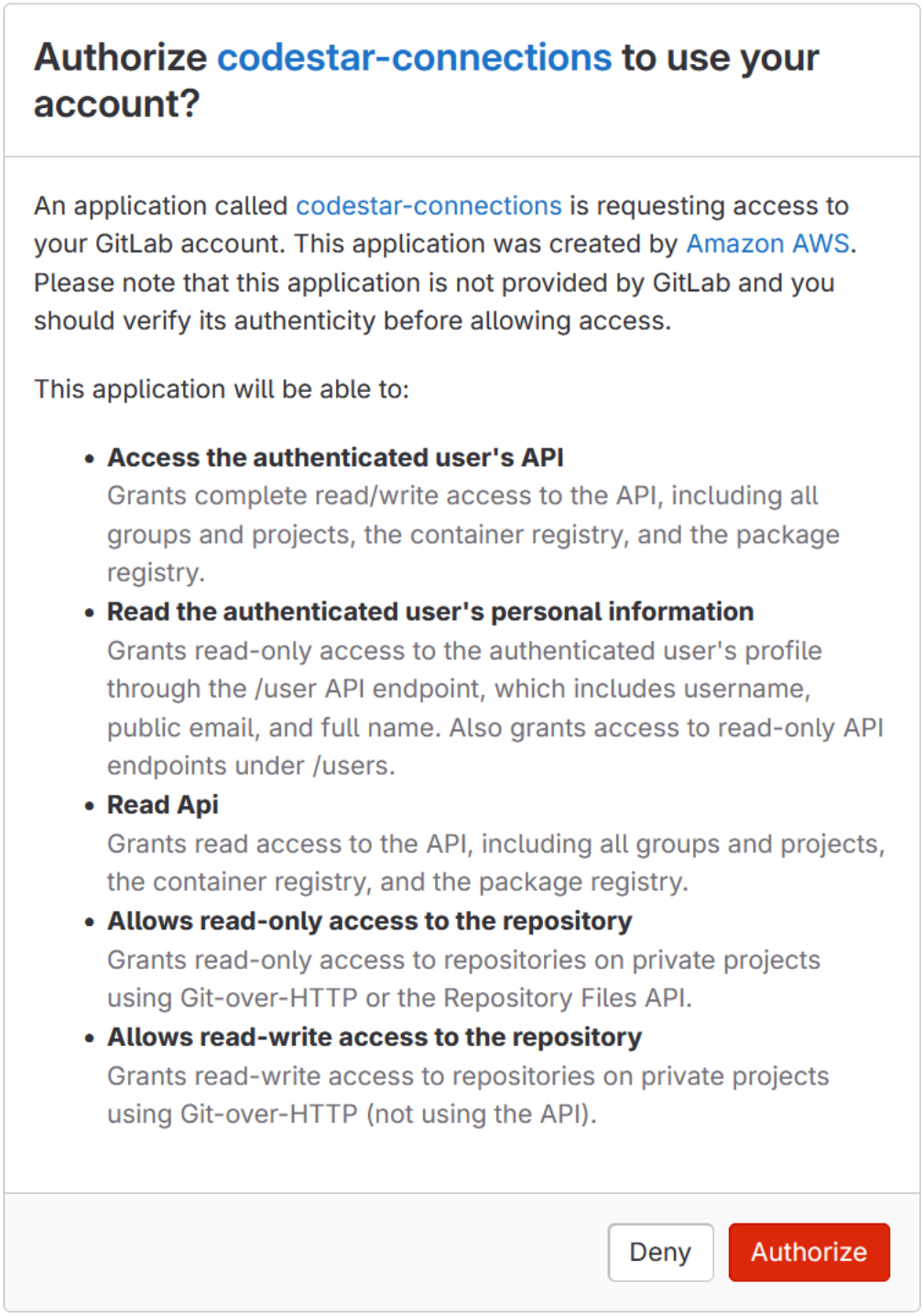
> **Note**
>
> 如果您在創建 GitLab .com 連接之前關閉彈出窗口，則需要刷新頁面。

4. 若要建立 GitLab .com 存放庫的連線，請在 [選取提供者] 下選擇GitLab。在 Connection name (連線名稱) 底下，輸入您要建立的連線名稱。選擇「Connect 至」 GitLab。



5. 顯示 GitLab .com 的登入頁面時，使用您的認證登入，然後選擇 [登入]。
6. 如果這是您第一次授權連線，則會顯示授權頁面，其中會顯示一則訊息，要求授權連線存取您的 GitLab .com 帳戶。

   選擇 Authorize (授權)。



7. 瀏覽器會返回連線主控台頁面。在「建立 GitLab 連線」下，新的連線會顯示在「連線名稱」中。
8. 選擇「Connect 至」 GitLab。

   您將返回 CodePipeline 控制台。

> **Note**
>
> 成功建立 GitLab .com 連線後，主視窗上會顯示成功橫幅。
> 如果您之前尚未在目前的電腦 GitLab 上登入，則需要手動關閉快顯視窗。

9. 在「存放庫名稱」中，透過使用命名空間指 GitLab 定專案路徑來選擇中的專案名稱。例如，對於群組層級存放庫，請以下列格式輸入存放庫名稱：`group-name/repository-name`如需有關路徑和命名空間的詳細資訊，請參閱 https://docs.gitlab.com/ee/api/projects.html #中的欄`path_with_namespace`位get-single-project。如需有關中命名空間的詳細資訊 GitLab，請參閱 https://docs.gitlab.com/ee/user/namespace/。

   > **Note**
   >
   > 對於中的群組 GitLab，您必須使用命名空間手動指定專案路徑。例如，對於群組`myrepo`中名為的存放庫`mygroup`，請輸入以下內容：`mygroup/myrepo`您可以在中的 URL 中找到具有命名空間的專案路徑 GitLab。

10. 如果您的動作是動作，您可以在管線觸發器下新增觸發CodeConnections器。若要設定管線觸發器組態並選擇性地使用觸發程式進行篩選，請參閱中的詳細資訊篩選程式碼推送或提取要求的觸發程序。

11. 在 Branch name (分支名稱) 中，選擇您要讓管道偵測來源變更的分支。

    > **Note**
    >
    > 如果分支名稱未自動填入，則您沒有存放庫的 Owner 存取權。項目名稱無效，或者使用的連接無法訪問項目/存儲庫。

12. 在 Output artifact format (輸出成品格式) 中，您必須選擇成品的格式。

    - 若要使用預設方法儲存來自 GitLab .com 動作的輸出成品，請選擇CodePipeline 預設值。此動作會存取 GitLab .com 存放庫中的檔案，並將成品儲存在管線人工因素存放區中的 ZIP 檔案中。
    - 若要存放包含儲存庫 URL 參考的 JSON 檔案，以便下游動作可以直接執行 Git 命令，請選擇 Full clone (完整複製)。此選項只能由 CodeBuild 下游動作使用。

      如果您選擇此選項，則需要更新 CodeBuild 專案服務角色的權限，如中所示新增連線至 Bitbucket GitHub、 GitHub 企業伺服器或 GitLab .com 的 CodeBuild GitClone 權限。如需說明

如何使用「完整複製」選項的教學課程，請參閱教學課程：搭 GitHub 配管線來源使用完整複製。

13. 選擇儲存來源動作並繼續。

建立與 GitLab .com (CLI) 的連線

您可以使用 AWS Command Line Interface (AWS CLI) 刪除連線。

若要這麼做，請使用 create-connection 命令。

> **Important**
>
> 依預設，透過 AWS CLI 或 AWS CloudFormation 建立的連線會處於 PENDING 狀態。建立連至 CLI 或 AWS CloudFormation 的連線後，請使用主控台編輯連線，將其狀態設為 AVAILABLE。

建立連線

1. 開啟終端機 (Linux、macOS 或 Unix) 或命令提示 (Windows)。使用 AWS CLI 來執行 create-connection 命令，為連線指定 `--connection-name` 和 `--provider-type`。在此範例中，第三方供應商名稱為 `GitLab`，而指定的連線名稱為 `MyConnection`。

   ```
   aws codestar-connections create-connection --provider-type GitLab --connection-name
    MyConnection
   ```

   如果成功，此命令會傳回類似下列內容的連線 ARN 資訊。

   ```
   {
       "ConnectionArn": "arn:aws:codestar-connections:us-west-2:account_id:connection/
   aEXAMPLE-8aad-4d5d-8878-dfcab0bc441f"
   }
   ```

2. 使用主控台完成連線。如需詳細資訊，請參閱更新擱置中的連線。
3. 管線預設會偵測程式碼推送至連線來源儲存庫時的變更。若要針對手動發行或 Git 標籤設定管線觸發程序組態，請執行下列其中一個動作：

   - 若要將管線觸發器組態設定為僅從手動發行版本開始，請將下列行新增至組態：

```
"DetectChanges": "false",
```

- 若要設定管線觸發器組態以篩選觸發器，請參閱中的詳細資訊篩選程式碼推送或提取要求的觸發程序。例如，以下內容會新增至管線 JSON 定義的管線層級。在這個範例中，`release-v0`和`release-v1`是要包含的 Git 標籤，而且`release-v2`是要排除的 Git 標籤。

```
"triggers": [
            {
                "providerType": "CodeStarSourceConnection",
                "gitConfiguration": {
                    "sourceActionName": "Source",
                    "push": [
                        {
                            "tags": {
                                "includes": [
                                    "release-v0", "release-v1"
                                ],
                                "excludes": [
                                    "release-v2"
                                ]
                            }
                        }
                    ]
                }
            }
        ]
```

GitLab 自我管理的連線

連線可讓您授權並建立將第三方供應商與 AWS 資源相關聯的組態。若要將第三方存放庫與管道的來源相關聯，請使用連線。

> **Note**
>
> 亞太區域 (香港)、亞太區域 (海德拉巴)、亞太區域 (雅加達)、亞太區域 (墨爾本)、亞太區域 (大阪)、非洲 (開普敦)、中東 (巴林)、中東 (阿聯酋)、歐洲 (西班牙)、歐洲 (蘇黎世)、以色列 (特拉維夫) 或 AWS GovCloud (美國西部) 區域不提供此功能。若要參考其他可用動作，請參閱產品與服務整合 CodePipeline。如需在歐洲 (米蘭) 區域進行此動作的考量，請參閱

中的附註CodeStarSourceConnection 適用於 Bitbucket 雲端 GitHub、 GitHub 企業伺服器、 GitLab .com 和 GitLab 自我管理動作。

若要在中新增 GitLab 自我管理的來源動作 CodePipeline，您可以選擇：

- 使用 CodePipeline 主控台「建立管線精靈」或 「編輯」動作頁面來選擇GitLab自我管理的提供者選項。請參建立與 GitLab自我管理 (主控台) 的連線閱以新增動作。主控台可協助您建立主機資源和連線資源。
- 使用 CLI 為`GitLabSelfManaged`提供者的動作新增`CreateSourceConnection`動作組態，並建立您的資源：
  - 若要建立連線資源，請參建立主機和與 GitLab自我管理 (CLI) 的連線閱使用 CLI 建立主機資源和連線資源。
  - 使用中的`CreateSourceConnection`範例動作配置CodeStarSourceConnection 適用於 Bitbucket 雲端 GitHub、 GitHub 企業伺服器、 GitLab .com 和 GitLab 自我管理動作來新增您的動作，如中所示建立管道 (CLI)。

> **Note**
>
> 您也可以使用「設定」下的「開發人員工具」主控台建立連線。請參閱建立連線。

開始之前：

- 您必須已經建立了具 GitLab 有自我管理安裝的 GitLab 企業版或 GitLab 社群版的帳戶。如需詳細資訊，請參閱 https://docs.gitlab.com/ee/subscriptions/self_managed/。

> **Note**
>
> 連線只能存取用於建立和授權連線之帳戶。

> **Note**
>
> 您可以建立與具有 Owner 角色的存放庫的連線 GitLab，然後連線可與資源 (例如) 搭配使用 CodePipeline。如果是群組中的儲存庫，您不需要為群組擁有者。

- 您必須已經創建了具有以下範圍縮小權限的 GitLab 個人訪問令牌（PAT）：api。如需詳細資訊，請參閱 https://docs.gitlab.com/ee/user/profile/personal_access_tokens.html。您必須是管理員才能建立和使用 PAT。

  > **Note**
  >
  > 您的 PAT 會用於授權主機，不會以其他方式儲存或由連線使用。若要設置主體，您可以建立臨時 PAT，然後在設置主體後刪除 PAT。

- 您可以選擇提前設定主機。您可以使用或不使用 VPC 來設定主機。如需 VPC 組態的詳細資訊以及有關建立主機的其他資訊，請參閱建立主機。

主題

建立與 GitLab自我管理 (主控台) 的連線

使用這些步驟來使用主 CodePipeline 控台為您的 GitLab自我管理儲存庫新增連線動作。

> **Note**
>
> GitLab 自我管理的連線只能存取用來建立連線的 GitLab 自我管理帳戶所擁有的儲存庫。

在您開始之前：

對於 GitLab 自我管理的主機連線，您必須完成為連線建立主機資源的步驟。請參閱管理主機以取得連線。

步驟 1：建立或編輯管道

若要建立或編輯管線

1. 登入 CodePipeline 主控台。
2. 選擇下列其中一項。

   - 選擇建立配管。按照「建立管道」中的步驟完成第一個畫面，然後選擇「下一步」。在 [來源] 頁面的 [來源提供者] 下，選擇 [GitLab 自我管理]。

- 選擇編輯現有配管。選擇 [編輯]，然後選擇 [編輯階段]。選擇新增或編輯來源動作。在 [編輯動作] 頁面上的 [動作名稱] 下，輸入動作的名稱。在動作提供者中，選擇GitLab 自我管理。

3. 執行以下任意一項：

   - 在 [連線] 下方，如果您尚未建立與提供者的連線，請選擇 [Connect 線到 GitLab 自我管理]。繼續執行步驟 2：建立與 GitLab 自我管理的連線。
   - 在 [連線] 下方，如果您已建立與提供者的連線，請選擇連線。繼續執行步驟 3：儲存連線的來源動作。

建立與 GitLab 自我管理的連線

選擇建立連線後，會顯示 [連線至 GitLab自我管理] 頁面。

連接到 GitLab 自我管理

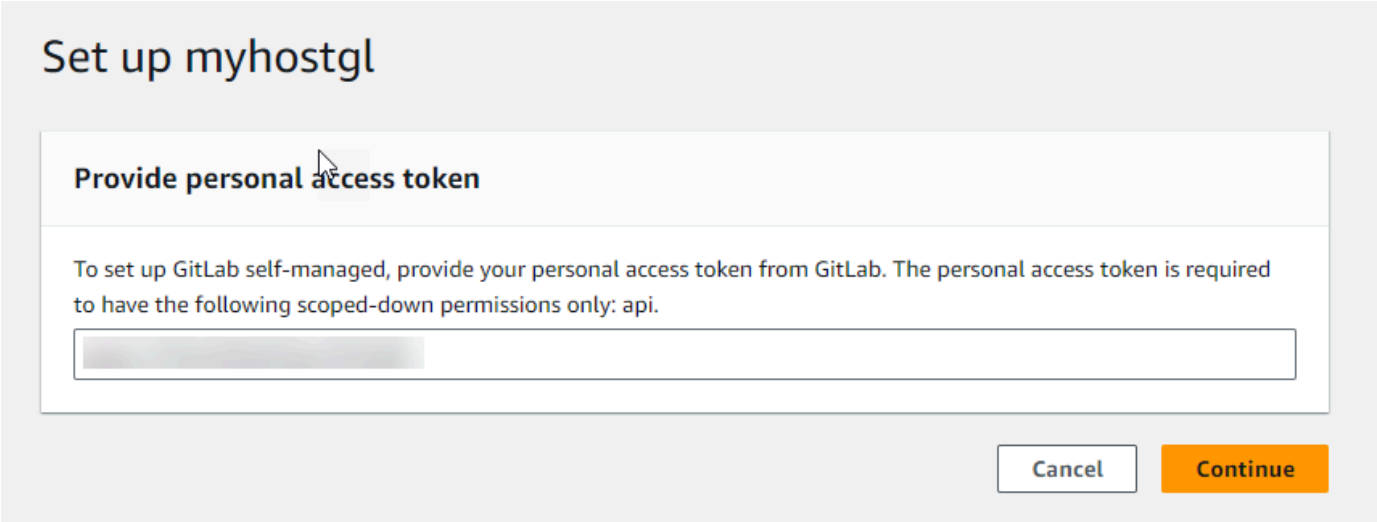
1. 針對 Connection name (連線名稱)，請輸入連線的名稱。
2. 在 URL 中，輸入伺服器的端點。

   > **Note**
   >
   > 如果提供的 URL 已用於設定連線的主機，系統會提示您選擇先前為該端點建立的主機資源 ARN。

3. 如果您已將伺服器啟動到 Amazon VPC，並且想要與 VPC 連線，請選擇「使用 VPC」並完成 VPC 的資訊。
4. 選擇「Connect 至 GitLab 自我管理」。建立的連線會顯示 Pending (待定) 狀態。系統會利用您提供的伺服器資訊為連線建立主機資源。主機名稱會使用 URL。
5. 選擇 Update pending connection (更新待定連線)。
6. 如果頁面開啟時顯示重新導向訊息，確認您要繼續前往提供者，請選擇「繼續」。輸入提供者的授權。
7. 將顯示設定 ***host_name*** 頁面。在 [提供個人存取權杖] 中，僅向您的 GitLab PAT 提供下列縮減權限：`api`

   > **Note**
   >
   > 只有管理員可以建立和使用 PAT。

選擇 繼續。



8. 連線頁面會顯示建立的連線處於 Available (可用) 狀態。

步驟 3：儲存您的 GitLab自我管理來源動作

使用精靈或 「編輯」動作頁面上的這些步驟，將來源動作與連線資訊一起儲存。

透過連線完成並儲存來源動作

1. 在 Repository name (儲存庫名稱) 中，選擇第三方儲存庫的名稱。
2. 如果您的動作是動作，您可以在管線觸發器下新增觸發 CodeConnections 器。若要設定管線觸發器組態並選擇性地使用觸發程式進行篩選，請參閱中的詳細資訊篩選程式碼推送或提取要求的觸發程序。
3. 在 Output artifact format (輸出成品格式) 中，您必須選擇成品的格式。
   - 若要使用預設方法儲存來 GitLab 自自我管理動作的輸出成品，請選擇 CodePipelinedefault。此動作會從存放庫存取檔案，並將人工因素儲存在管線人工因素存放區中的 ZIP 檔案中。
   - 若要存放包含儲存庫 URL 參考的 JSON 檔案，以便下游動作可以直接執行 Git 命令，請選擇 Full clone (完整複製)。此選項只能由 CodeBuild 下游動作使用。
4. 在精靈中選擇 [下一步]，或在 [編輯動作] 頁面上選擇 [儲

建立主機和與 GitLab自我管理 (CLI) 的連線

您可以使用 AWS Command Line Interface (AWS CLI) 建立連線。

若要這麼做，請使用 create-connection 命令。

> **Important**
>
> 依預設，透過 AWS CLI 或建立 AWS CloudFormation 的連線處於PENDING狀態。建立與 CLI 的連線之後 AWS CloudFormation，或使用主控台編輯連線以顯示其狀態AVAILABLE。

您可以使用 AWS Command Line Interface (AWS CLI) 為已安裝的連線建立主機。

您可以使用主機來代表安裝第三方供應商的基礎設施之端點。使用 CLI 完成主機建立後，主機處於 [擱置中] 狀態。然後，您可以設定或註冊主機，將其移至「可用」狀態。主機變為可用後，便可完成建立連線的步驟。

若要這麼做，請使用 create-host 命令。

> **Important**
>
> 依預設，透過建立 AWS CLI 的主機處於Pending狀態。使用 CLI 建立主機後，請使用主控台或 CLI 設定主機以顯示其狀態Available。

建立主機

1. 開啟終端機 (Linux、macOS 或 Unix) 或命令提示 (Windows)。使用 AWS CLI 來執行指create-host令，--provider-endpoint為您的連線指定--provider-type、和。--name在此範例中，第三方供應商名稱為 GitLabSelfManaged，而端點為 my-instance.dev。

   ```
   aws codestar-connections create-host --name MyHost --provider-type
    GitLabSelfManaged --provider-endpoint "https://my-instance.dev"
   ```

   如果成功，此命令會傳回類似下列內容的主機 Amazon Resource Name (ARN) 資訊。

   ```
   {
       "HostArn": "arn:aws:codestar-connections:us-west-2:account_id:host/My-
   Host-28aef605"
   }
   ```

   完成此步驟後，主機會處於 PENDING 狀態。

2. 使用主控台完成主機設定，並將主機變為 Available 狀態。

建立與 GitLab 自我管理的連線

1. 開啟終端機 (Linux、macOS 或 Unix) 或命令提示 (Windows)。使用 AWS CLI 來執行指create-connection令，--connection-name為您的連線指定--host-arn和。

```
aws codestar-connections create-connection --host-arn arn:aws:codestar-
connections:us-west-2:account_id:host/MyHost-234EXAMPLE --connection-name
 MyConnection
```

如果成功，此命令會傳回類似下列內容的連線 ARN 資訊。

```
{
    "ConnectionArn": "arn:aws:codestar-connections:us-west-2:account_id:connection/
aEXAMPLE-8aad"
}
```

2. 使用主控台來設定待定連線。
3. 管線預設會偵測程式碼推送至連線來源儲存庫時的變更。若要針對手動發行或 Git 標籤設定管線觸發程序組態，請執行下列其中一個動作：

   - 若要將管線觸發器組態設定為僅從手動發行版本開始，請將下列行新增至組態：

```
"DetectChanges": "false",
```

   - 若要設定管線觸發器組態以篩選觸發器，請參閱中的詳細資訊篩選程式碼推送或提取要求的觸發程序。例如，以下內容會新增至管線 JSON 定義的管線層級。在這個範例中，release-v0和release-v1是要包含的 Git 標籤，而且release-v2是要排除的 Git 標籤。

```
"triggers": [
            {
                "providerType": "CodeStarSourceConnection",
                "gitConfiguration": {
                    "sourceActionName": "Source",
                    "push": [
                        {
                            "tags": {
                                "includes": [
                                    "release-v0", "release-v1"
```

```
                        ],
                        "excludes": [
                            "release-v2"
                        ]
                    }
                }
            ]
        }
    }
]
```

在中編輯配管 CodePipeline

管道描述了您想要 AWS CodePipeline 遵循的發佈程序，包括必須完成的階段和動作。您可以透過編輯管道來新增或移除這些元素。然而，當您編輯管道時，不可變更管道名稱或管道中繼資料。

您可以使用配管編輯頁面編輯管線類型、變數和觸發器。您也可以在管道中新增或變更階段和動作。

和建立管道不同，編輯管道並不會傳回最近一次透過管道執行的修訂版。如果您想透過剛編輯的管道執行最近的修訂版，您必須將其手動傳回。否則，編輯的管道將在下次您更改來源階段中設定的來源位置時執行。如需相關資訊，請參閱 手動啟動管道。

您可以將動作新增至管道位於與管道不同的AWS區域中的管道。當一個AWS 服務是動作的提供者，且此動作類型/提供者類型與管線位於不同的AWS區域時，這是跨區域動作。如需有關跨區域動作的詳細資訊，請參閱在 CodePipeline 中新增跨區域動作。

CodePipeline 推送原始程式碼變更時，會使用變更偵測方法來啟動管線。這些偵測方法視原始碼類型而定：

- CodePipeline 使用 Amazon CloudWatch 事件偵測來 CodeCommit 源儲存庫或 Amazon S3 來源儲存貯體中的變更。

> **Note**
>
> 您使用主控台時會自動建立變更偵測資源。當您使用主控台建立或編輯管道時，將為您建立其他資源。如果您使用 AWS CLI 來建立管道，則必須自行建立其他資源。如需建立或更新 CodeCommit 配管的詳細資訊，請參閱建立 CodeCommit 來源 (CLI) 的 EventBridge 規則。如需使用 CLI 建立或更新 Amazon S3 管道的詳細資訊，請參閱為 Amazon S3 來源 (CLI) 建立 EventBridge 規則。

主題

編輯管道 (主控台)

您可以使用 CodePipeline 主控台新增、編輯或移除管道中的階段，以及新增、編輯或移除階段中的動作。

當您更新管線時，會正 CodePipeline常地完成所有執行中的動作，然後將完成執行中動作的階段和管線執行失敗。管道更新後，您將需要重新執行管道。如需執行管線的詳細資訊，請參閱手動啟動管道。

編輯管道

1. 請登入AWS Management Console並開啟 CodePipeline 主控台，網址為 http://console.aws.amazon.com/codesuite/codepipeline/home。

   所有與您的 AWS 帳戶相關的管道名稱都會顯示。
2. 在 Name (名稱) 中，選擇您想編輯的管道名稱。這會開啟管道的詳細檢視，包含管道中各階段的每項動作之每項狀態。
3. 在管道詳細資訊頁面上，選擇 Edit (編輯)。
4. 若要編輯配管類型，請選擇 「編輯：配管屬性」卡上的「編輯」。選擇下列其中一個選項，然後選擇 [完成]。

   - V1 類型管道具有包含標準管道、階段和動作層級參數的 JSON 結構。
   - V2 類型管線與 V1 類型具有相同的結構，以及其他參數支援，例如觸發程序和管線層級變數。

   管道類型的特性和價格不同。如需詳細資訊，請參閱 管線類型。
5. 若要編輯管線變數，請在 [編輯:變數] 卡上選擇 [編輯變數]。新增或變更管線層級的變數，然後選擇「完成」。

   如需管線層級變數的詳細資訊，請參閱Variables。如需在管線執行時傳送的管線層級變數的教學課程，請參閱。教學課程：使用管線層級變數

> **Note**
>
> 雖然在管線層級新增變數是可選的，但對於在沒有提供值的管線層級使用變數指定的配管，管線執行將會失敗。

6. 若要編輯管線觸發器，請選擇 [編輯:觸發器] 卡上的 [編輯觸發器]。新增或變更觸發器，然後選擇 [完成]。

   如需新增觸發程式的詳細資訊，請參閱建立與 Bitbucket Cloud GitHub (版本 2)、 GitHub 企業伺服器、 GitLab .com 或 GitLab自我管理的連線的步驟，例如。GitHub 連接

7. 若要編輯「編輯」頁面上的階段和動作，請執行下列其中一個動作：

   - 若要編輯階段，請選擇 Edit stage (編輯階段)。您可以搭配現有動作以序列和平行的方式新增動作：

     您也可選擇那些動作的編輯圖示以在此檢視中編輯動作。若要刪除動作，請在該動作上選擇刪除圖示。

   - 若要編輯動作，請選擇該動作的編輯圖示，然後在 Edit action (編輯動作) 上變更值。有星號 (*) 標記的項目為必要項。
     - 對於 CodeCommit 存放庫名稱和分支，會出現一則訊息，顯示要為此管道建立的 Amazon E CloudWatch vents 規則。如果您移除來 CodeCommit 源，會出現一則訊息，顯示要刪除的 Amazon CloudWatch 事件規則。
     - 對於 Amazon S3 來源儲存貯體，會出現一則訊息，顯示要為此管道建立的 Amazon CloudWatch 事件規則和AWS CloudTrail追蹤。如果您移除 Amazon S3 來源，會出現一則訊息，顯示要刪除的 Amazon CloudWatch 活動規則和AWS CloudTrail追蹤。若 AWS CloudTrail 線索正由其他管道使用，該線索將不會移除，且資料事件會被刪除。
   - 若要新增階段，請在您想新增階段的管道中的點選擇 + Add stage (+ 新增階段)。提供該階段名稱，然後為其新增至少一個動作。有星號 (*) 標記的項目為必要項。
   - 若要刪除階段，請在該階段上選擇刪除圖示。該階段與其所有動作都會被刪除。

   例如，如果您想在管道中的階段新增序列動作：

   1. 在您想要新增動作的階段中，選擇 Edit stage (編輯階段)，然後選擇 + Add action group (+ 新增動作群組)。
   2.

在 Edit action (編輯動作) 的 Action name (動作名稱) 中輸入您的動作名稱。Action provider (動作供應商) 清單會依類別顯示供應商選項。尋找類別 (例如 Deploy (部署))。在類別下，選擇提供者 (例如，AWS CodeDeploy)。在「區域」中，選擇建立資源的「AWS區域」或您計劃建立資源的地區。[Re gion] 欄位會指定針對此動作類型和提供者類型建立AWS資源的位置。此欄位僅針對動作提供者為的動作顯示AWS 服務。「區域」(Region) 欄位預設為與管線相同的「AWS區域」。

如需中動作需求的詳細資訊 CodePipeline，包括輸入與輸出人工因素的名稱及其使用方式，請參閱動作結構需求 CodePipeline。如需新增動作供應商以及針對每個供應商使用預設欄位的範例，請參閱建立管道 (主控台)。

若要將建置動作或測試動作新增 CodeBuild 至階段，請參閱使用CodeBuild 者指南中的 CodePipeline 搭配 CodeBuild 測試程式碼和執行組建。

> **Note**
>
> 某些動作提供者 (例如) 會要求您連線至提供者的網站 GitHub，才能完成動作的設定。請務必在連線至供應商的網站時，使用該網站的登入資料。請勿使用您的 AWS 登入資料。

3. 當您完成動作設定時，請選擇 Save (儲存)。

> **Note**
>
> 您不能在主控台檢視中重新命名階段。您可以使用想變更的名稱來新增階段，然後刪除舊的。刪除舊動作之前，請務必確認您已在該階段中新增所有您要的動作。

8. 當您完成管道編輯後，請選擇 Save (儲存) 以返回摘要頁面。

> **Important**
>
> 儲存變更後，您將無法復原變更。您必須再次編輯管道。若您在儲存變更時，有修訂正在透過管道執行，則該執行不會完成。若您想要特定的遞交或變更透過編輯過的管道來執行，您必須透過管道手動執行之。否則，下一個遞交或變更將透過管道自動執行。

9. 若要測試您的動作，請選擇 [發行變更] 以處理透過管線提交的通知，並將變更提交至管線的來源階段中指定的來源。或者，請按照 手動啟動管道 中的步驟以使用 AWS CLI 手動發佈變更。

編輯管道 (AWS CLI)

您可以使用 update-pipeline 命令來編輯管道。

當您更新管線時，會正 CodePipeline 常地完成所有執行中的動作，然後將完成執行中動作的階段和管線執行失敗。管道更新後，您將需要重新執行管道。如需執行管線的詳細資訊，請參閱手動啟動管道。

> **Important**
>
> 雖然您可以用 AWS CLI 來編輯包含合作夥伴動作的管道，但您必須手動編輯合作夥伴動作的 JSON。若您這麼做，該合作夥伴動作會在您更新管道後失敗。

編輯管道

1. 開啟終端機工作階段 (Linux、macOS 或 Unix) 或命令提示字元 (Windows)，然後執行get-pipeline指令，將管線結構複製到 JSON 檔案中。例如，針對名為 **MyFirstPipeline** 的管道輸入下列命令：

   ```
   aws codepipeline get-pipeline --name MyFirstPipeline >pipeline.json
   ```

   此命令不會傳回任何內容，但您建立的檔案應該會顯示在您執行命令的目錄中。
2. 以讓和純文字編輯器開啟 JSON 檔案，並修改該檔案的結構以反映您要在管道上進行的變更。例如，您可以新增或移除階段，或新增另一個動作至現有階段。

   下列範例會示範如何在管線 .json 檔案中新增另一個部署階段。此階段將在第一個部署階段 (*Staging (##)*) 後執行。

   > **Note**
   >
   > 此僅為該檔案的一部分，而非整個結構。如需詳細資訊，請參閱 CodePipeline 配管結構參照。

   ```
   ,
           {
               "name": "Staging",
               "actions": [
                       {
   ```

```
                    "inputArtifacts": [
                        {
                            "name": "MyApp"
                        }
                    ],
                    "name": "Deploy-CodeDeploy-Application",
                    "actionTypeId": {
                        "category": "Deploy",
                        "owner": "AWS",
                        "version": "1",
                        "provider": "CodeDeploy"
                    },
                    "outputArtifacts": [],
                    "configuration": {
                        "ApplicationName": "CodePipelineDemoApplication",
                        "DeploymentGroupName": "CodePipelineDemoFleet"
                    },
                    "runOrder": 1
                }
            ]
        },
    {
        "name": "Production",
        "actions":  [
                {
                    "inputArtifacts": [
                        {
                            "name": "MyApp"
                        }
                    ],
                    "name": "Deploy-Second-Deployment",
                    "actionTypeId": {
                        "category": "Deploy",
                        "owner": "AWS",
                        "version": "1",
                        "provider": "CodeDeploy"
                    },
                    "outputArtifacts": [],
                    "configuration": {
                    "ApplicationName": "CodePipelineDemoApplication",
                    "DeploymentGroupName": "CodePipelineProductionFleet"
                    },
                    "runOrder": 1
                }
```

```
                ]
            }
        ]
}
```

如需有關使用 CLI 為管道新增核准動作的資訊，請參閱 [將手動核准動作新增至 CodePipeline 中的管道](#)。

請確認您 JSON 檔案中的 PollForSourceChanges 參數已如下所示進行設定：

```
                "PollForSourceChanges": "false",
```

CodePipeline 使用 Amazon CloudWatch 事件偵測 CodeCommit 來源儲存庫和分支機構或 Amazon S3 來源儲存貯體中的變更。下一步驟包括了手動建立這些資源的說明。將旗標設為 false 將停用定期檢查，在使用建議的變更偵測方法時，您不需要該項功能。

3. 若要在與您的管道不同的區域中新增建置、測試或部署動作，您必須將以下項目新增到管道結構。如需詳細說明，請參閱 [在 CodePipeline 中新增跨區域動作](#)。

   - 將 Region 參數新增到動作的管道結構。
   - 使用 artifactStores 參數，為您在其中具有動作的每個區域指定成品儲存貯體。

4. 如果您使用的是使用 get-pipeline 命令擷取的管道結構，您必須在 JSON 檔案中修改結構。您必須從檔案移除 metadata 行，以讓 update-pipeline 命令可以使用它。從 JSON 檔案的管道結構移除此區段 ("metadata": { } 行以及 "created"、"pipelineARN" 和 "updated" 欄位)。

   例如，從結構中移除下列幾行：

```
"metadata": {
  "pipelineArn": "arn:aws:codepipeline:region:account-ID:pipeline-name",
  "created": "date",
  "updated": "date"
  }
```

   儲存檔案。

5. 如果您使用 CLI 編輯管道，您必須為管道手動管理建議的變更偵測資源：

   - 對於 CodeCommit 存放庫，您必須建立 CloudWatch 事件規則，如中所述[建立 CodeCommit 來源 (CLI) 的 EventBridge 規則](#)。

- 對於 Amazon S3 來源，您必須建立 CloudWatch 事件規則和AWS CloudTrail追蹤，如中所述Amazon S3 來源動作 EventBridge 與 AWS CloudTrail。

6. 若要套用您的變更，請執行 update-pipeline 命令、指定管道 JSON 檔案：

> **Important**
>
> 請確認在檔案名稱之前包含 `file://`。這是此命令必要項目。

```
aws codepipeline update-pipeline --cli-input-json file://pipeline.json
```

此命令會傳回所編輯管道的整個結構。

> **Note**
>
> update-pipeline 命令將終止管道。若在您執行 update-pipeline 命令時有修訂正在透過管道執行，該執行將停止。您必須手動啟動管道，以透過更新的管道執行該修訂。

7. 打開 CodePipeline 控制台，然後選擇剛剛編輯的管道。

此管道會顯示您的變更。下次您變更來源位置時，管道會透過修訂過的管道結構來執行該修訂。

8. 若要透過修訂的管道結構手動執行最新的修訂，請執行 start-pipeline-execution 命令。如需詳細資訊，請參閱 手動啟動管道。

有關管道結構和預期值的詳細資訊，請參閱CodePipeline 配管結構參照和 AWS CodePipelineAPI 參考。

在中檢視管線和詳細資訊 CodePipeline

您可以使用 AWS CodePipeline 主控台或 AWS CLI 來檢視與您的 AWS 帳戶關聯的管道詳細資訊。

主題

- 檢視管線 (主控台)
- 檢視管線中的動作詳細資訊 (主控台)
- 檢視管線 ARN 和服務角色 ARN (主控台)
- 檢視管道詳細資訊與歷程記錄 (CLI)

檢視管線 (主控台)

您可以檢視管道的狀態、轉換和成品更新。

> **Note**
>
> 一小時後，您瀏覽器中的管道詳細資訊檢視會自動停止重新整理。若要檢視目前資訊，請重新整理頁面。

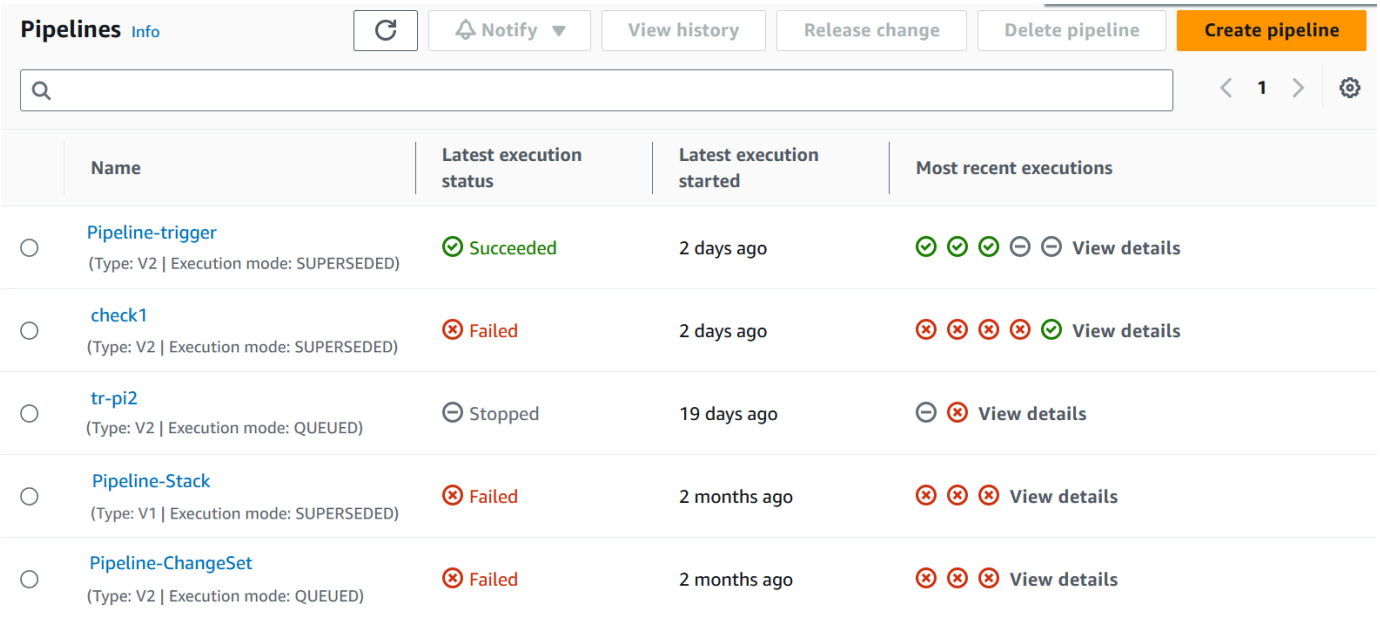
檢視配管

1. 請登入AWS Management Console並開啟 CodePipeline 主控台，網址為 http://console.aws.amazon.com/codesuite/codepipeline/home。

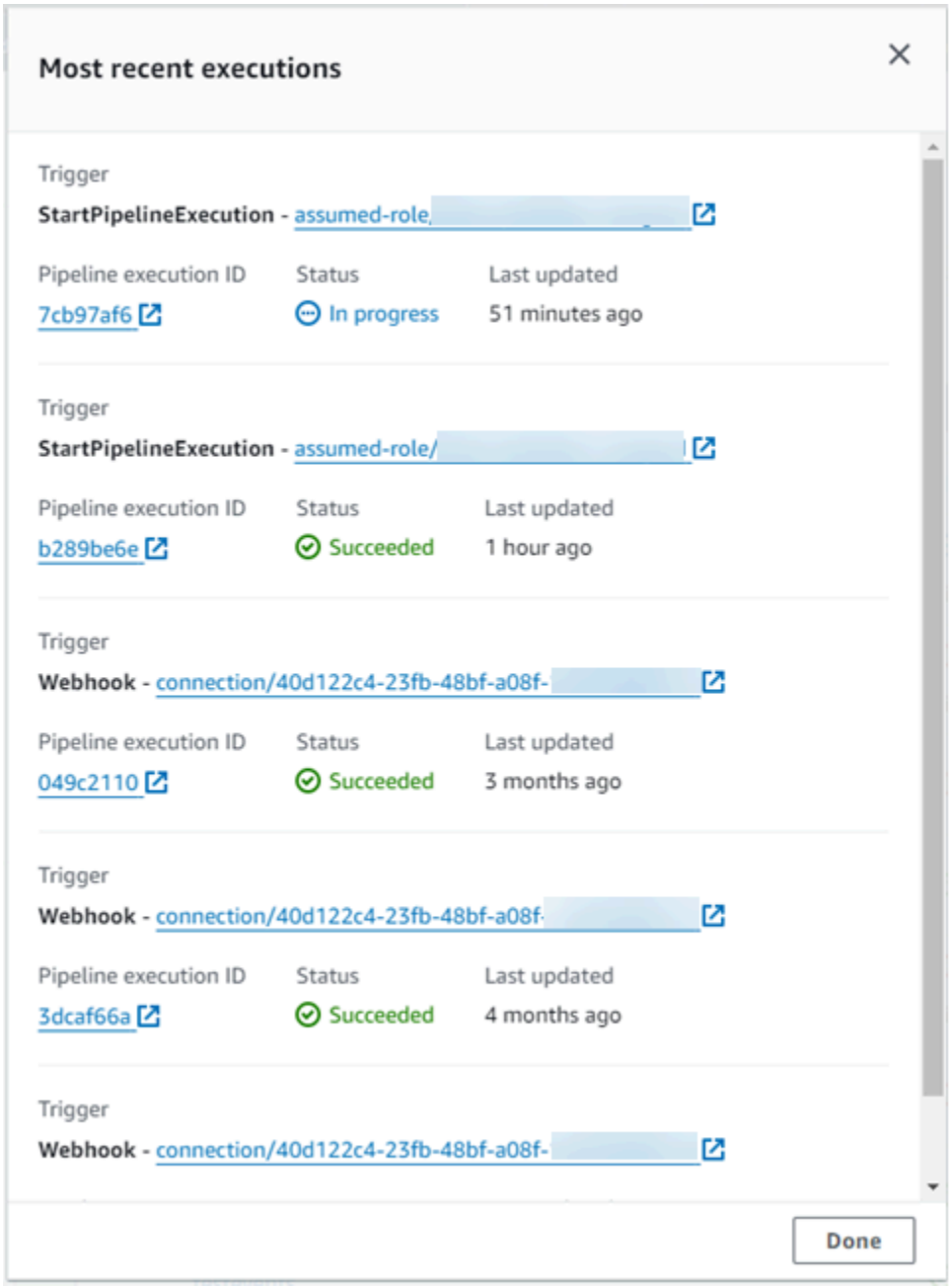
   「配管」頁面隨即顯示。此時會顯示該區域的所有管線清單。

   系統會顯示與您AWS帳戶相關聯的所有管道的名稱、類型、狀態、版本、建立日期和上次修改日期，以及最近開始的執行時間。

2. 顯示最近五個執行的狀態。

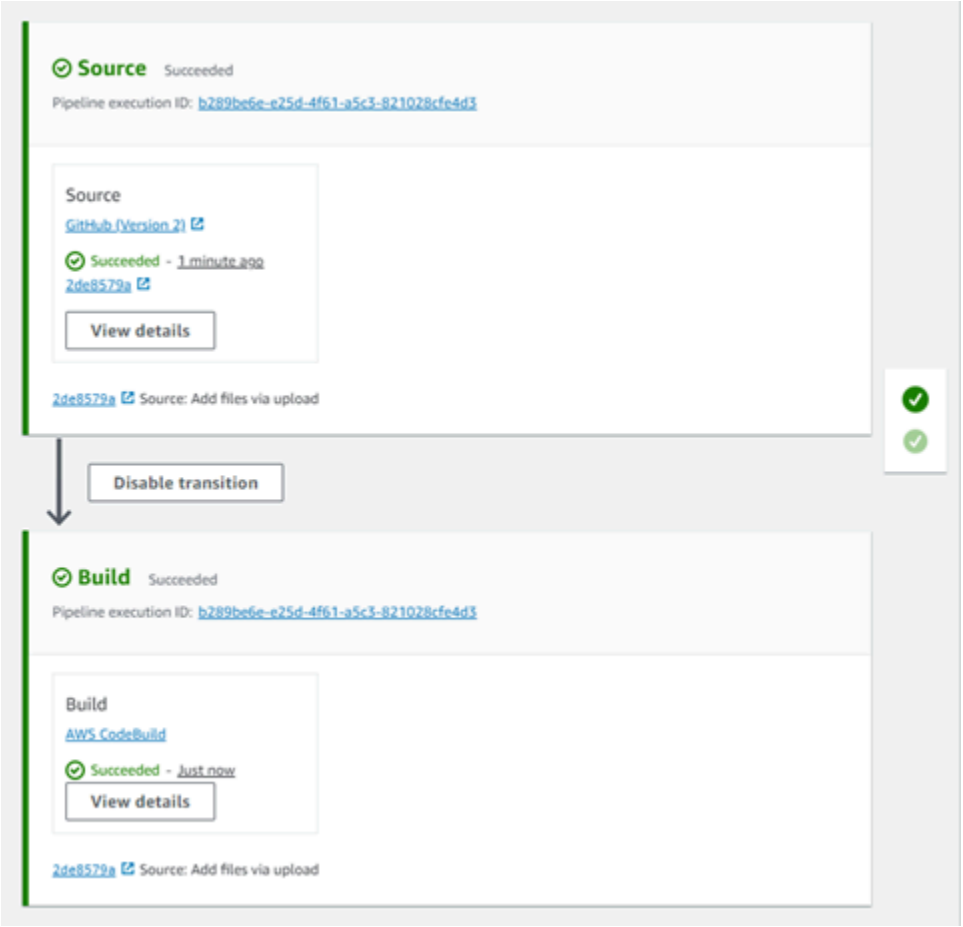


   選擇特定資料列旁的 [檢視詳細資料]，以顯示列出最近執行項目的詳細資料對話方塊。



若要檢視管線最近執行的詳細資訊，您也可以選擇管線旁邊的選取器，然後選擇 [檢視歷程記錄]。對於過去的執行，您可以檢視與來源成品相關的修訂詳細資訊，例如執行 ID、狀態、開始與結束時間、持續時間以及遞交 ID 及訊息。

3. 若要查看單一管道詳細資訊，請在 Name (名稱) 中選擇該管道。管道的詳細檢視，包含各階段的每項動作以及轉換的狀態將會顯示。



該圖形化檢視顯示了每一階段的下列資訊：

- 階段名稱。
- 為該階段設定的每一個動作。
- 階段之間的轉換階段 (啟用或停用)，如階段之間的箭頭所表示的狀態。啟用的轉換由箭頭指示，旁邊有 Disable transition (停用轉換) 按鈕。停用的轉換由下方有一個刪除線的箭頭指示，旁邊有一個 Enable transition (啟用轉換) 按鈕。
- 指示階段狀態的顏色條：
  - 灰色：尚未執行
  - 藍色：處理中
  - 綠色：成功
  - 紅色：失敗

該圖形化檢視也顯示了每一階段的下列動作資訊：

- 該動作的名稱。
- 動作的提供者，例如 CodeDeploy。
- 動作最後執行的時間。
- 不論動作成功或失敗。
- 有關最後一個動作執行 (若有的話) 的其他詳細資訊連結。
- 有關透過階段中最新管線執行執行的來源修訂版本，或 CodeDeploy 部署至目標執行個體的最新來源修訂版本 (針對部署) 的詳細資料。
- [檢視詳細資料] 按鈕可開啟對話方塊，其中包含動作執行、記錄和動作設定的詳細資訊。

> **Note**
>
> [記錄] 索引標籤可用於管線帳戶中已執行的 CodeBuild 和AWS CloudFormation動作。

4. 若要檢視動作提供者的詳細資訊，請選擇提供者。例如，在前面的範例管線中，如果您 CodeDeploy 在「暫存」或「生產」階段中選擇，則會顯示為該階段設定之建置群組的「 CodeDeploy 主控台」頁面。
5. 若要查看動作的進度，則會顯示在進行中的動作旁邊 (由「進行中」訊息指示)。若該動作正在進行中，您會看到逐漸增加的進度與正在進行的步驟或動作。
6. 若要核准或拒絕為手動核准所設定的動作，請選擇 Review (檢視)。
7. 若要重試階段中未成功完成的動作，請選擇 Retry (重試)。
8. 會顯示上次執行動作的狀態，包括該動作的結果 (「成功」或「失敗」)。

檢視管線中的動作詳細資訊 (主控台)

您可以檢視管道的詳細資訊，包括每個階段中動作的詳細資訊。

> **Note**
>
> 一小時後，您瀏覽器中的管道詳細資訊檢視會自動停止重新整理。若要檢視目前資訊，請重新整理頁面。

若要檢視管線中的動作詳細資訊

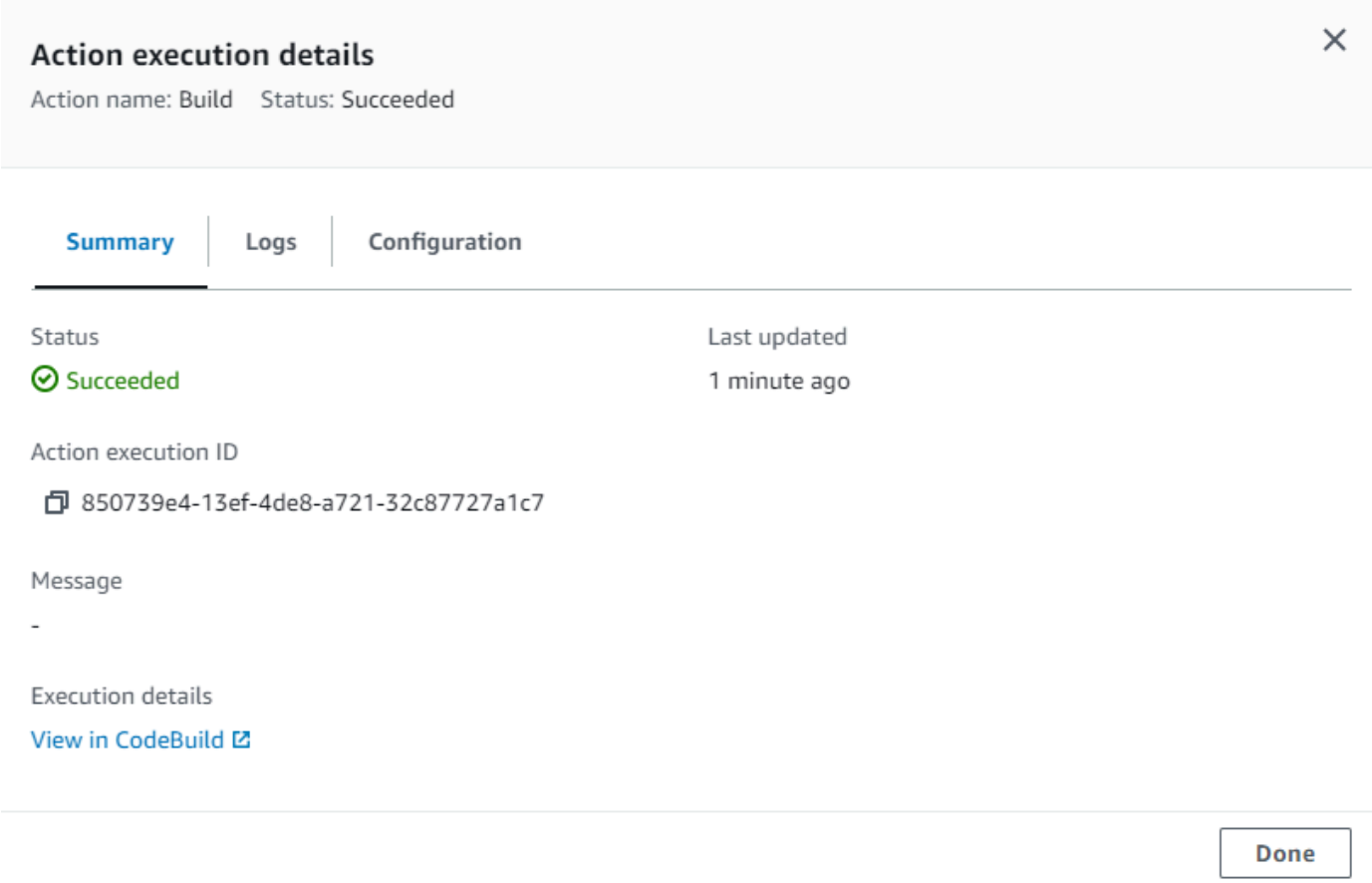
1. 請登入AWS Management Console並開啟 CodePipeline 主控台，網址為 http://console.aws.amazon.com/codesuite/codepipeline/home。

   「配管」頁面隨即顯示。

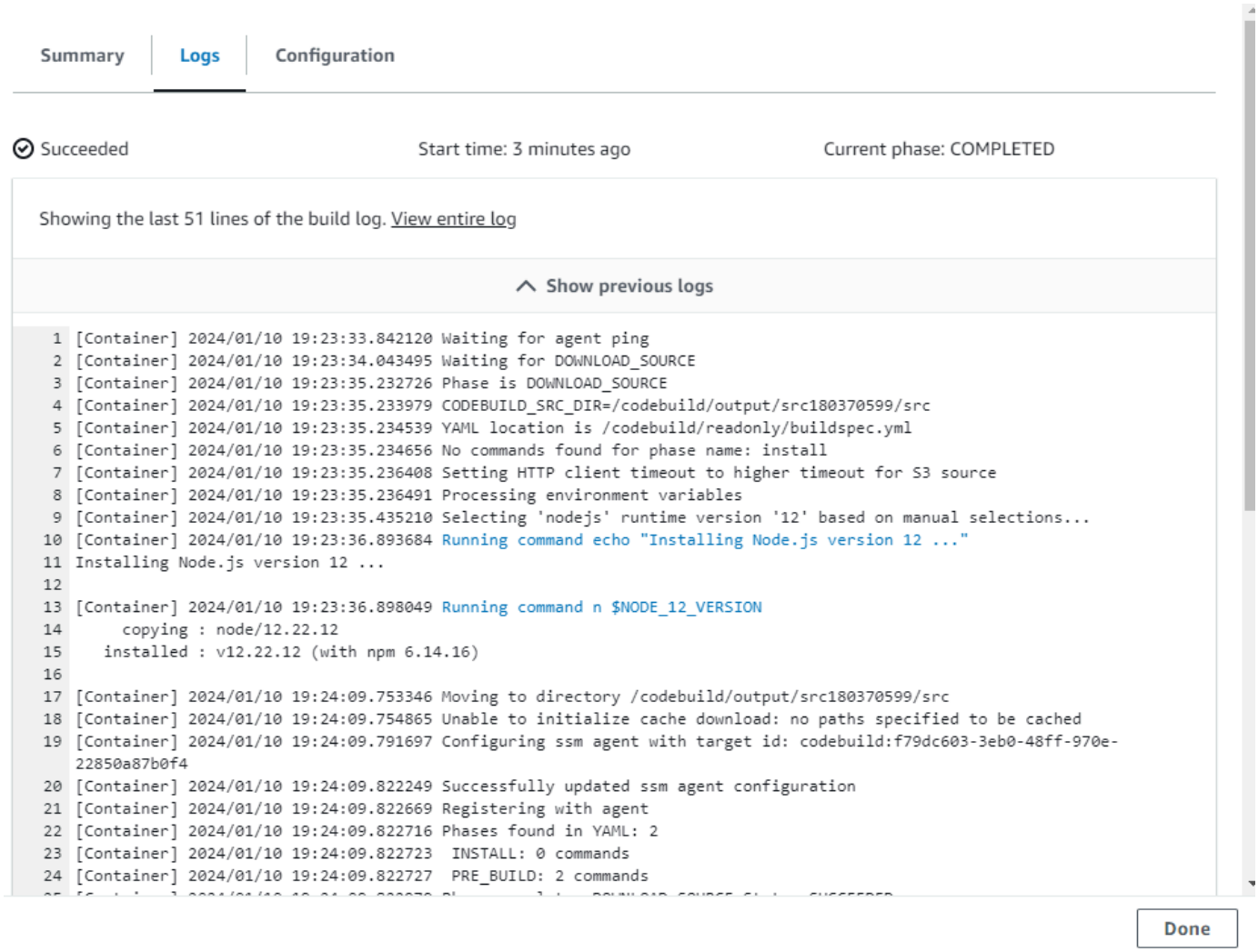
2. 在任何動作上，選擇 [檢視詳細資料] 開啟對話方塊，其中包含動作執行、記錄和動作設定的詳細資訊。

   > **Note**
   >
   > [記錄] 索引標籤可用於 CodeBuild 和AWS CloudFormation動作。

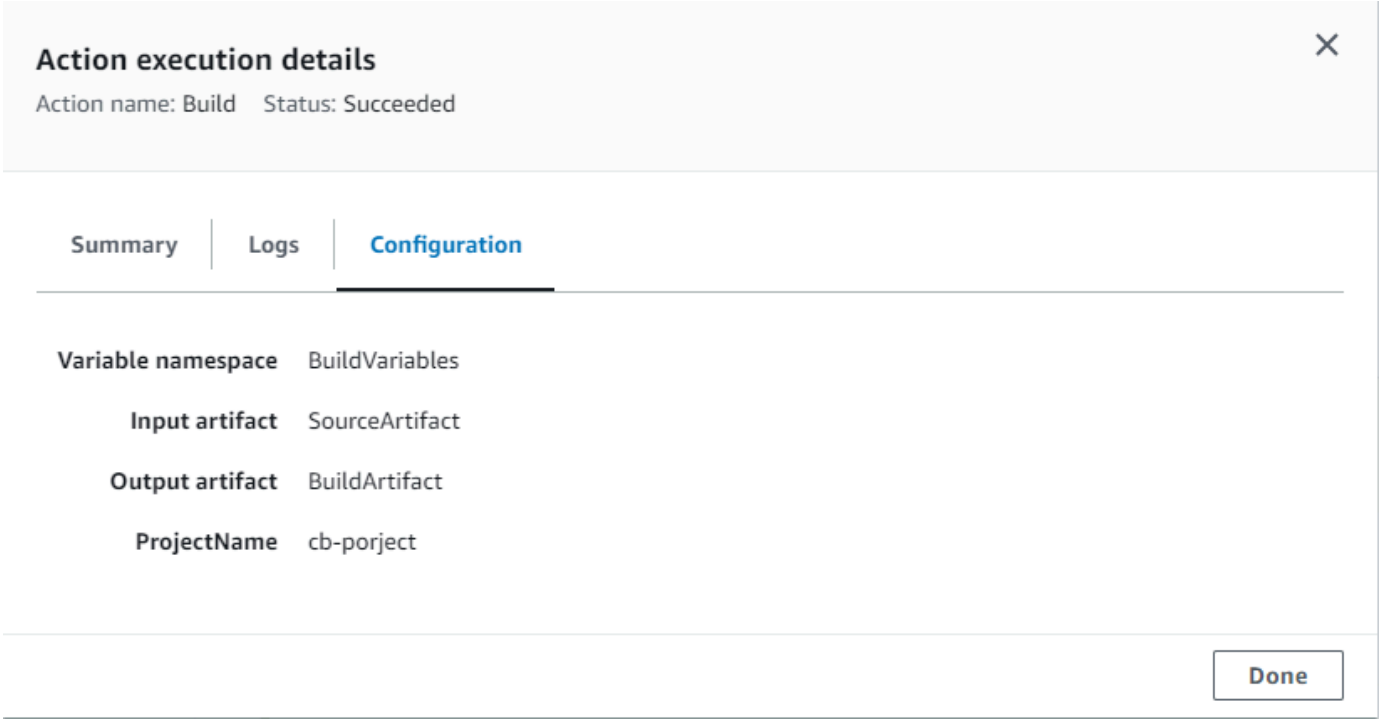
3. 若要查看管線階段中動作的動作摘要，請選擇動作上的 [檢視詳細資訊]，然後選擇 [摘要] 索引標籤。



4. 若要查看具有記錄檔之動作的處理行動記錄檔，請選擇「檢視動作的詳細資料」，然後選擇「記錄檔」索引標籤。



5. 若要查看動作的組態詳細資訊，請選擇「組態」索引標籤。



檢視管線 ARN 和服務角色 ARN (主控台)

您可以使用主控台來檢視管線設定，例如管線 ARN、服務角色 ARN 和管線人工因素存放區。

1. 請登入AWS Management Console並開啟 CodePipeline 主控台，網址為 http://console.aws.amazon.com/codesuite/codepipeline/home。

   將會顯示所有與您 AWS 帳戶建立關聯的管道名稱。
2. 選擇管道的名稱，然後在左側導覽窗格中選擇 [設定]。此頁面會顯示下列項目：

   - 管線名稱
   - 管道 Amazon 資源名稱（ARN）

     管道 ARN 是以下列格式建構：

     AR: aws: #####:##:##:####

     範例管道 ARN：

     `arn:aws:codepipeline:us-east-2:80398EXAMPLE:MyFirstPipeline`
   - 管道的 CodePipeline 服務角色 ARN

- 管道版本
- 管線的人工因素存放區的名稱和位置

檢視管道詳細資訊與歷程記錄 (CLI)

您可以執行下列命令來檢視關於管道與管道執行的詳細資訊：

- list-pipelines 命令，用以檢視所有與您的 AWS 帳戶相關之管道摘要。
- get-pipeline 命令，用以審閱單一管道的詳細資訊。
- list-pipeline-executions 以檢視最近期管道執行的摘要。
- get-pipeline-execution 以檢視管道執行相關資訊，包含成品詳細資訊、管道執行 ID、管道名稱、版本及狀態。
- get-pipeline-state 命令以檢視管道、階段和動作狀態。
- list-action-executions 以檢視管道的動作執行詳細資訊。

1. 開啟終端機 (Linux、macOS 或 Unix) 或命令提示字元 (視窗)，然後使用AWS CLI來執行list-pipelines命令：

   ```
   aws codepipeline list-pipelines
   ```

   此命令會傳回與您的 AWS 帳戶相關的所有管道清單。

2. 若要檢視管道相關詳細資訊，請執行 get-pipeline 命令，指定管道的獨特名稱。例如，若要檢視名為 *MyFirstPipeline* 管道的詳細資訊，請輸入下列內容：

   ```
   aws codepipeline get-pipeline --name MyFirstPipeline
   ```

   此命令會傳回管道結構。

在 CodePipeline 中刪除管道

您可隨時編輯管道以變更其功能，但您也可以決定是否刪除它。您可以使用 AWS CodePipeline 主控台或 AWS CLI 中的 delete-pipeline 來刪除管道。

主題

刪除管道 (主控台)

刪除管道

1. 登入AWS Management Console開啟 CodePipeline 主控台，前往http://console.aws.amazon.com/codesuite/codepipeline/home。

   所有與您的 AWS 帳戶相關的管道名稱與狀態都會顯示。
2. 在 Name (名稱) 中，選擇您想刪除的管道名稱。
3. 在管道詳細資訊頁面上，選擇 Edit (編輯)。
4. 在 Edit (編輯) 頁面上，選擇 Delete (刪除)。
5. 在欄位中輸入 **`delete`** 以確認，然後選擇 Delete (刪除)。

> **Important**
> 這個操作無法復原。

刪除管道 (CLI)

若要使用 AWS for WordPressAWS CLI若要手動刪除管道，請使用刪除管道命令。

> **Important**
> 管道刪除後是無法復原的。沒有確認對話方塊。在命令執行後，該管道會被刪除，但管道中使用的資源都不會被刪除。這可讓您輕鬆建立新管道，這些管道會使用那些資源來自動化發佈您的軟體。

刪除管道

1. 開啟終端機 (Linux、macOS 或 Unix) 或命令提示 (Windows)，然後使用AWS CLI執行delete-pipeline命令，指定您想刪除的管道名稱。例如，若要刪除名為的管道*`MyFirstPipeline`*：

```
aws codepipeline delete-pipeline --name MyFirstPipeline
```

此命令不會傳回任何結果。

2. 刪除您不再使用的資源。

> **Note**
>
> 刪除管道並不會刪除管道中使用的資源 (例如您用來部署程式碼的 CodeDePipeline 或 Elastic Beanstalk 應用程式)，或者，如果您從 CodePipeline 主控台建立管道，Amazon S3 會建立，以供您存放管道成品。請務必刪除您不再使用的資源，才不會在日後繼續為其付費。例如，當您第一次使用主控台建立管道時，CodePipeline 將建立一個 Amazon S3 存貯體以存放所有管道的所有成品。如果您已刪除所有管道，請依照刪除儲存貯體中的步驟進行。

在 CodePipeline 中建立使用來自另一個 AWS 帳戶資源的管道

您可能希望建立一個管道，該管道使用由另一個 AWS 帳戶建立或管理的資源。例如，您可能想要為管道使用一個帳戶，並為 CodeDeploy 資源使用另一個帳戶。

> **Note**
>
> 使用來自多個帳戶的動作建立管道時，您必須設定帳戶，讓它們在跨帳戶管道的限制內仍可存取成品。以下限制適用於跨帳戶動作：
>
> - 通常，若為下列情況，動作只能取用一個成品：
>   - 動作與管道帳戶 OR 位於同一帳戶
>   - 已在道管帳戶中為另一個帳戶 OR 中的動作建立成品
>   - 與此動作位於同一帳戶的前一動作已產生成品
>
>   換言之，您無法將成品從一個帳戶傳遞至另一個帳戶，如果任一帳戶都不是管道帳戶的話。
> - 以下動作類型不支援跨帳戶動作：
>   - Jenkins 建置動作

例如，您必須建立要使用的 AWS Key Management Service (AWS KMS) 金鑰，新增金鑰至管道，然後設定帳戶政策和角色以啟用跨帳戶存取。如果是 AWS KMS 金鑰，您可以使用金鑰 ID、金鑰 ARN 或別名 ARN。

> **Note**
>
> 只會在建立 KMS 金鑰的帳戶中辨識別名。如果是跨帳戶動作，您只可以使用金鑰 ID 或金鑰 ARN 來識別金鑰。跨帳戶動作涉及使用來自其他帳戶 (AccountB) 的角色，因此指定金鑰 ID 將使用其他帳戶 (AccountB) 中的金鑰。

在此逐步教學和其範例中，*AccountA* 是最初用於建立管道的帳戶。它具備用於儲存管道成品的 Amazon S3 儲存貯體和使用的服務角色AWS CodePipeline。*AccountB* 帳戶最初是用於建立 CodeDeploy 應用程式、部署群組以及 CodeDeploy 所使用的服務角色。

Account A A ######### Account B ###CodeDeploy#####Accoun tA TA ###

- 請求 *AccountB* 的 ARN 或帳戶 ID (在此逐步教學中，*AccountB* ID 為 *012ID_ACCOUNT_B*)。
- *############AWS KMS###################### (CodePipeline_Service_ Role) # AccountB ####*
- 建立一個 Amazon S3 儲存貯體政策，以授予 *AccountB* 存取 Amazon S3 儲存貯體的權限 (例如，*codepipeline-us-east-2-* 1234567890)。
- 建立允許 *AccountA ### Account B* 設定的角色的原則，並將該原則附加至服務角色 (*CodePipeline_Service_* Role)。
- 編輯管道以使用客戶受管的 AWS KMS 金鑰，而非預設金鑰。

若要讓 *AccountB* 允許其資源存取在 *AccountA* 中建立的管道，*AccountB* 必須：

- 請求 *AccountA* 的 ARN 或帳戶 ID (在此逐步教學中，*AccountA* ID 為 *012ID_ACCOUNT_A*)。
- 建立套用至針對設定的 Amazon EC2 執行個體角色的政策CodeDeploy，允許存取 Amazon S3 儲存貯體 (*codepipeline-us-east-2-123* 4567890)。
- 建立套用至設定的 Amazon EC2 執行個體角色的政策CodeDeploy，允許存取用於在 *Account* A 中加密管道成品的AWS KMS客戶受管金鑰。
- 透過信任關係政策來設定和附加 IAM *CrossAccount## (_Role*)，該政策允許 *Account* A 擔任該角色。
- 建立允許存取管線所需之部署資源的原則，並將其附加至 *CrossAccount_Role*。

- *###### Amazon S3 #### (codepipeline-us-east-2-1234567890) ########## _Role#CrossAccount*

主題

必要條件：建立 AWS KMS 加密金鑰

客戶受管金鑰是專屬於特定區域的，與所有 AWS KMS 金鑰相同。您必須在管道建立的相同區域 (例如 us-east-2) 中建立客戶受管 AWS KMS 金鑰。

如何在 AWS KMS 中建立客戶受管金鑰

1. AWS Management Console使用*## A ####*開AWS KMS控制台。
2. 在左側選擇 Customer managed keys (客戶受管金鑰)。
3. 選擇 Create key (建立金鑰)。在 Configure key (設定金鑰) 中，保持 Symmetric (對稱) 預設值的選取狀態，然後選擇 Next (下一步)。
4. 在別名中，輸入要用於此機碼的別名 (例如 *PipelineName-Key*)。為此金鑰提供說明和標籤 (選用)，然後選擇 Next (下一步)。
5. 在 [定義金鑰系統管理權限] 中，針對此機碼選擇您要作為管理員的一個或多個角色，然後選擇 [下一步]。
6. 在「定義金鑰使用權限」的「此帳戶」下，選取管線的服務角色名稱 (例如 CodePipeline_Service_Role)。在「其他AWS帳戶」下，選擇「新增其他AWS帳戶」 為 *AccountB* 輸入帳戶 ID 以完成 ARN，然後選擇 Next (下一步)。
7. 在 Review and edit key policy (檢閱和編輯金鑰政策) 中檢閱政策，然後選擇 Finish (完成)。
8. 從金鑰清單中選擇您的金鑰別名，並複製其 ARN (例如 ***arn:aws:kms:us-east-2:012ID_ACCOUNT_A:key/2222222-3333333-4444-556677EXAMPLE***)。在您編輯管道和設定政策時將需要用到這項資料。

步驟 1：設定帳戶政策與角色

建立 AWS KMS 金鑰後，您必須建立並連接可啟用跨帳戶存取的政策。這將需要 *AccountA* 和 *AccountB* 的動作。

主題

- 在建立管道的帳戶 (AccountA) 中設定政策和角色
- 在擁有 AWS 資源的帳戶 (AccountB) 中設定政策和角色

在建立管道的帳戶 (*AccountA*) 中設定政策和角色

若要建立使用與其他AWS帳戶關聯之CodeDeploy資源的管道，*Account* A 必須為用於存放成品的 Amazon S3 儲存貯體和的服務角色設定政策。CodePipeline

為授予 AccountB (主控台) 存取權的 Amazon S3 儲存貯體建立政策

1. 登入使AWS Management Console用 *AccountA*，然後 Amazon S3 位於 https://console.aws.amazon.com/s3/。
2. 在 Amazon S3 儲存貯體清單中，選擇存放管道成品的 Amazon S3 儲存貯體。*#####codepipeline-region-1234567EXAMPLE################AWS####1234567EXAMPLE ########################-2-1234567890##codepipeline-us-east*
3. 在 Amazon S3 儲存貯體的詳細資料頁面上，選擇屬性。
4. 在屬性窗格中，展開 Permissions (許可)，然後選擇 Add bucket policy (新增儲存貯體政策)。

   > **Note**
   >
   > 如果政策已附加至 Amazon S3 儲存貯體，請選擇編輯儲存貯體政策。然後，您可以按以下範例新增陳述式至現有政策。若要新增政策，請選擇連結，然後按照 AWS 政策產生器中的說明進行。如需詳細資訊，請參閱 IAM 政策概觀。

5. 在 Bucket Policy Editor (儲存貯體政策編輯器) 視窗中，輸入以下政策。這可讓 *AccountB* 存取管道成品，並授與 *AccountB* 在動作 (例如自訂來源或建置動作) 建立輸出成品時將其新增的能力。

   在下列範例中，*AccountB* 的 ARN 為 *012ID_ACCOUNT_B* 。亞馬遜存儲桶的 ARN 是 *codepipeline-us-east-* 2-1234567890。將這些 ARN 取代為您要允許存取的帳戶的 ARN 和 Amazon S3 儲存貯體的 ARN：

```
{
 "Version": "2012-10-17",
 "Id": "SSEAndSSLPolicy",
 "Statement": [
{
  "Sid": "DenyUnEncryptedObjectUploads",
  "Effect": "Deny",
  "Principal": "*",
  "Action": "s3:PutObject",
  "Resource": "arn:aws:s3:::codepipeline-us-east-2-1234567890/*",
  "Condition": {
 "StringNotEquals": {
 "s3:x-amz-server-side-encryption": "aws:kms"
 }
   }
 },
{
  "Sid": "DenyInsecureConnections",
  "Effect": "Deny",
  "Principal": "*",
  "Action": "s3:*",
  "Resource": "arn:aws:s3:::codepipeline-us-east-2-1234567890/*",
  "Condition": {
 "Bool": {
    "aws:SecureTransport": false
  }
 }
  },
{
  "Sid": "",
  "Effect": "Allow",
  "Principal": {
 "AWS": "arn:aws:iam::012ID_ACCOUNT_B:root"
 },
  "Action": [
           "s3:Get*",
           "s3:Put*"
      ],
  "Resource": "arn:aws:s3:::codepipeline-us-east-2-1234567890/*"
},
{
  "Sid": "",
  "Effect": "Allow",
```

```
    "Principal": {
        "AWS": "arn:aws:iam::012ID_ACCOUNT_B:root"
    },
    "Action": "s3:ListBucket",
    "Resource": "arn:aws:s3:::codepipeline-us-east-2-1234567890"
  }
    ]
}
```

6. 選擇 Save (儲存)，然後關閉政策編輯器。
7. 選擇儲存 Amazon S3 儲存貯體的許可。

為 CodePipeline 服務角色建立政策 (主控台)

1. 使AWS Management Console用 *AccountA* #登入，然後開啟位於 https://console.aws.amazon.com/iam/ 開啟 IAM 主控台。
2. 在導覽窗格中，選擇 Roles (角色)。
3. 在角色清單的 Role Name (角色名稱) 下，選擇 CodePipeline 的服務角色名稱。
4. 在許可標籤上，選擇新增內嵌政策。
5. 選擇 [JSON] 索引標籤，然後輸入下列原則，以允許 *AccountB* 擔任該角色。在下列範例中，*012ID_ACCOUNT_B* 為 *AccountB* 的 ARN：

```
{
    "Version": "2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": "sts:AssumeRole",
        "Resource": [
            "arn:aws:iam::012ID_ACCOUNT_B:role/*"
        ]
    }
  }
```

6. 選擇 Review policy (檢閱政策)。
7. 在 Name (名稱) 中，輸入此政策的名稱。選擇 建立政策。

在擁有 AWS 資源的帳戶 (*AccountB*) 中設定政策和角色

在中建立應用程式、部署和部署群組時CodeDeploy，也會建立 Amazon EC2 執行個體角色。(若您使用執行部署逐步說明精靈，將為您建立此角色，但您也可以手動建立角色。) 若要讓在 *Account* A 中建立的管道使用在 Ac *countB* 中建立的CodeDeploy資源，您必須：

- 設定執行個體角色的政策，以便其存取存放管道成品的 Amazon S3 儲存貯體。
- 在為跨帳戶存取設定的 *AccountB* 中建立第二個角色。

  第二個角色不僅必須能夠存取 *Account* A 中的 Amazon S3 儲存貯體，還必須包含允許存取CodeDeploy資源的政策，以及允許 *Account* A 擔任該角色的信任關係政策。

> **Note**
>
> 這些政策專用於設定 CodeDeploy 資源，以供使用不同 AWS 帳戶建立的管道運用。其他 AWS 資源將需要針對資源要求所制定的政策。

若要為 CodeDeploy (主控台) 設定的 Amazon EC2 執行個體角色建立政策

1. 登入使AWS Management Console用 *AccountB*，然後開啟位於 https://console.aws.amazon.com/iam/ 開啟 IAM 主控台。
2. 在導覽窗格中，選擇 Roles (角色)。
3. 在角色清單中的角色名稱下，選擇用作CodeDeploy應用程式之 Amazon EC2 執行個體角色的服務角色名稱。此角色名稱可能不同，且部署群組可使用一個以上的執行個體角色。如需詳細資訊，請參閱建立 Amazon EC2 執行個體描述檔。
4. 在許可標籤上，選擇新增內嵌政策。
5. 選擇 JSON 索引標籤，然後輸入下列政策，以授與 *AccountA* 用來存放管道成品之 Amazon S3 儲存貯體的存取權 (在此範例中為 *codepipeline-us-east-2-* 1234567890)：

```
{
   "Version": "2012-10-17",
   "Statement": [
     {
       "Effect": "Allow",
       "Action": [
         "s3:Get*"
       ],
```

```
        "Resource": [
          "arn:aws:s3:::codepipeline-us-east-2-1234567890/*"
        ]
      },
      {
        "Effect": "Allow",
        "Action": [
          "s3:ListBucket"
        ],
        "Resource": [
          "arn:aws:s3:::codepipeline-us-east-2-1234567890"
        ]
      }
    ]
 }
```

6. 選擇 Review policy (檢閱政策)。
7. 在 Name (名稱) 中，輸入此政策的名稱。選擇 建立政策。
8. 建立 AWS KMS 的第二項政策，其中 ***arn:aws:kms:us-east-1:012ID_ACCOUNT_A:key/2222222-3333333-4444-556677EXAMPLE*** 為在 *AccountA* 中建立，並且允許 *AccountB* 使用的客戶受管金鑰 ARN：

```
{
    "Version": "2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Action": [
           "kms:DescribeKey",
           "kms:GenerateDataKey*",
           "kms:Encrypt",
           "kms:ReEncrypt*",
           "kms:Decrypt"
         ],
        "Resource": [
           "arn:aws:kms:us-
east-1:012ID_ACCOUNT_A:key/2222222-3333333-4444-556677EXAMPLE"
          ]
      }
   ]
}
```

> **Important**
>
> 您必須在此政策中使用 *AccountA* 的帳戶 ID，做為 AWS KMS 金鑰的資源 ARN 一部分，如此處所示，否則政策無法運作。

9. 選擇 Review policy (檢閱政策)。
10. 在 Name (名稱) 中，輸入此政策的名稱。選擇 建立政策。

現在，建立用於跨帳戶存取的 IAM 角色，並進行設定，以便 *AccountA* *####*該角色。此角色必須包含允許存取用於在 *AccountA* 中存放成品的CodeDeploy資源和 Amazon S3 儲存貯體的政策。

在 IAM 中設定跨帳戶角色

1. 登入使AWS Management Console用 *AccountB*，然後開啟位於 https://console.aws.amazon.com/iam 開啟 IAM 主控台。
2. 在導覽窗格中，選擇 Roles (角色)。選擇 Create Role (建立角色)。
3. 在 Select type of trusted entity (選取信任的實體類型) 下，選擇 Another AWS account (另一個 帳戶)。在 [指定可使用此角色的帳戶] 底下的 [帳戶 ID] 中，輸入將在 CodePipeline (*AccountA*) 中建立管道的帳戶的帳戶 ID，然後選擇 [下一步：權限]。AWS

> **Note**
>
> 此步驟建立在 *AccountB* 與 *AccountA* 之間的信任關係政策。

4. 在 [附加權限原則] 下，選擇 [AmazonS3]ReadOnlyAccess，然後選擇 [下一步:標籤]。

> **Note**
>
> 這並非您將使用的政策。您必須選擇一個政策以完成精靈。

5. 選擇 下一步：檢閱 。在角色名稱中輸入此角色的名稱 (例如，*CrossAccount_Role*)。只要它遵循 IAM 中的命名慣例，就可以將此角色命名為任何想要的名稱。可考慮給予角色清楚說明其用途的名稱。選擇 Create Role (建立角色)。
6. 從角色清單中選擇剛建立的角色 (例如 *CrossAccount_Role*)，以開啟該角色的 [摘要] 頁面。
7. 在許可標籤上，選擇新增內嵌政策。
8. 選擇 JSON 索引標籤，然後輸入下列原則以允許存取CodeDeploy資源：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codedeploy:CreateDeployment",
        "codedeploy:GetDeployment",
        "codedeploy:GetDeploymentConfig",
        "codedeploy:GetApplicationRevision",
        "codedeploy:RegisterApplicationRevision"
      ],
      "Resource": "*"
    }
  ]
}
```

9. 選擇 Review policy (檢閱政策)。
10. 在 Name (名稱) 中，輸入此政策的名稱。選擇 建立政策。
11. 在許可標籤上，選擇新增內嵌政策。
12. 選擇 JSON 索引標籤，然後輸入下列政策，以允許此角色從 *AccountA* 中擷取輸入成品，並將輸出成品放入 Amazon S3 儲存貯體：

```
{
   "Version": "2012-10-17",
   "Statement": [
     {
       "Effect": "Allow",
       "Action": [
         "s3:GetObject*",
         "s3:PutObject",
         "s3:PutObjectAcl"
       ],
       "Resource": [
         "arn:aws:s3:::codepipeline-us-east-2-1234567890/*"
       ]
     }
   ]
}
```

13. 選擇 Review policy (檢閱政策)。

14. 在 Name (名稱) 中，輸入此政策的名稱。選擇 建立政策。
15. 在 [權限] 索引標籤上，ReadOnlyAccess在 [原則名稱] 下的政策清單中找到 AmazonS3，然後選擇原則旁邊的刪除圖示 (X)。出現提示時，選擇 Detach (分離)。

步驟 2：編輯管道

您無法使用 CodePipeline 主控台來建立或編輯使用與其他 AWS 帳戶相關之資源的管道。不過，您可以使用主控台來建立管道的一般結構，然後使用 AWS CLI 來編輯管道並新增那些資源。或者，您可以使用現有管道的架構，並手動將資源加入其中。

新增與其他 AWS 帳戶相關的資源 (AWS CLI)

1. 在終端機 (Linux、macOS 或 Unix) 或命令提示 (Windows)，然後在您要向其中新增資源的管道執行get-pipeline命令。複製命令輸出到 JSON 檔案。例如，針對名為 MyFirstPipeline 的管道，您會輸入與下列類似的程式碼：

   ```
   aws codepipeline get-pipeline --name MyFirstPipeline >pipeline.json
   ```

   輸出會傳送至*##.* json 檔案。
2. 在任何純文字編輯器中開啟 JSON 檔案。"type": "S3"在成品存放區之後，新增 KMS encryptionKey、ID 和類型資訊，其中 *codepipeline-us-east-2-1234567890* 是用於存放管道成品的 Amazon S3 儲存貯體的名稱，並且***arn:aws:kms:us-east-1:012ID_ACCOUNT_A:key/2222222-3333333-4444-556677EXAMPLE***是您剛建立的客戶管理金鑰的 ARN：

   ```
   {
     "artifactStore”: {
       "location": "codepipeline-us-east-2-1234567890",
       "type": "S3",
       "encryptionKey": {
         "id": "arn:aws:kms:us-
   east-1:012ID_ACCOUNT_A:key/2222222-3333333-4444-556677EXAMPLE",
         "type": "KMS"
       }
     },
   ```
3. 在階段中新增部署動作以使用與 *AccountB* 相關聯的CodeDeploy資源，包括您建立的跨帳戶角色的roleArn值 (*CrossAccount_* Role)。

下列範例顯示新增名為的部署動作的 JSON *ExternalDeploy*。它會使用在 *AccountB* 中建立的CodeDeploy資源，在名為「*#*備」階段。在下列範例中，*AccountB* 的 ARN 為 *012ID_ACCOUNT_B*：

```
,
            {
                "name": "Staging",
                "actions": [
                    {
                        "inputArtifacts": [
                            {
                                "name": "MyAppBuild"
                            }
                        ],
                        "name": "ExternalDeploy",
                        "actionTypeId": {
                            "category": "Deploy",
                            "owner": "AWS",
                            "version": "1",
                            "provider": "CodeDeploy"
                        },
                        "outputArtifacts": [],
                        "configuration": {
                            "ApplicationName": "AccountBApplicationName",
                            "DeploymentGroupName": "AccountBApplicationGroupName"
                        },
                        "runOrder": 1,
                        "roleArn":
 "arn:aws:iam::012ID_ACCOUNT_B:role/CrossAccount_Role"
                    }
                ]
            }
```

> **Note**
>
> 這並非適用於整個管道的 JSON，只是階段中動作的結構。

4. 您必須從檔案移除 `metadata` 行，以讓 update-pipeline 命令可以使用它。從 JSON 檔案的管道結構移除此區段 (`"metadata": { }` 行以及 `"created"`、`"pipelineARN"` 和 `"updated"` 欄位)。

例如，從結構中移除下列幾行：

```
"metadata": {
  "pipelineArn": "arn:aws:codepipeline:region:account-ID:pipeline-name",
  "created": "date",
  "updated": "date"
  }
```

儲存檔案。

5. 若要套用您的變更，請執行 update-pipeline 命令，並指定管道 JSON 檔案，與下面類似：

> **Important**
>
> 請確認在檔案名稱之前包含 `file://`。這是此命令必要項目。

```
aws codepipeline update-pipeline --cli-input-json file://pipeline.json
```

此命令會傳回所編輯管道的整個結構。

測試使用與其他 AWS 帳戶相關之資源的管道

1. 在終端機 (Linux、macOS 或 Unix) 或命令提示 (Windows) 執行start-pipeline-execution命令，然後指定管道名稱：

```
aws codepipeline start-pipeline-execution --name MyFirstPipeline
```

如需詳細資訊，請參閱手動啟動管道。

2. AWS Management Console使用 *AccountA* 登入，然後在 http://console.aws.amazon.com/codesuite/codepipeline/home 開啟CodePipeline主控台。

所有與您的 AWS 帳戶相關的管道名稱都會顯示。

3. 在 Name (名稱) 中，選擇您剛編輯的管道名稱。這會開啟管道的詳細檢視，包含管道中各階段的每項動作之狀態。

4. 從管道觀看進度。等待成功訊息，說明使用與其他 AWS 帳戶相關之資源的動作成功。

> **Note**
>
> 若您在使用 *AccountA* 登入的情況下嘗試檢視動作的詳細資訊，將收到錯誤訊息。請先登出，然後使用 *AccountB* 登入來檢視 CodeDeploy 中的部署詳細資訊。

移轉輪詢管道以使用事件型變更偵測

AWS CodePipeline支援完整、持 end-to-end 續傳遞，其中包括在程式碼變更時啟動管道。有兩種支援的方式可在程式碼變更時啟動管線：事件型變更偵測和輪詢。我們建議您對管道使用事件型變更偵測。

使用此處包含的程序將輪詢管道遷移 (更新) 到管道的事件型變更偵測方法。

管線的建議事件型變更偵測方法由管線來源決定，例如 CodeCommit。例如，在這種情況下，輪詢管道將需要使用 EventBridge.

如何移轉輪詢管道

若要移轉輪詢管道，請確定您的輪詢管道，然後決定建議的事件型變更偵測方法：

- 使用中的步驟檢視帳戶中的輪詢管道來確定您的輪詢管道。
- 在表格中，找出您的管線來源類型，然後選擇您要用來移轉輪詢管線之實作的程序。每個區段都包含多種移轉方法，例如使用 CLI 或AWS CloudFormation.

如何將管道遷移至建議的變更偵測方法		
管線來源	建議的事件型偵測方法	移轉程序
AWS CodeCommit	EventBridge （推薦）。	請參閱 使用 CodeCommit 來源移轉輪詢管道。
Amazon S3	EventBridge 和值區已啟用事件通知（建議）。	請參閱 遷移已啟用事件的 S3 來源的輪詢管道。
Amazon S3	EventBridge 和一AWS CloudTrail條小徑。	請參閱 使用 S3 來源和 CloudTrail 追蹤移轉輪詢管道。

如何將管道遷移至建議的變更偵測方法		
管線來源	建議的事件型偵測方法	移轉程序
GitHub 第一版	連線 (建議使用)	請參閱 將第 1 GitHub 版來源動作的輪詢管道移轉至連線。
GitHub 第一版	網絡掛鉤	請參閱 將第 1 GitHub 版來源動作的輪詢管道遷移至 Webhook。

Important

對於適用的管線動作組態更新 (例如具有 GitHub版本 1 動作的管道)，您必須在 Source 動作的組態中將`PollForSourceChanges`參數明確設定為 false，以停止管線進行輪詢。因此，可能會透過設定 EventBridge 規則並省略參數，錯誤地設定具有事件型變更偵測和輪詢的管道。`PollForSourceChanges`這會導致重複的管道執行項目，且管道會計入輪詢管道總數額度，而此額度的預設值遠低於以事件為基礎的管道數。如需詳細資訊，請參閱AWS CodePipeline 中的配額。

檢視帳戶中的輪詢管道

第一步，請使用下列其中一個指令碼來判斷您帳戶中的哪些管道設定要進行輪詢。這些是移轉至以事件為基礎的變更偵測的管道。

檢視帳戶中的輪詢管道 (指令碼)

請遵循下列步驟，使用指令碼來判斷帳戶中使用輪詢的管道。

1. 開啟終端機視窗，然後執行下列其中一項作業：

   - 執行下列命令以建立名為 PollingPipelinesExtractor.sh 的新指令碼。

     ```
     vi PollingPipelinesExtractor.sh
     ```

   - 要使用 Python 腳本，請運行以下命令來創建一個名為 PollingPipelinesExtractor.py 的新 Python 腳本。

```
vi PollingPipelinesExtractor.py
```

2. 將以下代碼複製並粘貼到PollingPipelinesExtractor腳本中。執行下列任意一項：

   - 將下列程式碼複製並貼到 PollingPipelinesExtractor.sh 指令碼中。

```
#!/bin/bash

set +x

POLLING_PIPELINES=()
LAST_EXECUTED_DATES=()
NEXT_TOKEN=null
HAS_NEXT_TOKEN=true
if [[ $# -eq 0 ]] ; then
    echo 'Please provide region name'
    exit 0
fi
REGION=$1


while [ "$HAS_NEXT_TOKEN" != "false" ]; do
    if [ "$NEXT_TOKEN" != "null" ];
        then
            LIST_PIPELINES_RESPONSE=$(aws codepipeline list-pipelines --region
 $REGION --next-token $NEXT_TOKEN)
        else
            LIST_PIPELINES_RESPONSE=$(aws codepipeline list-pipelines --region
 $REGION)
    fi
    LIST_PIPELINES=$(jq -r '.pipelines[].name' <<< "$LIST_PIPELINES_RESPONSE")
    NEXT_TOKEN=$(jq -r '.nextToken' <<< "$LIST_PIPELINES_RESPONSE")
    if [ "$NEXT_TOKEN" == "null" ];
        then
            HAS_NEXT_TOKEN=false
    fi

    for pipline_name in $LIST_PIPELINES
    do
        PIPELINE=$(aws codepipeline get-pipeline --name $pipline_name --region
 $REGION)
```

```
        HAS_POLLABLE_ACTIONS=$(jq '.pipeline.stages[].actions[] |
 select(.actionTypeId.category == "Source") | select(.actionTypeId.owner
 == ("ThirdParty","AWS")) | select(.actionTypeId.provider ==
 ("GitHub","S3","CodeCommit")) | select(.configuration.PollForSourceChanges ==
 ("true",null))' <<< "$PIPELINE")
        if [ ! -z "$HAS_POLLABLE_ACTIONS" ];
        then
            POLLING_PIPELINES+=("$pipline_name")
            PIPELINE_EXECUTIONS=$(aws codepipeline list-pipeline-executions --
pipeline-name $pipline_name --region $REGION)
            LAST_EXECUTION=$(jq -r '.pipelineExecutionSummaries[0]' <<<
 "$PIPELINE_EXECUTIONS")
            if [ "$LAST_EXECUTION" != "null" ];
                then
                    LAST_EXECUTED_TIMESTAMP=$(jq -r '.startTime' <<<
 "$LAST_EXECUTION")
                    LAST_EXECUTED_DATE="$(date -r ${LAST_EXECUTED_TIMESTAMP%.*})"
                else
                    LAST_EXECUTED_DATE="Not executed in last year"
            fi
            LAST_EXECUTED_DATES+=("$LAST_EXECUTED_DATE")
        fi
    done

done

fileName=$REGION-$(date +%s)
printf "| %-30s | %-30s |\n" "Polling Pipeline Name" "Last Executed Time"
printf "| %-30s | %-30s |\n" "_____________________" "___________________"
for i in "${!POLLING_PIPELINES[@]}"; do
  printf "| %-30s | %-30s |\n" "${POLLING_PIPELINES[i]}"
 "${LAST_EXECUTED_DATES[i]}"
  printf "${POLLING_PIPELINES[i]}," >> $fileName.csv
done

printf "\nSaving Polling Pipeline Names to file $fileName.csv."
```

- 將以下代碼複製並粘貼到 PollingPipelinesExtractor.py 腳本中。

```
import boto3
import sys
import time
import math
```

```
hasNextToken = True
nextToken = ""
pollablePipelines = []
lastExecutedTimes = []
if len(sys.argv) == 1:
    raise Exception("Please provide region name.")
session = boto3.Session(profile_name='default', region_name=sys.argv[1])
codepipeline = session.client('codepipeline')

def is_pollable_action(action):
    actionTypeId = action['actionTypeId']
    configuration = action['configuration']
    return actionTypeId['owner'] in {"AWS", "ThirdParty"}
 and actionTypeId['provider'] in {"GitHub", "CodeCommit",
 "S3"} and ('PollForSourceChanges' not in configuration or
 configuration['PollForSourceChanges'] == 'true')

def has_pollable_actions(pipeline):
    hasPollableAction = False
    pipelineDefinition = codepipeline.get_pipeline(name=pipeline['name'])
['pipeline']
    for action in pipelineDefinition['stages'][0]['actions']:
        hasPollableAction = is_pollable_action(action)
        if hasPollableAction:
            break
    return hasPollableAction

def get_last_executed_time(pipelineName):

 pipelineExecutions=codepipeline.list_pipeline_executions(pipelineName=pipelineName)
['pipelineExecutionSummaries']
    if pipelineExecutions:
        return pipelineExecutions[0]['startTime'].strftime("%A %m/%d/%Y, %H:%M:
%S")
    else:
        return "Not executed in last year"

while hasNextToken:
    if nextToken=="":
        list_pipelines_response = codepipeline.list_pipelines()
    else:
        list_pipelines_response =
 codepipeline.list_pipelines(nextToken=nextToken)
    if 'nextToken' in list_pipelines_response:
```

```
        nextToken = list_pipelines_response['nextToken']
    else:
        hasNextToken= False
    for pipeline in list_pipelines_response['pipelines']:
        if has_pollable_actions(pipeline):
            pollablePipelines.append(pipeline['name'])
            lastExecutedTimes.append(get_last_executed_time(pipeline['name']))

fileName="{region}-
{timeNow}.csv".format(region=sys.argv[1],timeNow=math.trunc(time.time()))
file = open(fileName, 'w')

print ("{:<30} {:<30} {:<30}".format('Polling Pipeline Name', '|','Last Executed
 Time'))
print ("{:<30} {:<30} {:<30}".format('_____________________',
 '|','___________________'))
for i in range(len(pollablePipelines)):
    print("{:<30} {:<30} {:<30}".format(pollablePipelines[i], '|',
 lastExecutedTimes[i]))
    file.write("{pipeline},".format(pipeline=pollablePipelines[i]))
file.close()
print("\nSaving Polling Pipeline Names to file
 {fileName}".format(fileName=fileName))
```

3. 對於您擁有管線的每個區域，您必須執行該區域的指令碼。若要執行指令碼，請執行下列其中一個動作：

   - 執行下列命令以執行名為 PollingPipelinesExtractor.sh 的指令碼。在此範例中，「區域」為「美國西部 -2」。

     ```
     ./PollingPipelinesExtractor.sh us-west-2
     ```

   - 對於蟒蛇腳本，運行以下命令來運行名為 PollingPipelinesExtractor.py 的 Python 腳本。在此範例中，「區域」為「美國西部 -2」。

     ```
     python3 PollingPipelinesExtractor.py us-west-2
     ```

   在指令碼的下列範例輸出中，Region us-west-2 會傳回輪詢管線清單，並顯示每個管線的上次執行時間。

```
 % ./pollingPipelineExtractor.sh us-west-2

| Polling Pipeline Name | Last Executed Time           |
| _____________________ | ____________________         |
| myCodeBuildPipeline   | Wed Mar 8 09:35:49 PST 2023  |
| myCodeCommitPipeline  | Mon Apr 24 22:32:32 PDT 2023 |
| TestPipeline          | Not executed in last year    |

Saving list of polling pipeline names to us-west-2-1682496174.csv...%
```

分析指令碼輸出，並針對清單中的每個管線，將輪詢來源更新為建議的事件型變更偵測方法。

> **Note**
>
> 您的輪詢管線是由管線對`PollForSourceChanges`參數的動作組態決定的。如果管線來源組態已省略`PollForSourceChanges`參數，則 CodePipeline預設為輪詢存放庫以進行來源變更。這種行為與包含並設置`PollForSourceChanges`為 true 相同。如需詳細資訊，請參閱管道來源動作的組態參數，例如中的 Amazon S3 來源動作組態參數Amazon S3 來源動作。

請注意，此指令碼也會產生一個 .csv 檔案，其中包含您帳戶中的輪詢管線清單，並將 .csv 檔案儲存到目前的工作資料夾中。

使用 CodeCommit 來源移轉輪詢管道

您可以遷移輪詢管道，以用 EventBridge 來偵測 CodeCommit來源儲存庫或 Amazon S3 來源儲存貯體中的變更。

CodeCommit--對於具有 CodeCommit 來源的管線，請修改管線，以便透過自動化變更偵測 EventBridge。從下列方法中選擇以實作移轉：

- 控制台：遷移輪詢管道 (CodeCommit 或 Amazon S3 來源) (主控台)
- CLI：移轉輪詢管線 (CodeCommit 來源) (CLI)
- AWS CloudFormation: 移轉輪詢管線 (CodeCommit 來源) (AWS CloudFormation範本)

遷移輪詢管道 (CodeCommit 或 Amazon S3 來源) (主控台)

您可以使用主 CodePipeline 控台更新管道，以用來偵測 EventBridge 來 CodeCommit 源儲存庫或 Amazon S3 來源儲存貯體中的變更。

> **Note**
>
> 當您使用主控台編輯具有 CodeCommit 來源儲存庫或 Amazon S3 來源儲存貯體的管道時，系統會為您建立規則和 IAM 角色。如果您使AWS CLI用編輯管道，則必須自行建立 EventBridge 規則和 IAM 角色。如需詳細資訊，請參閱 CodeCommit 來源動作和 EventBridge。

請使用這些步驟來編輯正在使用定期檢查的管道。如果您想要建立管道，請參閱在中建立管線 CodePipeline。

編輯管道來源階段

1. 請登入AWS Management Console並開啟 CodePipeline 主控台，網址為 http://console.aws.amazon.com/codesuite/codepipeline/home。

   所有與您的 AWS 帳戶相關的管道名稱都會顯示。
2. 在 Name (名稱) 中，選擇您想編輯的管道名稱。這會開啟管道的詳細檢視，包含管道中各階段的每項動作之每項狀態。
3. 在管道詳細資訊頁面上，選擇 Edit (編輯)。
4. 在 Edit (編輯) 階段，選擇來源動作上的編輯圖示。
5. 展開「變更偵測選項」，然後選擇「使用 CloudWatch 事件」以在發生變更時自動啟動我的管線 (建議選項)。

   會出現一 EventBridge 則訊息，顯示要針對此管線建立的規則。選擇 Update (更新)。

   如果您要更新具有 Amazon S3 來源的管道，您會看到下列訊息。選擇 Update (更新)。
6. 當您完成管道編輯後，請選擇 Save pipeline changes (儲存管道變更) 以返回摘要頁面。

   會出現一則訊息，顯示要為管線建立的 EventBridge 規則名稱。選擇 Save and continue (儲存並繼續)。
7. 若要測試您的動作，請使用 AWS CLI 發佈變更，將變更遞交至管道的來源階段中指定的來源。

移轉輪詢管線 (CodeCommit 來源) (CLI)

請遵循下列步驟來編輯使用輪詢 (定期檢查) 的配管，以使用 EventBridge 規則來啟動配管。如果您想要建立管道，請參閱在中建立管線 CodePipeline。

若要使用建置事件驅動的管線 CodeCommit，您可以編輯管線的PollForSourceChanges參數，然後建立下列資源：

- EventBridge 事件
- 允許此事件啟動管道的 IAM 角色

若要編輯管線的 PollForSourceChanges參數

> **Important**
>
> 當您使用這個方法建立管道時，如果沒有明確設為 false，則 PollForSourceChanges 參數會預設為 true。當新增基於事件的變更偵測時，您必須將該參數新增到輸出，並將其設為 false 以停用輪詢。否則，您的管道會針對單一來源變更啟動兩次。如需詳細資訊，請參閱 PollForSourceChanges 參數的預設設定。

1. 執行 get-pipeline 命令，將管道結構複製到 JSON 檔案。例如，針對名為 MyFirstPipeline 的管道，執行下列命令：

   ```
   aws codepipeline get-pipeline --name MyFirstPipeline >pipeline.json
   ```

   此命令不會傳回任何內容，但您建立的檔案應該會顯示在您執行命令的目錄中。

2. 在任何純文字編輯器中開啟 JSON 檔案，然後將 PollForSourceChanges 參數變更為 false，來編輯來源階段，如這個範例所示。

   為什麼我會做出此變更？ 將此參數變更為 false 會關閉定期檢查，因此您只能使用事件型變更偵測。

   ```
   "configuration": {
       "PollForSourceChanges": "false",
       "BranchName": "main",
       "RepositoryName": "MyTestRepo"
   },
   ```

3. 如果您使用的是使用 get-pipeline 命令擷取的管道結構，請從 JSON 檔案中移除 `metadata` 行。否則，update-pipeline 命令無法使用它。移除 `"metadata": { }` 行，以及 `"created"`、`"pipelineARN"` 和 `"updated"` 欄位。

   例如，從結構中移除下列幾行：

   ```
   "metadata": {
       "pipelineArn": "arn:aws:codepipeline:region:account-ID:pipeline-name",
       "created": "date",
       "updated": "date"
   },
   ```

   儲存檔案。

4. 若要套用您的變更，請執行 update-pipeline 命令、指定管道 JSON 檔案：

   > **Important**
   >
   > 請確認在檔案名稱之前包含 `file://`。這是此命令必要項目。

   ```
   aws codepipeline update-pipeline --cli-input-json file://pipeline.json
   ```

   此命令會傳回所編輯管道的整個結構。

   > **Note**
   >
   > update-pipeline 命令將終止管道。若在您執行 update-pipeline 命令時有修訂正在透過管道執行，該執行將停止。您必須手動啟動管道，以透過更新的管道執行該修訂。使用 **`start-pipeline-execution`** 命令來手動啟動您的管道。

若要以作 CodeCommit 為事件來源與 CodePipeline 目標建立 EventBridge 規則

1. 新增用 EventBridge CodePipeline 於呼叫規則的權限。如需詳細資訊，請參閱在 Amazon EventBridge 使用以資源為基礎的政策。

   a. 使用下列範例建立允許擔任服務角色 EventBridge 的信任原則。將信任政策命名為 `trustpolicyforEB.json`。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "events.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

b. 使用下列命令來建立 Role-for-MyRule 角色，並連接信任政策。

```
aws iam create-role --role-name Role-for-MyRule --assume-role-policy-document
 file://trustpolicyforEB.json
```

c. 為名為 MyFirstPipeline 的管道建立許可政策 JSON，如這個範例所示。將許可政策命名為 permissionspolicyforEB.json。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codepipeline:StartPipelineExecution"
            ],
            "Resource": [
                "arn:aws:codepipeline:us-west-2:80398EXAMPLE:MyFirstPipeline"
            ]
        }
    ]
}
```

d. 使用執行以下命令，將 CodePipeline-Permissions-Policy-for-EB 許可政策連接到 Role-for-MyRule 角色。

為什麼我會做出此變更？ 將此原則新增至角色會建立的權限 EventBridge。

```
aws iam put-role-policy --role-name Role-for-MyRule --policy-name CodePipeline-
Permissions-Policy-For-EB --policy-document file://permissionspolicyforEB.json
```

2. 呼叫 put-rule 命令，並包含 `--name`、`--event-pattern` 和 `--role-arn` 參數。

   為什麼我會做出此變更？ 此命令可讓 AWS CloudFormation 建立事件。

   以下範例命令會建立名為 `MyCodeCommitRepoRule` 的規則。

```
aws events put-rule --name "MyCodeCommitRepoRule" --event-pattern "{\"source\":
[\"aws.codecommit\"],\"detail-type\":[\"CodeCommit Repository State Change\"],
\"resources\":[\"repository-ARN\"],\"detail\":{\"referenceType\":[\"branch\"],
\"referenceName\":[\"main\"]}}" --role-arn "arn:aws:iam::ACCOUNT_ID:role/Role-for-
MyRule"
```

3. 若要新增 CodePipeline 為目標，請呼叫指put-targets令並包含下列參數：

   - `--rule` 參數與您使用 put-rule 所建立的 `rule_name` 搭配使用。
   - `--targets` 參數與目標清單中目標的清單 `Id` 和目標管道的 ARN 搭配使用。

   以下命令範例指定名為 `MyCodeCommitRepoRule` 的規則，該目標 `Id` 是由數字 1 組成，指出在規則的目標清單中，這是目標 1。此範例命令也會指定管道的範例 ARN。儲存庫中若發生變更，管道就會啟動。

```
aws events put-targets --rule MyCodeCommitRepoRule --targets
 Id=1,Arn=arn:aws:codepipeline:us-west-2:80398EXAMPLE:TestPipeline
```

移轉輪詢管線 (CodeCommit 來源) (AWS CloudFormation範本)

若要使用 AWS CodeCommit 建置事件驅動型管道，您可以編輯管道的 `PollForSourceChanges` 參數，然後將以下資源新增至您的範本：

- 一個 EventBridge 規則
- 適用於您 EventBridge 規則的 IAM 角色

如果您使用 AWS CloudFormation 來建立和管理您的管道，則您的範本會包含如下的內容。

> **Note**
>
> 來源階段 (稱為 PollForSourceChanges) 中的 Configuration 屬性。如果該屬性未包含在您的範本中，則 PollForSourceChanges 會預設為 true。

YAML

```
Resources:
  AppPipeline:
    Type: AWS::CodePipeline::Pipeline
    Properties:
      Name: codecommit-polling-pipeline
      RoleArn:
        !GetAtt CodePipelineServiceRole.Arn
      Stages:
        -
          Name: Source
          Actions:
            -
              Name: SourceAction
              ActionTypeId:
                Category: Source
                Owner: AWS
                Version: 1
                Provider: CodeCommit
              OutputArtifacts:
                - Name: SourceOutput
              Configuration:
                BranchName: !Ref BranchName
                RepositoryName: !Ref RepositoryName
                PollForSourceChanges: true
              RunOrder: 1
```

JSON

```
"Stages": [
    {
        "Name": "Source",
  "Actions": [{
      "Name": "SourceAction",
      "ActionTypeId": {
```

```
    "Category": "Source",
    "Owner": "AWS",
    "Version": 1,
    "Provider": "CodeCommit"
      },
      "OutputArtifacts": [{
          "Name": "SourceOutput"
      }],
      "Configuration": {
          "BranchName": {
        "Ref": "BranchName"
    },
    "RepositoryName": {
        "Ref": "RepositoryName"
    },
    "PollForSourceChanges": true
            },
            "RunOrder": 1
        }]
    },
```

更新您的管道 AWS CloudFormation 範本並建立 EventBridge 規則

1. 在範本的 `Resources` 下，使用 `AWS::IAM::Role` AWS CloudFormation 資源，來設定允許事件啟動管道的 IAM 角色。此項目會建立一個使用兩個政策的角色：

   - 第一個政策允許要承擔的角色。
   - 第二個政策提供啟動管道的許可。

   為什麼我會做出此變更？ 新增資`AWS::IAM::Role`源可AWS CloudFormation建立的權限 EventBridge。此資源會新增至您的 AWS CloudFormation 堆疊。

   YAML

   ```
     EventRole:
       Type: AWS::IAM::Role
       Properties:
         AssumeRolePolicyDocument:
           Version: 2012-10-17
           Statement:
             -
   ```

```
            Effect: Allow
            Principal:
              Service:
                - events.amazonaws.com
            Action: sts:AssumeRole
      Path: /
      Policies:
        -
          PolicyName: eb-pipeline-execution
          PolicyDocument:
            Version: 2012-10-17
            Statement:
              -
                Effect: Allow
                Action: codepipeline:StartPipelineExecution
                Resource: !Join [ '', [ 'arn:aws:codepipeline:', !Ref
 'AWS::Region', ':', !Ref 'AWS::AccountId', ':', !Ref AppPipeline ] ]
```

JSON

```
"EventRole": {
  "Type": "AWS::IAM::Role",
  "Properties": {
    "AssumeRolePolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "Service": [
              "events.amazonaws.com"
            ]
          },
          "Action": "sts:AssumeRole"
        }
      ]
    },
    "Path": "/",
    "Policies": [
      {
        "PolicyName": "eb-pipeline-execution",
        "PolicyDocument": {
          "Version": "2012-10-17",
```

```
          "Statement": [
            {
              "Effect": "Allow",
              "Action": "codepipeline:StartPipelineExecution",
              "Resource": {
                "Fn::Join": [
                  "",
                  [
                    "arn:aws:codepipeline:",
                    {
                      "Ref": "AWS::Region"
                    },
                    ":",
                    {
                      "Ref": "AWS::AccountId"
                    },
                    ":",
                    {
                      "Ref": "AppPipeline"
                    }
                  ]

...
```

2. 在範本的下Resources，使用資AWS::Events::RuleAWS CloudFormation源新增 EventBridge 規則。此事件模式會建立事件，以監視存放庫的推送變更。 EventBridge 偵測到存放庫狀態變更時，規則會StartPipelineExecution在您的目標管線上叫用。

   我為什麼要進行這項變更？ 新增 AWS::Events::Rule 資源可讓 AWS CloudFormation 建立事件。此資源會新增至您的 AWS CloudFormation 堆疊。

   YAML

```
EventRule:
  Type: AWS::Events::Rule
  Properties:
    EventPattern:
      source:
        - aws.codecommit
      detail-type:
        - 'CodeCommit Repository State Change'
      resources:
```

```
          - !Join [ '', [ 'arn:aws:codecommit:', !Ref 'AWS::Region', ':', !Ref
 'AWS::AccountId', ':', !Ref RepositoryName ] ]
        detail:
          event:
            - referenceCreated
            - referenceUpdated
          referenceType:
            - branch
          referenceName:
            - main
      Targets:
        -
          Arn:
            !Join [ '', [ 'arn:aws:codepipeline:', !Ref 'AWS::Region', ':', !Ref
 'AWS::AccountId', ':', !Ref AppPipeline ] ]
          RoleArn: !GetAtt EventRole.Arn
          Id: codepipeline-AppPipeline
```

JSON

```
"EventRule": {
  "Type": "AWS::Events::Rule",
  "Properties": {
    "EventPattern": {
      "source": [
        "aws.codecommit"
      ],
      "detail-type": [
        "CodeCommit Repository State Change"
      ],
      "resources": [
        {
          "Fn::Join": [
            "",
            [
              "arn:aws:codecommit:",
              {
                "Ref": "AWS::Region"
              },
              ":",
              {
                "Ref": "AWS::AccountId"
```

```
              },
              ":",
              {
                "Ref": "RepositoryName"
              }
            ]
          ]
        }
      ],
      "detail": {
        "event": [
          "referenceCreated",
          "referenceUpdated"
        ],
        "referenceType": [
          "branch"
        ],
        "referenceName": [
          "main"
        ]
      }
    },
    "Targets": [
      {
        "Arn": {
          "Fn::Join": [
            "",
            [
              "arn:aws:codepipeline:",
              {
                "Ref": "AWS::Region"
              },
              ":",
              {
                "Ref": "AWS::AccountId"
              },
              ":",
              {
                "Ref": "AppPipeline"
              }
            ]
          ]
        },
        "RoleArn": {
```

```
          "Fn::GetAtt": [
            "EventRole",
            "Arn"
          ]
        },
        "Id": "codepipeline-AppPipeline"
      }
    ]
  }
},
```

3. 將更新的範本儲存到本機電腦，然後開啟 AWS CloudFormation 主控台。
4. 選擇您的堆疊，然後選擇 Create Change Set for Current Stack (建立目前堆疊的變更集)。
5. 上傳範本，然後檢視 AWS CloudFormation 中所列的變更。這些是會針對堆疊進行的變更。您應該會在清單中看到新資源。
6. 選擇 Execute (執行)。

若要編輯管線的 PollForSourceChanges參數

> **Important**
>
> 在許多情況下，當您建立管道時，PollForSourceChanges 參數會預設為 true。當新增基於事件的變更偵測時，您必須將該參數新增到輸出，並將其設為 false 以停用輪詢。否則，您的管道會針對單一來源變更啟動兩次。如需詳細資訊，請參閱 PollForSourceChanges 參數的預設設定。

- 在範本中，將 PollForSourceChanges 變更為 false。如果您並未在管道定義中包含 PollForSourceChanges，請新增它，並將其設為 false。

  為什麼我會做出此變更？ 將此參數變更為 false 會關閉定期檢查，因此您只能使用事件型變更偵測。

  YAML

```
        Name: Source
        Actions:
          -
            Name: SourceAction
            ActionTypeId:
```

```
                Category: Source
                Owner: AWS
                Version: 1
                Provider: CodeCommit
              OutputArtifacts:
                - Name: SourceOutput
              Configuration:
                BranchName: !Ref BranchName
                RepositoryName: !Ref RepositoryName
                PollForSourceChanges: false
              RunOrder: 1
```

JSON

```
{
  "Name": "Source",
  "Actions": [
    {
      "Name": "SourceAction",
      "ActionTypeId": {
        "Category": "Source",
        "Owner": "AWS",
        "Version": 1,
        "Provider": "CodeCommit"
      },
      "OutputArtifacts": [
        {
          "Name": "SourceOutput"
        }
      ],
      "Configuration": {
        "BranchName": {
          "Ref": "BranchName"
        },
        "RepositoryName": {
          "Ref": "RepositoryName"
        },
        "PollForSourceChanges": false
      },
      "RunOrder": 1
    }
  ]
},
```

Example

如果您使用 AWS CloudFormation 建立這些資源，則在建立或更新儲存庫中的檔案時，會觸發您的管道。以下是最終範本片段：

YAML

```
Resources:
  EventRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: 2012-10-17
        Statement:
          -
            Effect: Allow
            Principal:
              Service:
                - events.amazonaws.com
            Action: sts:AssumeRole
      Path: /
      Policies:
        -
          PolicyName: eb-pipeline-execution
          PolicyDocument:
            Version: 2012-10-17
            Statement:
              -
                Effect: Allow
                Action: codepipeline:StartPipelineExecution
                Resource: !Join [ '', [ 'arn:aws:codepipeline:', !Ref 'AWS::Region',
 ':', !Ref 'AWS::AccountId', ':', !Ref AppPipeline ] ]
  EventRule:
    Type: AWS::Events::Rule
    Properties:
      EventPattern:
        source:
          - aws.codecommit
        detail-type:
          - 'CodeCommit Repository State Change'
```

```
       resources:
         - !Join [ '', [ 'arn:aws:codecommit:', !Ref 'AWS::Region', ':', !Ref
'AWS::AccountId', ':', !Ref RepositoryName ] ]
       detail:
         event:
           - referenceCreated
           - referenceUpdated
         referenceType:
           - branch
         referenceName:
           - main
     Targets:
       -
         Arn:
           !Join [ '', [ 'arn:aws:codepipeline:', !Ref 'AWS::Region', ':', !Ref
'AWS::AccountId', ':', !Ref AppPipeline ] ]
         RoleArn: !GetAtt EventRole.Arn
         Id: codepipeline-AppPipeline
 AppPipeline:
   Type: AWS::CodePipeline::Pipeline
   Properties:
     Name: codecommit-events-pipeline
     RoleArn:
       !GetAtt CodePipelineServiceRole.Arn
     Stages:
       -
         Name: Source
         Actions:
           -
             Name: SourceAction
             ActionTypeId:
               Category: Source
               Owner: AWS
               Version: 1
               Provider: CodeCommit
             OutputArtifacts:
               - Name: SourceOutput
             Configuration:
               BranchName: !Ref BranchName
               RepositoryName: !Ref RepositoryName
               PollForSourceChanges: false
             RunOrder: 1
```

```
...
```

JSON

```
    "Resources": {

...

        "EventRole": {
            "Type": "AWS::IAM::Role",
            "Properties": {
                "AssumeRolePolicyDocument": {
                    "Version": "2012-10-17",
                    "Statement": [
                        {
                            "Effect": "Allow",
                            "Principal": {
                                "Service": [
                                    "events.amazonaws.com"
                                ]
                            },
                            "Action": "sts:AssumeRole"
                        }
                    ]
                },
                "Path": "/",
                "Policies": [
                    {
                        "PolicyName": "eb-pipeline-execution",
                        "PolicyDocument": {
                            "Version": "2012-10-17",
                            "Statement": [
                                {
                                    "Effect": "Allow",
                                    "Action": "codepipeline:StartPipelineExecution",
                                    "Resource": {
                                        "Fn::Join": [
                                            "",
                                            [
                                                "arn:aws:codepipeline:",
                                                {
                                                    "Ref": "AWS::Region"
                                                },
```

```
                                        ":",
                                        {
                                            "Ref": "AWS::AccountId"
                                        },
                                        ":",
                                        {
                                            "Ref": "AppPipeline"
                                        }
                                    ]
                                ]
                            }
                        }
                    ]
                }
            }
        ]
    }
},
"EventRule": {
    "Type": "AWS::Events::Rule",
    "Properties": {
        "EventPattern": {
            "source": [
                "aws.codecommit"
            ],
            "detail-type": [
                "CodeCommit Repository State Change"
            ],
            "resources": [
                {
                    "Fn::Join": [
                        "",
                        [
                            "arn:aws:codecommit:",
                            {
                                "Ref": "AWS::Region"
                            },
                            ":",
                            {
                                "Ref": "AWS::AccountId"
                            },
                            ":",
                            {
                                "Ref": "RepositoryName"
```

```
                                }
                            ]
                        ]
                    }
                ],
                "detail": {
                    "event": [
                        "referenceCreated",
                        "referenceUpdated"
                    ],
                    "referenceType": [
                        "branch"
                    ],
                    "referenceName": [
                        "main"
                    ]
                }
            },
            "Targets": [
                {
                    "Arn": {
                        "Fn::Join": [
                            "",
                            [
                                "arn:aws:codepipeline:",
                                {
                                    "Ref": "AWS::Region"
                                },
                                ":",
                                {
                                    "Ref": "AWS::AccountId"
                                },
                                ":",
                                {
                                    "Ref": "AppPipeline"
                                }
                            ]
                        ]
                    },
                    "RoleArn": {
                        "Fn::GetAtt": [
                            "EventRole",
                            "Arn"
                        ]
```

```
                    },
                    "Id": "codepipeline-AppPipeline"
                }
            ]
        }
    },
    "AppPipeline": {
        "Type": "AWS::CodePipeline::Pipeline",
        "Properties": {
            "Name": "codecommit-events-pipeline",
            "RoleArn": {
                "Fn::GetAtt": [
                    "CodePipelineServiceRole",
                    "Arn"
                ]
            },
            "Stages": [
                {
                    "Name": "Source",
                    "Actions": [
                        {
                            "Name": "SourceAction",
                            "ActionTypeId": {
                                "Category": "Source",
                                "Owner": "AWS",
                                "Version": 1,
                                "Provider": "CodeCommit"
                            },
                            "OutputArtifacts": [
                                {
                                    "Name": "SourceOutput"
                                }
                            ],
                            "Configuration": {
                                "BranchName": {
                                    "Ref": "BranchName"
                                },
                                "RepositoryName": {
                                    "Ref": "RepositoryName"
                                },
                                "PollForSourceChanges": false
                            },
                            "RunOrder": 1
                        }
```

```
                    ]
                },

...
```

遷移已啟用事件的 S3 來源的輪詢管道

對於具有 Amazon S3 來源的管道，請修改管道，以便透 EventBridge 過啟用事件通知的來源儲存貯體自動執行變更偵測。如果您使用 CLI 或AWS CloudFormation遷移管道，建議使用此方法。

> **Note**
>
> 這包括使用已啟用事件通知的值區，您不需要在其中建立個別 CloudTrail 追蹤。如果您使用主控台，則會為您設定事件規則和 CloudTrail 追蹤。如需這些步驟，請參閱使用 S3 來源和 CloudTrail 追蹤移轉輪詢管道。

- CLI：使用 S3 來源和 CloudTrail 追蹤 (CLI) 移轉輪詢管道
- AWS CloudFormation: 使用 S3 來源和 CloudTrail 追蹤 (AWS CloudFormation範本) 移轉輪詢管道

使用已啟用事件 (CLI) 的 S3 來源移轉輪詢管道

請遵循下列步驟來編輯使用輪詢 (定期檢查) EventBridge 改為使用事件的管線。如果您想要建立管道，請參閱在中建立管線 CodePipeline。

若要使用 Amazon S3 建立事件驅動管道，您可以編輯管道的`PollForSourceChanges`參數，然後建立下列資源：

- EventBridge 事件規則
- IAM 角色，可讓 EventBridge 事件啟動您的管道

使用 Amazon S3 做為事件來源和目標建立 EventBridge 規則 CodePipeline ，並套用許可政策

1. 授與用 EventBridge CodePipeline 於呼叫規則的權限。如需詳細資訊，請參閱在 Amazon EventBridge 使用以資源為基礎的政策。

a. 使用以下範本來建立信任政策，以允許 EventBridge 擔任服務角色。將其命名為 `trustpolicyforEB.json`。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "events.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

b. 使用下列命令來建立 `Role-for-MyRule` 角色，並連接信任政策。

為什麼我會做出此變更？ 將此信任政策新增至角色即會建立 EventBridge 的許可。

```
aws iam create-role --role-name Role-for-MyRule --assume-role-policy-document
 file://trustpolicyforEB.json
```

c. 針對名為 `MyFirstPipeline` 的管道建立許可政策 JSON，如下所示。將許可政策命名為 `permissionspolicyforEB.json`。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codepipeline:StartPipelineExecution"
            ],
            "Resource": [
                "arn:aws:codepipeline:us-west-2:80398EXAMPLE:MyFirstPipeline"
            ]
        }
    ]
}
```

d. 使用執行以下命令，將新的 CodePipeline-Permissions-Policy-for-EB 許可政策連接到您所建立的 Role-for-MyRule 角色。

```
aws iam put-role-policy --role-name Role-for-MyRule --policy-name CodePipeline-
Permissions-Policy-For-EB --policy-document file://permissionspolicyforEB.json
```

2. 呼叫 put-rule 命令，並包含 --name、--event-pattern 和 --role-arn 參數。

以下範例命令會建立名為 EnabledS3SourceRule 的規則。

```
aws events put-rule --name "EnabledS3SourceRule" --event-pattern "{\"source\":
[\"aws.s3\"],\"detail-type\":[\"Object Created\"],\"detail\":{\"bucket\":{\"name\":
[\"my-bucket\"]}}}" --role-arn "arn:aws:iam::ACCOUNT_ID:role/Role-for-MyRule"
```

3. 若要新增 CodePipeline 為目標，請呼叫put-targets指令並包含--rule和--targets參數。

以下命令指定名為 EnabledS3SourceRule 的規則，該目標 Id 是由數字 1 組成，指出在規則的目標清單中，這是目標 1。此命令也會指定管道的範例 ARN。儲存庫中若發生變更，管道就會啟動。

```
aws events put-targets --rule EnabledS3SourceRule --targets Id=codepipeline-
AppPipeline,Arn=arn:aws:codepipeline:us-west-2:80398EXAMPLE:TestPipeline
```

若要編輯管線的 PollForSourceChanges參數

> **Important**
>
> 當您使用這個方法建立管道時，如果沒有明確設為 false，則 PollForSourceChanges 參數會預設為 true。當新增基於事件的變更偵測時，您必須將該參數新增到輸出，並將其設為 false 以停用輪詢。否則，您的管道會針對單一來源變更啟動兩次。如需詳細資訊，請參閱 PollForSourceChanges 參數的預設設定。

1. 執行 get-pipeline 命令，將管道結構複製到 JSON 檔案。例如，針對名為 MyFirstPipeline 的管道，執行下列命令：

```
aws codepipeline get-pipeline --name MyFirstPipeline >pipeline.json
```

此命令不會傳回任何內容，但您建立的檔案應該會顯示在您執行命令的目錄中。

2. 在任何純文字編輯器中開啟 JSON 檔案，然後將名為 `storage-bucket` 之儲存貯體的 `PollForSourceChanges` 參數變更為 `false`，來編輯來源階段，如這個範例所示。

   為什麼我會做出此變更？ 將此參數設為 `false` 會關閉定期檢查，因此您只能使用事件型變更偵測。

   ```
   "configuration": {
       "S3Bucket": "storage-bucket",
       "PollForSourceChanges": "false",
       "S3ObjectKey": "index.zip"
   },
   ```

3. 如果您使用的是使用 get-pipeline 命令擷取的管道結構，則必須從 JSON 檔案中移除 `metadata` 行。否則，update-pipeline 命令無法使用它。移除 `"metadata": { }` 行，以及 `"created"`、`"pipelineARN"` 和 `"updated"` 欄位。

   例如，從結構中移除下列幾行：

   ```
   "metadata": {
       "pipelineArn": "arn:aws:codepipeline:region:account-ID:pipeline-name",
       "created": "date",
       "updated": "date"
   },
   ```

   儲存檔案。

4. 若要套用您的變更，請執行 update-pipeline 命令、指定管道 JSON 檔案：

   > **Important**
   >
   > 請確認在檔案名稱之前包含 `file://`。這是此命令必要項目。

   ```
   aws codepipeline update-pipeline --cli-input-json file://pipeline.json
   ```

   此命令會傳回所編輯管道的整個結構。

> **Note**
>
> update-pipeline 命令將終止管道。若在您執行 update-pipeline 命令時有修訂正在透過管道執行，該執行將停止。您必須手動啟動管道，以透過更新的管道執行該修訂。使用 start-pipeline-execution 命令來手動啟動您的管道。

遷移已啟用事件 (AWS CloudFormation範本) 的 S3 來源的輪詢管道

此程序適用於來源儲存貯體已啟用事件的管道。

使用以下步驟透過 Amazon S3 來源編輯管道，從輪詢到事件型變更偵測。

若要使用 Amazon S3 建立事件驅動管道，您可以編輯管道的`PollForSourceChanges`參數，然後將下列資源新增至範本：

- EventBridge 允許此事件啟動管道的規則和 IAM 角色。

如果您使用 AWS CloudFormation 來建立和管理您的管道，則您的範本會包含如下的內容。

> **Note**
>
> 來源階段 (稱為 `PollForSourceChanges`) 中的 `Configuration` 屬性。如果您的範本未包含該屬性，則 `PollForSourceChanges` 會預設為 `true`。

YAML

```
AppPipeline:
  Type: AWS::CodePipeline::Pipeline
  Properties:
    RoleArn: !GetAtt CodePipelineServiceRole.Arn
    Stages:
      -
        Name: Source
        Actions:
          -
            Name: SourceAction
            ActionTypeId:
```

```
                Category: Source
                Owner: AWS
                Version: 1
                Provider: S3
              OutputArtifacts:
                -
                  Name: SourceOutput
              Configuration:
                S3Bucket: !Ref SourceBucket
                S3ObjectKey: !Ref S3SourceObjectKey
                PollForSourceChanges: true
              RunOrder: 1


...
```

JSON

```
        "AppPipeline": {
            "Type": "AWS::CodePipeline::Pipeline",
            "Properties": {
                "RoleArn": {
                    "Fn::GetAtt": ["CodePipelineServiceRole", "Arn"]
                },
                "Stages": [
                    {
                        "Name": "Source",
                        "Actions": [
                            {
                                "Name": "SourceAction",
                                "ActionTypeId": {
                                    "Category": "Source",
                                    "Owner": "AWS",
                                    "Version": 1,
                                    "Provider": "S3"
                                },
                                "OutputArtifacts": [
                                    {
                                        "Name": "SourceOutput"
                                    }
                                ],
                                "Configuration": {
                                    "S3Bucket": {
```

```
                                    "Ref": "SourceBucket"
                                },
                                "S3ObjectKey": {
                                    "Ref": "SourceObjectKey"
                                },
                                "PollForSourceChanges": true
                            },
                            "RunOrder": 1
                        }
                    ]
                },


...
```

使用 Amazon S3 做為事件來源和目標建立 EventBridge 規則 CodePipeline ，並套用許可政策

1. 在範本的 Resources 下，使用 AWS::IAM::Role AWS CloudFormation 資源，來設定允許事件啟動管道的 IAM 角色。此項目會建立一個使用兩個政策的角色：

   - 第一個政策允許要承擔的角色。
   - 第二個政策提供啟動管道的許可。

   為什麼我會做出此變更？ 新增AWS::IAM::Role資源可AWS CloudFormation建立的權限 EventBridge。此資源會新增至您的 AWS CloudFormation 堆疊。

   YAML

   ```
   EventRole:
     Type: AWS::IAM::Role
     Properties:
       AssumeRolePolicyDocument:
         Version: 2012-10-17
         Statement:
           -
             Effect: Allow
             Principal:
               Service:
                 - events.amazonaws.com
             Action: sts:AssumeRole
       Path: /
   ```

```
      Policies:
        -
          PolicyName: eb-pipeline-execution
          PolicyDocument:
            Version: 2012-10-17
            Statement:
              -
                Effect: Allow
                Action: codepipeline:StartPipelineExecution
                Resource: !Join [ '', [ 'arn:aws:codepipeline:', !Ref
 'AWS::Region', ':', !Ref 'AWS::AccountId', ':', !Ref AppPipeline ] ]


...
```

JSON

```
  "EventRole": {
    "Type": "AWS::IAM::Role",
    "Properties": {
      "AssumeRolePolicyDocument": {
        "Version": "2012-10-17",
        "Statement": [
          {
            "Effect": "Allow",
            "Principal": {
              "Service": [
                "events.amazonaws.com"
              ]
            },
            "Action": "sts:AssumeRole"
          }
        ]
      },
      "Path": "/",
      "Policies": [
        {
          "PolicyName": "eb-pipeline-execution",
          "PolicyDocument": {
            "Version": "2012-10-17",
            "Statement": [
              {
                "Effect": "Allow",
```

```
                "Action": "codepipeline:StartPipelineExecution",
                "Resource": {
                  "Fn::Join": [
                    "",
                    [
                      "arn:aws:codepipeline:",
                      {
                        "Ref": "AWS::Region"
                      },
                      ":",
                      {
                        "Ref": "AWS::AccountId"
                      },
                      ":",
                      {
                        "Ref": "AppPipeline"
                      }
                    ]
                  ]

...
```

2. 使用AWS::Events::RuleAWS CloudFormation資源新增 EventBridge 規則。此事件模式會建立一個事件，以監控 Amazon S3 來源儲存貯體中物件的建立或刪除。此外，會包含您管道的目標。建立物件後，此規則會StartPipelineExecution在您的目標管線上叫用。

   為什麼我會做出此變更？ 新增 AWS::Events::Rule 資源可讓 AWS CloudFormation 建立事件。此資源會新增至您的 AWS CloudFormation 堆疊。

   YAML

```
  EventRule:
    Type: AWS::Events::Rule
    Properties:
      EventBusName: default
      EventPattern:
        source:
          - aws.s3
        detail-type:
          - Object Created
        detail:
          bucket:
            name:
```

```
              - !Ref SourceBucket
      Name: EnabledS3SourceRule
      State: ENABLED
      Targets:
        -
          Arn:
            !Join [ '', [ 'arn:aws:codepipeline:', !Ref 'AWS::Region', ':', !Ref
 'AWS::AccountId', ':', !Ref AppPipeline ] ]
          RoleArn: !GetAtt EventRole.Arn
          Id: codepipeline-AppPipeline


...
```

JSON

```
 "EventRule": {
   "Type": "AWS::Events::Rule",
   "Properties": {
 "EventBusName": "default",
 "EventPattern": {
     "source": [
  "aws.s3"
     ],
     "detail-type": [
   "Object Created"
     ],
     "detail": {
   "bucket": {
       "name": [
     "s3-pipeline-source-fra-bucket"
       ]
       }
           }
 },
 "Name": "EnabledS3SourceRule",
       "State": "ENABLED",
       "Targets": [
       {
         "Arn": {
           "Fn::Join": [
             "",
```

```
              [
                "arn:aws:codepipeline:",
                {
                  "Ref": "AWS::Region"
                },
                ":",
                {
                  "Ref": "AWS::AccountId"
                },
                ":",
                {
                  "Ref": "AppPipeline"
                }
              ]
            ]
          },
          "RoleArn": {
            "Fn::GetAtt": [
              "EventRole",
              "Arn"
            ]
          },
          "Id": "codepipeline-AppPipeline"
        }
      ]
    }
  }
},

...
```

3. 儲存您的更新範本到本機電腦，並開啟 AWS CloudFormation 主控台。
4. 選擇您的堆疊，然後選擇 Create Change Set for Current Stack (建立目前堆疊的變更集)。
5. 上傳您的更新範本，然後檢視中 AWS CloudFormation 所列的變更。這些是會針對堆疊進行的變更。您應該會在清單中看到新資源。
6. 選擇 Execute (執行)。

若要編輯管線的 PollForSourceChanges參數

> **Important**
>
> 當您使用這個方法建立管道時，如果沒有明確設為 false，則 PollForSourceChanges 參數會預設為 true。當新增基於事件的變更偵測時，您必須將該參數新增到輸出，並將其設為 false 以停用輪詢。否則，您的管道會針對單一來源變更啟動兩次。如需詳細資訊，請參閱 PollForSourceChanges 參數的預設設定。

- 在範本中，將 PollForSourceChanges 變更為 false。如果您並未在管道定義中包含 PollForSourceChanges，請新增它，並將其設為 false。

  為什麼我會做出此變更？ 將 PollForSourceChanges變更為 false 會關閉定期檢查，因此您只能使用事件型變更偵測。

  YAML

```
        Name: Source
        Actions:
          -
            Name: SourceAction
            ActionTypeId:
              Category: Source
              Owner: AWS
              Version: 1
              Provider: S3
            OutputArtifacts:
              - Name: SourceOutput
            Configuration:
              S3Bucket: !Ref SourceBucket
              S3ObjectKey: !Ref SourceObjectKey
              PollForSourceChanges: false
            RunOrder: 1
```

  JSON

```
{
   "Name": "SourceAction",
   "ActionTypeId": {
     "Category": "Source",
```

```
      "Owner": "AWS",
      "Version": 1,
      "Provider": "S3"
    },
    "OutputArtifacts": [
      {
        "Name": "SourceOutput"
      }
    ],
    "Configuration": {
      "S3Bucket": {
        "Ref": "SourceBucket"
      },
      "S3ObjectKey": {
        "Ref": "SourceObjectKey"
      },
      "PollForSourceChanges": false
    },
    "RunOrder": 1
  }
```

Example

如果您使用 AWS CloudFormation 建立這些資源，則在建立或更新儲存庫中的檔案時，會觸發您的管道。

> **Note**
>
> 不要在此處停止。雖然管道已建立，但您必須為 Amazon S3 管道建立第二個AWS CloudFormation範本。如果您未建立第二個範本，您的管道不會有任何變更偵測功能。

YAML

```
Parameters:
  SourceObjectKey:
    Description: 'S3 source artifact'
    Type: String
    Default: SampleApp_Linux.zip
  ApplicationName:
    Description: 'CodeDeploy application name'
```

```
    Type: String
    Default: DemoApplication
  BetaFleet:
    Description: 'Fleet configured in CodeDeploy'
    Type: String
    Default: DemoFleet

Resources:
  SourceBucket:
    Type: AWS::S3::Bucket
    Properties:
      NotificationConfiguration:
        EventBridgeConfiguration:
          EventBridgeEnabled: true
      VersioningConfiguration:
        Status: Enabled
  CodePipelineArtifactStoreBucket:
    Type: AWS::S3::Bucket
  CodePipelineArtifactStoreBucketPolicy:
    Type: AWS::S3::BucketPolicy
    Properties:
      Bucket: !Ref CodePipelineArtifactStoreBucket
      PolicyDocument:
        Version: 2012-10-17
        Statement:
          -
            Sid: DenyUnEncryptedObjectUploads
            Effect: Deny
            Principal: '*'
            Action: s3:PutObject
            Resource: !Join [ '', [ !GetAtt CodePipelineArtifactStoreBucket.Arn, '/
*' ] ]
            Condition:
              StringNotEquals:
                s3:x-amz-server-side-encryption: aws:kms
          -
            Sid: DenyInsecureConnections
            Effect: Deny
            Principal: '*'
            Action: s3:*
            Resource: !Sub ${CodePipelineArtifactStoreBucket.Arn}/*
            Condition:
              Bool:
                aws:SecureTransport: false
```

```
  CodePipelineServiceRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: 2012-10-17
        Statement:
          -
            Effect: Allow
            Principal:
              Service:
                - codepipeline.amazonaws.com
            Action: sts:AssumeRole
      Path: /
      Policies:
        -
          PolicyName: AWS-CodePipeline-Service-3
          PolicyDocument:
            Version: 2012-10-17
            Statement:
              -
                Effect: Allow
                Action:
                  - codecommit:CancelUploadArchive
                  - codecommit:GetBranch
                  - codecommit:GetCommit
                  - codecommit:GetUploadArchiveStatus
                  - codecommit:UploadArchive
                Resource: 'resource_ARN'
              -
                Effect: Allow
                Action:
                  - codedeploy:CreateDeployment
                  - codedeploy:GetApplicationRevision
                  - codedeploy:GetDeployment
                  - codedeploy:GetDeploymentConfig
                  - codedeploy:RegisterApplicationRevision
                Resource: 'resource_ARN'
              -
                Effect: Allow
                Action:
                  - codebuild:BatchGetBuilds
                  - codebuild:StartBuild
                Resource: 'resource_ARN'
              -
```

```
                Effect: Allow
                Action:
                  - devicefarm:ListProjects
                  - devicefarm:ListDevicePools
                  - devicefarm:GetRun
                  - devicefarm:GetUpload
                  - devicefarm:CreateUpload
                  - devicefarm:ScheduleRun
                Resource: 'resource_ARN'
              -
                Effect: Allow
                Action:
                  - lambda:InvokeFunction
                  - lambda:ListFunctions
                Resource: 'resource_ARN'
              -
                Effect: Allow
                Action:
                  - iam:PassRole
                Resource: 'resource_ARN'
              -
                Effect: Allow
                Action:
                  - elasticbeanstalk:*
                  - ec2:*
                  - elasticloadbalancing:*
                  - autoscaling:*
                  - cloudwatch:*
                  - s3:*
                  - sns:*
                  - cloudformation:*
                  - rds:*
                  - sqs:*
                  - ecs:*
                Resource: 'resource_ARN'
  AppPipeline:
    Type: AWS::CodePipeline::Pipeline
    Properties:
      Name: s3-events-pipeline
      RoleArn:
        !GetAtt CodePipelineServiceRole.Arn
      Stages:
        -
          Name: Source
```

```
          Actions:
            -
              Name: SourceAction
              ActionTypeId:
                Category: Source
                Owner: AWS
                Version: 1
                Provider: S3
              OutputArtifacts:
                - Name: SourceOutput
              Configuration:
                S3Bucket: !Ref SourceBucket
                S3ObjectKey: !Ref SourceObjectKey
                PollForSourceChanges: false
              RunOrder: 1
        -
          Name: Beta
          Actions:
            -
              Name: BetaAction
              InputArtifacts:
                - Name: SourceOutput
              ActionTypeId:
                Category: Deploy
                Owner: AWS
                Version: 1
                Provider: CodeDeploy
              Configuration:
                ApplicationName: !Ref ApplicationName
                DeploymentGroupName: !Ref BetaFleet
              RunOrder: 1
      ArtifactStore:
        Type: S3
        Location: !Ref CodePipelineArtifactStoreBucket
  EventRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: 2012-10-17
        Statement:
          -
            Effect: Allow
            Principal:
              Service:
```

```
                - events.amazonaws.com
            Action: sts:AssumeRole
      Path: /
      Policies:
        -
          PolicyName: eb-pipeline-execution
          PolicyDocument:
            Version: 2012-10-17
            Statement:
              -
                Effect: Allow
                Action: codepipeline:StartPipelineExecution
                Resource: !Join [ '', [ 'arn:aws:codepipeline:', !Ref 'AWS::Region',
':', !Ref 'AWS::AccountId', ':', !Ref AppPipeline ] ]
 EventRule:
   Type: AWS::Events::Rule
   Properties:
     EventBusName: default
     EventPattern:
       source:
         - aws.s3
       detail-type:
         - Object Created
       detail:
         bucket:
           name:
             - !Ref SourceBucket
     Name: EnabledS3SourceRule
     State: ENABLED
     Targets:
       -
         Arn:
           !Join [ '', [ 'arn:aws:codepipeline:', !Ref 'AWS::Region', ':', !Ref
'AWS::AccountId', ':', !Ref AppPipeline ] ]
         RoleArn: !GetAtt EventRole.Arn
         Id: codepipeline-AppPipeline
```

JSON

```
{
    "Parameters": {
        "SourceObjectKey": {
```

```
            "Description": "S3 source artifact",
            "Type": "String",
            "Default": "SampleApp_Linux.zip"
        },
        "ApplicationName": {
            "Description": "CodeDeploy application name",
            "Type": "String",
            "Default": "DemoApplication"
        },
        "BetaFleet": {
            "Description": "Fleet configured in CodeDeploy",
            "Type": "String",
            "Default": "DemoFleet"
        }
    },
    "Resources": {
        "SourceBucket": {
            "Type": "AWS::S3::Bucket",
              "Properties": {
                "NotificationConfiguration": {
                    "EventBridgeConfiguration": {
                        "EventBridgeEnabled": true
                    }
                },
                "VersioningConfiguration": {
                    "Status": "Enabled"
                }
            }
        },
        "CodePipelineArtifactStoreBucket": {
            "Type": "AWS::S3::Bucket"
        },
        "CodePipelineArtifactStoreBucketPolicy": {
            "Type": "AWS::S3::BucketPolicy",
            "Properties": {
                "Bucket": {
                    "Ref": "CodePipelineArtifactStoreBucket"
                },
                "PolicyDocument": {
                    "Version": "2012-10-17",
                    "Statement": [
                        {
                            "Sid": "DenyUnEncryptedObjectUploads",
                            "Effect": "Deny",
```

```
                "Principal": "*",
                "Action": "s3:PutObject",
                "Resource": {
                    "Fn::Join": [
                        "",
                        [
                            {
                                "Fn::GetAtt": [
                                    "CodePipelineArtifactStoreBucket",
                                    "Arn"
                                ]
                            },
                            "/*"
                        ]
                    ]
                },
                "Condition": {
                    "StringNotEquals": {
                        "s3:x-amz-server-side-encryption": "aws:kms"
                    }
                }
            },
            {
                "Sid": "DenyInsecureConnections",
                "Effect": "Deny",
                "Principal": "*",
                "Action": "s3:*",
                "Resource": {
                    "Fn::Join": [
                        "",
                        [
                            {
                                "Fn::GetAtt": [
                                    "CodePipelineArtifactStoreBucket",
                                    "Arn"
                                ]
                            },
                            "/*"
                        ]
                    ]
                },
                "Condition": {
                    "Bool": {
                        "aws:SecureTransport": false
```

```
                                }
                            }
                        }
                    ]
                }
            }
        },
        "CodePipelineServiceRole": {
            "Type": "AWS::IAM::Role",
            "Properties": {
                "AssumeRolePolicyDocument": {
                    "Version": "2012-10-17",
                    "Statement": [
                        {
                            "Effect": "Allow",
                            "Principal": {
                                "Service": [
                                    "codepipeline.amazonaws.com"
                                ]
                            },
                            "Action": "sts:AssumeRole"
                        }
                    ]
                },
                "Path": "/",
                "Policies": [
                    {
                        "PolicyName": "AWS-CodePipeline-Service-3",
                        "PolicyDocument": {
                            "Version": "2012-10-17",
                            "Statement": [
                                {
                                    "Effect": "Allow",
                                    "Action": [
                                        "codecommit:CancelUploadArchive",
                                        "codecommit:GetBranch",
                                        "codecommit:GetCommit",
                                        "codecommit:GetUploadArchiveStatus",
                                        "codecommit:UploadArchive"
                                    ],
                                    "Resource": "resource_ARN"
                                },
                                {
                                    "Effect": "Allow",
```

```
            "Action": [
                "codedeploy:CreateDeployment",
                "codedeploy:GetApplicationRevision",
                "codedeploy:GetDeployment",
                "codedeploy:GetDeploymentConfig",
                "codedeploy:RegisterApplicationRevision"
            ],
            "Resource": "resource_ARN"
        },
        {
            "Effect": "Allow",
            "Action": [
                "codebuild:BatchGetBuilds",
                "codebuild:StartBuild"
            ],
            "Resource": "resource_ARN"
        },
        {
            "Effect": "Allow",
            "Action": [
                "devicefarm:ListProjects",
                "devicefarm:ListDevicePools",
                "devicefarm:GetRun",
                "devicefarm:GetUpload",
                "devicefarm:CreateUpload",
                "devicefarm:ScheduleRun"
            ],
            "Resource": "resource_ARN"
        },
        {
            "Effect": "Allow",
            "Action": [
                "lambda:InvokeFunction",
                "lambda:ListFunctions"
            ],
            "Resource": "resource_ARN"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": "resource_ARN"
        },
```

```
                            {
                                "Effect": "Allow",
                                "Action": [
                                    "elasticbeanstalk:*",
                                    "ec2:*",
                                    "elasticloadbalancing:*",
                                    "autoscaling:*",
                                    "cloudwatch:*",
                                    "s3:*",
                                    "sns:*",
                                    "cloudformation:*",
                                    "rds:*",
                                    "sqs:*",
                                    "ecs:*"
                                ],
                                "Resource": "resource_ARN"
                            }
                        ]
                    }
                }
            ]
        }
    },
    "AppPipeline": {
        "Type": "AWS::CodePipeline::Pipeline",
        "Properties": {
            "Name": "s3-events-pipeline",
            "RoleArn": {
                "Fn::GetAtt": [
                    "CodePipelineServiceRole",
                    "Arn"
                ]
            },
            "Stages": [
                {
                    "Name": "Source",
                    "Actions": [
                        {
                            "Name": "SourceAction",
                            "ActionTypeId": {
                                "Category": "Source",
                                "Owner": "AWS",
                                "Version": 1,
                                "Provider": "S3"
```

```
                        },
                        "OutputArtifacts": [
                            {
                                "Name": "SourceOutput"
                            }
                        ],
                        "Configuration": {
                            "S3Bucket": {
                                "Ref": "SourceBucket"
                            },
                            "S3ObjectKey": {
                                "Ref": "SourceObjectKey"
                            },
                            "PollForSourceChanges": false
                        },
                        "RunOrder": 1
                    }
                ]
            },
            {
                "Name": "Beta",
                "Actions": [
                    {
                        "Name": "BetaAction",
                        "InputArtifacts": [
                            {
                                "Name": "SourceOutput"
                            }
                        ],
                        "ActionTypeId": {
                            "Category": "Deploy",
                            "Owner": "AWS",
                            "Version": 1,
                            "Provider": "CodeDeploy"
                        },
                        "Configuration": {
                            "ApplicationName": {
                                "Ref": "ApplicationName"
                            },
                            "DeploymentGroupName": {
                                "Ref": "BetaFleet"
                            }
                        },
                        "RunOrder": 1
```

```
                        }
                    ]
                }
            ],
            "ArtifactStore": {
                "Type": "S3",
                "Location": {
                    "Ref": "CodePipelineArtifactStoreBucket"
                }
            }
        }
    },
    "EventRole": {
        "Type": "AWS::IAM::Role",
        "Properties": {
            "AssumeRolePolicyDocument": {
                "Version": "2012-10-17",
                "Statement": [
                    {
                        "Effect": "Allow",
                        "Principal": {
                            "Service": [
                                "events.amazonaws.com"
                            ]
                        },
                        "Action": "sts:AssumeRole"
                    }
                ]
            },
            "Path": "/",
            "Policies": [
                {
                    "PolicyName": "eb-pipeline-execution",
                    "PolicyDocument": {
                        "Version": "2012-10-17",
                        "Statement": [
                            {
                                "Effect": "Allow",
                                "Action": "codepipeline:StartPipelineExecution",
                                "Resource": {
                                    "Fn::Join": [
                                        "",
                                        [
                                            "arn:aws:codepipeline:",
```

```
                                                {
                                                    "Ref": "AWS::Region"
                                                },
                                                ":",
                                                {
                                                    "Ref": "AWS::AccountId"
                                                },
                                                ":",
                                                {
                                                    "Ref": "AppPipeline"
                                                }
                                            ]
                                        ]
                                    }
                                }
                            ]
                        }
                    }
                ]
            }
        },
        "EventRule": {
            "Type": "AWS::Events::Rule",

            "Properties": {
                "EventBusName": "default",
          "EventPattern": {
              "source": [
            "aws.s3"
          ],
                "detail-type": [
        "Object Created"
          ],
          "detail": {
        "bucket": {
                        "name": [
                            {
                            "Ref": "SourceBucket"
                            }
                        ]
                    }
                }
    },
    "Name": "EnabledS3SourceRule",
```

```
        "State": "ENABLED",
                "Targets": [
                    {
                        "Arn": {
                            "Fn::Join": [
                                "",
                                [
                                    "arn:aws:codepipeline:",
                                    {
                                        "Ref": "AWS::Region"
                                    },
                                    ":",
                                    {
                                        "Ref": "AWS::AccountId"
                                    },
                                    ":",
                                    {
                                        "Ref": "AppPipeline"
                                    }
                                ]
                            ]
                        },
                        "RoleArn": {
                            "Fn::GetAtt": [
                                "EventRole",
                                "Arn"
                            ]
                        },
                        "Id": "codepipeline-AppPipeline"
                    }
                ]
            }
        }
    }
    }
}
```

使用 S3 來源和 CloudTrail 追蹤移轉輪詢管道

對於具有 Amazon S3 來源的管道，請修改管道，以便透過自動執行變更偵測 EventBridge。從下列方法中選擇以實作移轉：

- 控制台：遷移輪詢管道 (CodeCommit 或 Amazon S3 來源) (主控台)
- CLI：使用 S3 來源和 CloudTrail 追蹤 (CLI) 移轉輪詢管道
- AWS CloudFormation: 使用 S3 來源和 CloudTrail 追蹤 (AWS CloudFormation範本) 移轉輪詢管道

使用 S3 來源和 CloudTrail 追蹤 (CLI) 移轉輪詢管道

請遵循下列步驟來編輯使用輪詢 (定期檢查) EventBridge 改為使用事件的管線。如果您想要建立管道，請參閱在中建立管線 CodePipeline。

若要使用 Amazon S3 建立事件驅動管道，您可以編輯管道的`PollForSourceChanges`參數，然後建立下列資源：

- AWS CloudTrailAmazon S3 可用來記錄事件的追蹤、儲存貯體和儲存貯體政策。
- EventBridge 事件
- IAM 角色，可讓 EventBridge 事件啟動您的管道

建立 AWS CloudTrail 線索並啟用記錄

若要使用 AWS CLI 建立線索，請呼叫 create-trail 命令，指定：

- 線索名稱。
- 您已套用 AWS CloudTrail 儲存貯體政策的儲存貯體。

若要取得更多資訊，請參閱〈使用AWS指令行介面建立系統線〉。

1. 呼叫 create-trail 命令，並包含 `--name` 和 `--s3-bucket-name` 參數。

   為什麼我會做出此變更？ 這會為您的 S3 來源儲存貯體建立所需的 CloudTrail 線索。

   以下命令使用 `--name` 和 `--s3-bucket-name`，來建立名為 `my-trail` 的線索，以及名為 `myBucket` 的儲存貯體。

   ```
   aws cloudtrail create-trail --name my-trail --s3-bucket-name myBucket
   ```

2. 呼叫 start-logging 命令並加入 `--name` 參數。

   我為什麼要進行這項變更？ 此命令會啟動來源值區的 CloudTrail 記錄，並將事件傳送至 EventBridge。

範例：

以下命令範例會使用了 --name，以在名為 my-trail 的線索上啟動日誌記錄。

```
aws cloudtrail start-logging --name my-trail
```

3. 呼叫 put-event-selectors 命令，並包含 --trail-name 和 --event-selectors 參數。使用事件選取器來指定您希望追蹤記錄來源儲存貯體的資料事件，並將事件傳送至 EventBridge 規則。

   我為什麼要進行這項變更？ 此命令會篩選事件。

   範例：

   以下命令範例使用 --trail-name 與 --event-selectors，來指定來源儲存貯體和字首的資料事件，名為 myBucket/myFolder。

```
aws cloudtrail put-event-selectors --trail-name my-trail --event-selectors
 '[{ "ReadWriteType": "WriteOnly", "IncludeManagementEvents":false,
 "DataResources": [{ "Type": "AWS::S3::Object", "Values": ["arn:aws:s3:::myBucket/
myFolder/file.zip"] }] }]'
```

使用 Amazon S3 做為事件來源和目標建立 EventBridge 規則 CodePipeline ，並套用許可政策

1. 授與用 EventBridge CodePipeline 於呼叫規則的權限。如需詳細資訊，請參閱在 Amazon EventBridge 使用以資源為基礎的政策。

   a. 使用下列範例建立信任原則，以 EventBridge 允許擔任服務角色。將其命名為 trustpolicyforEB.json。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "events.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
```

```
}
```

b. 使用下列命令來建立 Role-for-MyRule 角色，並連接信任政策。

為什麼我會做出此變更？ 將此信任原則新增至角色會建立的權限 EventBridge。

```
aws iam create-role --role-name Role-for-MyRule --assume-role-policy-document
 file://trustpolicyforEB.json
```

c. 針對名為 MyFirstPipeline 的管道建立許可政策 JSON，如下所示。將許可政策命名為 permissionspolicyforEB.json。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codepipeline:StartPipelineExecution"
            ],
            "Resource": [
                "arn:aws:codepipeline:us-west-2:80398EXAMPLE:MyFirstPipeline"
            ]
        }
    ]
}
```

d. 使用執行以下命令，將新的 CodePipeline-Permissions-Policy-for-EB 許可政策連接到您所建立的 Role-for-MyRule 角色。

```
aws iam put-role-policy --role-name Role-for-MyRule --policy-name CodePipeline-
Permissions-Policy-For-EB --policy-document file://permissionspolicyforEB.json
```

2. 呼叫 put-rule 命令，並包含 --name、--event-pattern 和 --role-arn 參數。

以下範例命令會建立名為 MyS3SourceRule 的規則。

```
aws events put-rule --name "MyS3SourceRule" --event-pattern "{\"source\":
[\"aws.s3\"],\"detail-type\":[\"AWS API Call via CloudTrail\"],\"detail\":
{\"eventSource\":[\"s3.amazonaws.com\"],\"eventName\":[\"CopyObject\",\"PutObject
\",\"CompleteMultipartUpload\"],\"requestParameters\":{\"bucketName\":[\"my-bucket
\"],\"key\":[\"my-key\"]}}}
```

```
 --role-arn "arn:aws:iam::ACCOUNT_ID:role/Role-for-MyRule"
```

3. 若要新增 CodePipeline 為目標，請呼叫put-targets指令並包含--rule和--targets參數。

   以下命令指定名為 MyS3SourceRule 的規則，該目標 Id 是由數字 1 組成，指出在規則的目標清單中，這是目標 1。此命令也會指定管道的範例 ARN。儲存庫中若發生變更，管道就會啟動。

```
aws events put-targets --rule MyS3SourceRule --targets
 Id=1,Arn=arn:aws:codepipeline:us-west-2:80398EXAMPLE:TestPipeline
```

若要編輯管線的 PollForSourceChanges參數

> **Important**
>
> 當您使用這個方法建立管道時，如果沒有明確設為 false，則 PollForSourceChanges 參數會預設為 true。當新增基於事件的變更偵測時，您必須將該參數新增到輸出，並將其設為 false 以停用輪詢。否則，您的管道會針對單一來源變更啟動兩次。如需詳細資訊，請參閱 PollForSourceChanges 參數的預設設定。

1. 執行 get-pipeline 命令，將管道結構複製到 JSON 檔案。例如，針對名為 MyFirstPipeline 的管道，執行下列命令：

```
aws codepipeline get-pipeline --name MyFirstPipeline >pipeline.json
```

   此命令不會傳回任何內容，但您建立的檔案應該會顯示在您執行命令的目錄中。

2. 在任何純文字編輯器中開啟 JSON 檔案，然後將名為 storage-bucket 之儲存貯體的 PollForSourceChanges 參數變更為 false，來編輯來源階段，如這個範例所示。

   為什麼我會做出此變更？ 將此參數設為 false 會關閉定期檢查，因此您只能使用事件型變更偵測。

```
"configuration": {
    "S3Bucket": "storage-bucket",
    "PollForSourceChanges": "false",
    "S3ObjectKey": "index.zip"
},
```

3. 如果您使用的是使用 get-pipeline 命令擷取的管道結構，則必須從 JSON 檔案中移除 `metadata` 行。否則，update-pipeline 命令無法使用它。移除 `"metadata": { }` 行，以及 `"created"`、`"pipelineARN"` 和 `"updated"` 欄位。

   例如，從結構中移除下列幾行：

   ```
   "metadata": {
       "pipelineArn": "arn:aws:codepipeline:region:account-ID:pipeline-name",
       "created": "date",
       "updated": "date"
   },
   ```

   儲存檔案。

4. 若要套用您的變更，請執行 update-pipeline 命令、指定管道 JSON 檔案：

   > **Important**
   >
   > 請確認在檔案名稱之前包含 `file://`。這是此命令必要項目。

   ```
   aws codepipeline update-pipeline --cli-input-json file://pipeline.json
   ```

   此命令會傳回所編輯管道的整個結構。

   > **Note**
   >
   > update-pipeline 命令將終止管道。若在您執行 update-pipeline 命令時有修訂正在透過管道執行，該執行將停止。您必須手動啟動管道，以透過更新的管道執行該修訂。使用 start-pipeline-execution 命令來手動啟動您的管道。

使用 S3 來源和 CloudTrail 追蹤 (AWS CloudFormation範本) 移轉輪詢管道

使用以下步驟透過 Amazon S3 來源編輯管道，從輪詢到事件型變更偵測。

若要使用 Amazon S3 建立事件驅動管道，您可以編輯管道的`PollForSourceChanges`參數，然後將下列資源新增至範本：

- EventBridge 要求必須記錄所有 Amazon S3 事件。您必須建立AWS CloudTrail追蹤、儲存貯體和儲存貯體政策，Amazon S3 可用來記錄發生的事件。如需詳細資訊，請參閱記錄追蹤的資料事件和記錄追蹤的管理事件。
- EventBridge 規則和 IAM 角色，以允許此事件啟動我們的管道。

如果您使用 AWS CloudFormation 來建立和管理您的管道，則您的範本會包含如下的內容。

> **Note**
>
> 來源階段 (稱為 PollForSourceChanges) 中的 Configuration 屬性。如果您的範本未包含該屬性，則 PollForSourceChanges 會預設為 true。

YAML

```
AppPipeline:
  Type: AWS::CodePipeline::Pipeline
  Properties:
    RoleArn: !GetAtt CodePipelineServiceRole.Arn
    Stages:
      -
        Name: Source
        Actions:
          -
            Name: SourceAction
            ActionTypeId:
              Category: Source
              Owner: AWS
              Version: 1
              Provider: S3
            OutputArtifacts:
              -
                Name: SourceOutput
            Configuration:
              S3Bucket: !Ref SourceBucket
              S3ObjectKey: !Ref S3SourceObjectKey
              PollForSourceChanges: true
            RunOrder: 1
```

```
...
```

JSON

```
        "AppPipeline": {
            "Type": "AWS::CodePipeline::Pipeline",
            "Properties": {
                "RoleArn": {
                    "Fn::GetAtt": ["CodePipelineServiceRole", "Arn"]
                },
                "Stages": [
                    {
                        "Name": "Source",
                        "Actions": [
                            {
                                "Name": "SourceAction",
                                "ActionTypeId": {
                                    "Category": "Source",
                                    "Owner": "AWS",
                                    "Version": 1,
                                    "Provider": "S3"
                                },
                                "OutputArtifacts": [
                                    {
                                        "Name": "SourceOutput"
                                    }
                                ],
                                "Configuration": {
                                    "S3Bucket": {
                                        "Ref": "SourceBucket"
                                    },
                                    "S3ObjectKey": {
                                        "Ref": "SourceObjectKey"
                                    },
                                    "PollForSourceChanges": true
                                },
                                "RunOrder": 1
                            }
                        ]
                    },


...
```

使用 Amazon S3 做為事件來源和目標建立 EventBridge 規則 CodePipeline ，並套用許可政策

1. 在範本的 Resources 下，使用 AWS::IAM::Role AWS CloudFormation 資源，來設定允許事件啟動管道的 IAM 角色。此項目會建立一個使用兩個政策的角色：

   - 第一個政策允許要承擔的角色。
   - 第二個政策提供啟動管道的許可。

   為什麼我會做出此變更？ 新增AWS::IAM::Role資源可AWS CloudFormation建立的權限 EventBridge。此資源會新增至您的 AWS CloudFormation 堆疊。

   YAML

```
  EventRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: 2012-10-17
        Statement:
          -
            Effect: Allow
            Principal:
              Service:
                - events.amazonaws.com
            Action: sts:AssumeRole
      Path: /
      Policies:
        -
          PolicyName: eb-pipeline-execution
          PolicyDocument:
            Version: 2012-10-17
            Statement:
              -
                Effect: Allow
                Action: codepipeline:StartPipelineExecution
                Resource: !Join [ '', [ 'arn:aws:codepipeline:', !Ref
 'AWS::Region', ':', !Ref 'AWS::AccountId', ':', !Ref AppPipeline ] ]


...
```

JSON

```
"EventRole": {
  "Type": "AWS::IAM::Role",
  "Properties": {
    "AssumeRolePolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "Service": [
              "events.amazonaws.com"
            ]
          },
          "Action": "sts:AssumeRole"
        }
      ]
    },
    "Path": "/",
    "Policies": [
      {
        "PolicyName": "eb-pipeline-execution",
        "PolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [
            {
              "Effect": "Allow",
              "Action": "codepipeline:StartPipelineExecution",
              "Resource": {
                "Fn::Join": [
                  "",
                  [
                    "arn:aws:codepipeline:",
                    {
                      "Ref": "AWS::Region"
                    },
                    ":",
                    {
                      "Ref": "AWS::AccountId"
                    },
                    ":",
                    {
```

```
                        "Ref": "AppPipeline"
                      }
                    ]
                  ]

...
```

2. 使用AWS::Events::RuleAWS CloudFormation資源新增 EventBridge 規則。此事件模式會在您的 Amazon S3 來源儲存貯體CompleteMultipartUpload上建立監控CopyObject事件。PutObject此外，會包含您管道的目標。當 CopyObject、PutObject 或 CompleteMultipartUpload 發生時，此規則會在目標管道上呼叫 StartPipelineExecution。

   為什麼我會做出此變更？ 新增 AWS::Events::Rule 資源可讓 AWS CloudFormation 建立事件。此資源會新增至您的 AWS CloudFormation 堆疊。

   YAML

```
  EventRule:
    Type: AWS::Events::Rule
    Properties:
      EventPattern:
        source:
          - aws.s3
        detail-type:
          - 'AWS API Call via CloudTrail'
        detail:
          eventSource:
            - s3.amazonaws.com
          eventName:
            - CopyObject
            - PutObject
            - CompleteMultipartUpload
          requestParameters:
            bucketName:
              - !Ref SourceBucket
            key:
              - !Ref SourceObjectKey
      Targets:
        -
          Arn:
            !Join [ '', [ 'arn:aws:codepipeline:', !Ref 'AWS::Region', ':', !Ref
 'AWS::AccountId', ':', !Ref AppPipeline ] ]
```

```
          RoleArn: !GetAtt EventRole.Arn
          Id: codepipeline-AppPipeline


...
```

JSON

```
 "EventRule": {
   "Type": "AWS::Events::Rule",
   "Properties": {
     "EventPattern": {
       "source": [
         "aws.s3"
       ],
       "detail-type": [
         "AWS API Call via CloudTrail"
       ],
       "detail": {
         "eventSource": [
           "s3.amazonaws.com"
         ],
         "eventName": [
           "CopyObject",
           "PutObject",
           "CompleteMultipartUpload"
         ],
         "requestParameters": {
           "bucketName": [
             {
               "Ref": "SourceBucket"
             }
           ],
           "key": [
             {
               "Ref": "SourceObjectKey"
             }
           ]
         }
       }
     },
     "Targets": [
```

```
        {
          "Arn": {
            "Fn::Join": [
              "",
              [
                "arn:aws:codepipeline:",
                {
                  "Ref": "AWS::Region"
                },
                ":",
                {
                  "Ref": "AWS::AccountId"
                },
                ":",
                {
                  "Ref": "AppPipeline"
                }
              ]
            ]
          },
          "RoleArn": {
            "Fn::GetAtt": [
              "EventRole",
              "Arn"
            ]
          },
          "Id": "codepipeline-AppPipeline"
        }
      ]
    }
  }
},

...
```

3. 將此片段新增到您的第一個範本，以允許跨堆疊功能：

   YAML

```
Outputs:
  SourceBucketARN:
    Description: "S3 bucket ARN that Cloudtrail will use"
    Value: !GetAtt SourceBucket.Arn
    Export:
```

```
      Name: SourceBucketARN
```

JSON

```
  "Outputs" : {
    "SourceBucketARN" : {
      "Description" : "S3 bucket ARN that Cloudtrail will use",
      "Value" : { "Fn::GetAtt": ["SourceBucket", "Arn"] },
      "Export" : {
        "Name" : "SourceBucketARN"
      }
    }

...
```

4. 儲存您的更新範本到本機電腦，並開啟 AWS CloudFormation 主控台。
5. 選擇您的堆疊，然後選擇 Create Change Set for Current Stack (建立目前堆疊的變更集)。
6. 上傳您的更新範本，然後檢視中 AWS CloudFormation 所列的變更。這些是會針對堆疊進行的變更。您應該會在清單中看到新資源。
7. 選擇 Execute (執行)。

若要編輯管線的 PollForSourceChanges參數

> **Important**
>
> 當您使用這個方法建立管道時，如果沒有明確設為 false，則 `PollForSourceChanges` 參數會預設為 true。當新增基於事件的變更偵測時，您必須將該參數新增到輸出，並將其設為 false 以停用輪詢。否則，您的管道會針對單一來源變更啟動兩次。如需詳細資訊，請參閱 PollForSourceChanges 參數的預設設定。

- 在範本中，將 `PollForSourceChanges` 變更為 `false`。如果您並未在管道定義中包含 `PollForSourceChanges`，請新增它，並將其設為 `false`。

  為什麼我會做出此變更？ 將 `PollForSourceChanges`變更為 `false` 會關閉定期檢查，因此您只能使用事件型變更偵測。

YAML

```
        Name: Source
        Actions:
          -
            Name: SourceAction
            ActionTypeId:
              Category: Source
              Owner: AWS
              Version: 1
              Provider: S3
            OutputArtifacts:
              - Name: SourceOutput
            Configuration:
              S3Bucket: !Ref SourceBucket
              S3ObjectKey: !Ref SourceObjectKey
              PollForSourceChanges: false
            RunOrder: 1
```

JSON

```
{
   "Name": "SourceAction",
   "ActionTypeId": {
     "Category": "Source",
     "Owner": "AWS",
     "Version": 1,
     "Provider": "S3"
   },
   "OutputArtifacts": [
     {
       "Name": "SourceOutput"
     }
   ],
   "Configuration": {
     "S3Bucket": {
       "Ref": "SourceBucket"
     },
     "S3ObjectKey": {
       "Ref": "SourceObjectKey"
     },
     "PollForSourceChanges": false
```

```
    },
    "RunOrder": 1
  }
```

若要為 Amazon S3 管道的 CloudTrail 資源建立第二個範本

- 在單獨範本中的 Resources 下，使用 AWS::S3::Bucket、AWS::S3::BucketPolicy 和 AWS::CloudTrail::Trail AWS CloudFormation 資源，以提供簡單的儲存貯體定義和 CloudTrail 的線索。

  我為什麼要進行這項變更？ 鑑於每個帳戶目前有五個 CloudTrail 追蹤的限制，必須個別建立和管理追蹤。(請參閱中的限制AWS CloudTrail。) 不過，您可以在單一追蹤中包含多個 Amazon S3 儲存貯體，因此您可以建立一次追蹤，然後視需要為其他管道新增 Amazon S3 儲存貯體。將下列內容貼至您的第二個範例範本檔案中。

  YAML

```
################################################################################
# Prerequisites:
#   - S3 SourceBucket and SourceObjectKey must exist
################################################################################

Parameters:
  SourceObjectKey:
    Description: 'S3 source artifact'
    Type: String
    Default: SampleApp_Linux.zip

Resources:
  AWSCloudTrailBucketPolicy:
    Type: AWS::S3::BucketPolicy
    Properties:
      Bucket: !Ref AWSCloudTrailBucket
      PolicyDocument:
        Version: 2012-10-17
        Statement:
          -
            Sid: AWSCloudTrailAclCheck
            Effect: Allow
            Principal:
```

```
              Service:
                - cloudtrail.amazonaws.com
            Action: s3:GetBucketAcl
            Resource: !GetAtt AWSCloudTrailBucket.Arn
          -
            Sid: AWSCloudTrailWrite
            Effect: Allow
            Principal:
              Service:
                - cloudtrail.amazonaws.com
            Action: s3:PutObject
            Resource: !Join [ '', [ !GetAtt AWSCloudTrailBucket.Arn, '/
AWSLogs/', !Ref 'AWS::AccountId', '/*' ] ]
            Condition:
              StringEquals:
                s3:x-amz-acl: bucket-owner-full-control
  AWSCloudTrailBucket:
    Type: AWS::S3::Bucket
    DeletionPolicy: Retain
  AwsCloudTrail:
    DependsOn:
      - AWSCloudTrailBucketPolicy
    Type: AWS::CloudTrail::Trail
    Properties:
      S3BucketName: !Ref AWSCloudTrailBucket
      EventSelectors:
        -
          DataResources:
            -
              Type: AWS::S3::Object
              Values:
                - !Join [ '', [ !ImportValue SourceBucketARN, '/', !Ref
 SourceObjectKey ] ]
          ReadWriteType: WriteOnly
          IncludeManagementEvents: false
      IncludeGlobalServiceEvents: true
      IsLogging: true
      IsMultiRegionTrail: true


...
```

JSON

```
{
  "Parameters": {
    "SourceObjectKey": {
      "Description": "S3 source artifact",
      "Type": "String",
      "Default": "SampleApp_Linux.zip"
    }
  },
  "Resources": {
    "AWSCloudTrailBucket": {
      "Type": "AWS::S3::Bucket",
        "DeletionPolicy": "Retain"
    },
    "AWSCloudTrailBucketPolicy": {
      "Type": "AWS::S3::BucketPolicy",
      "Properties": {
        "Bucket": {
          "Ref": "AWSCloudTrailBucket"
        },
        "PolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [
            {
              "Sid": "AWSCloudTrailAclCheck",
              "Effect": "Allow",
              "Principal": {
                "Service": [
                  "cloudtrail.amazonaws.com"
                ]
              },
              "Action": "s3:GetBucketAcl",
              "Resource": {
                "Fn::GetAtt": [
                  "AWSCloudTrailBucket",
                  "Arn"
                ]
              }
            },
            {
              "Sid": "AWSCloudTrailWrite",
              "Effect": "Allow",
```

```
            "Principal": {
              "Service": [
                "cloudtrail.amazonaws.com"
              ]
            },
            "Action": "s3:PutObject",
            "Resource": {
              "Fn::Join": [
                "",
                [
                  {
                    "Fn::GetAtt": [
                      "AWSCloudTrailBucket",
                      "Arn"
                    ]
                  },
                  "/AWSLogs/",
                  {
                    "Ref": "AWS::AccountId"
                  },
                  "/*"
                ]
              ]
            },
            "Condition": {
              "StringEquals": {
                "s3:x-amz-acl": "bucket-owner-full-control"
              }
            }
          }
        ]
      }
    }
  },
  "AwsCloudTrail": {
    "DependsOn": [
      "AWSCloudTrailBucketPolicy"
    ],
    "Type": "AWS::CloudTrail::Trail",
    "Properties": {
      "S3BucketName": {
        "Ref": "AWSCloudTrailBucket"
      },
      "EventSelectors": [
```

```
          {
            "DataResources": [
              {
                "Type": "AWS::S3::Object",
                "Values": [
                  {
                    "Fn::Join": [
                      "",
                      [
                        {
                          "Fn::ImportValue": "SourceBucketARN"
                        },
                        "/",
                        {
                          "Ref": "SourceObjectKey"
                        }
                      ]
                    ]
                  }
                ]
              }
            ],
            "ReadWriteType": "WriteOnly",
            "IncludeManagementEvents": false
          }
        ],
        "IncludeGlobalServiceEvents": true,
        "IsLogging": true,
        "IsMultiRegionTrail": true
      }
    }
  }
}

...
```

Example

如果您使用 AWS CloudFormation 建立這些資源，則在建立或更新儲存庫中的檔案時，會觸發您的管道。

> **Note**
>
> 不要在此處停止。雖然管道已建立，但您必須為 Amazon S3 管道建立第二個AWS CloudFormation範本。如果您未建立第二個範本，您的管道不會有任何變更偵測功能。

YAML

```
Resources:
  SourceBucket:
    Type: AWS::S3::Bucket
    Properties:
      VersioningConfiguration:
        Status: Enabled
  CodePipelineArtifactStoreBucket:
    Type: AWS::S3::Bucket
  CodePipelineArtifactStoreBucketPolicy:
    Type: AWS::S3::BucketPolicy
    Properties:
      Bucket: !Ref CodePipelineArtifactStoreBucket
      PolicyDocument:
        Version: 2012-10-17
        Statement:
          -
            Sid: DenyUnEncryptedObjectUploads
            Effect: Deny
            Principal: '*'
            Action: s3:PutObject
            Resource: !Join [ '', [ !GetAtt CodePipelineArtifactStoreBucket.Arn, '/
*' ] ]
            Condition:
              StringNotEquals:
                s3:x-amz-server-side-encryption: aws:kms
          -
            Sid: DenyInsecureConnections
            Effect: Deny
            Principal: '*'
            Action: s3:*
            Resource: !Join [ '', [ !GetAtt CodePipelineArtifactStoreBucket.Arn, '/
*' ] ]
            Condition:
              Bool:
                aws:SecureTransport: false
```

```
  CodePipelineServiceRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: 2012-10-17
        Statement:
          -
            Effect: Allow
            Principal:
              Service:
                - codepipeline.amazonaws.com
            Action: sts:AssumeRole
      Path: /
      Policies:
        -
          PolicyName: AWS-CodePipeline-Service-3
          PolicyDocument:
            Version: 2012-10-17
            Statement:
              -
                Effect: Allow
                Action:
                  - codecommit:CancelUploadArchive
                  - codecommit:GetBranch
                  - codecommit:GetCommit
                  - codecommit:GetUploadArchiveStatus
                  - codecommit:UploadArchive
                Resource: 'resource_ARN'
              -
                Effect: Allow
                Action:
                  - codedeploy:CreateDeployment
                  - codedeploy:GetApplicationRevision
                  - codedeploy:GetDeployment
                  - codedeploy:GetDeploymentConfig
                  - codedeploy:RegisterApplicationRevision
                Resource: 'resource_ARN'
              -
                Effect: Allow
                Action:
                  - codebuild:BatchGetBuilds
                  - codebuild:StartBuild
                Resource: 'resource_ARN'
              -
```

```
                Effect: Allow
                Action:
                  - devicefarm:ListProjects
                  - devicefarm:ListDevicePools
                  - devicefarm:GetRun
                  - devicefarm:GetUpload
                  - devicefarm:CreateUpload
                  - devicefarm:ScheduleRun
                Resource: 'resource_ARN'
              -
                Effect: Allow
                Action:
                  - lambda:InvokeFunction
                  - lambda:ListFunctions
                Resource: 'resource_ARN'
              -
                Effect: Allow
                Action:
                  - iam:PassRole
                Resource: 'resource_ARN'
              -
                Effect: Allow
                Action:
                  - elasticbeanstalk:*
                  - ec2:*
                  - elasticloadbalancing:*
                  - autoscaling:*
                  - cloudwatch:*
                  - s3:*
                  - sns:*
                  - cloudformation:*
                  - rds:*
                  - sqs:*
                  - ecs:*
                Resource: 'resource_ARN'
  AppPipeline:
    Type: AWS::CodePipeline::Pipeline
    Properties:
      Name: s3-events-pipeline
      RoleArn:
        !GetAtt CodePipelineServiceRole.Arn
      Stages:
        -
          Name: Source
```

```
          Actions:
            -
              Name: SourceAction
              ActionTypeId:
                Category: Source
                Owner: AWS
                Version: 1
                Provider: S3
              OutputArtifacts:
                - Name: SourceOutput
              Configuration:
                S3Bucket: !Ref SourceBucket
                S3ObjectKey: !Ref SourceObjectKey
                PollForSourceChanges: false
              RunOrder: 1
        -
          Name: Beta
          Actions:
            -
              Name: BetaAction
              InputArtifacts:
                - Name: SourceOutput
              ActionTypeId:
                Category: Deploy
                Owner: AWS
                Version: 1
                Provider: CodeDeploy
              Configuration:
                ApplicationName: !Ref ApplicationName
                DeploymentGroupName: !Ref BetaFleet
              RunOrder: 1
      ArtifactStore:
        Type: S3
        Location: !Ref CodePipelineArtifactStoreBucket
  EventRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: 2012-10-17
        Statement:
          -
            Effect: Allow
            Principal:
              Service:
```

```
                - events.amazonaws.com
            Action: sts:AssumeRole
      Path: /
      Policies:
        -
          PolicyName: eb-pipeline-execution
          PolicyDocument:
            Version: 2012-10-17
            Statement:
              -
                Effect: Allow
                Action: codepipeline:StartPipelineExecution
                Resource: !Join [ '', [ 'arn:aws:codepipeline:', !Ref 'AWS::Region',
':', !Ref 'AWS::AccountId', ':', !Ref AppPipeline ] ]
  EventRule:
    Type: AWS::Events::Rule
    Properties:
      EventPattern:
        source:
          - aws.s3
        detail-type:
          - 'AWS API Call via CloudTrail'
        detail:
          eventSource:
            - s3.amazonaws.com
          eventName:
            - PutObject
            - CompleteMultipartUpload
          resources:
            ARN:
              - !Join [ '', [ !GetAtt SourceBucket.Arn, '/', !Ref
 SourceObjectKey ] ]
      Targets:
        -
          Arn:
            !Join [ '', [ 'arn:aws:codepipeline:', !Ref 'AWS::Region', ':', !Ref
 'AWS::AccountId', ':', !Ref AppPipeline ] ]
          RoleArn: !GetAtt EventRole.Arn
          Id: codepipeline-AppPipeline

Outputs:
  SourceBucketARN:
    Description: "S3 bucket ARN that Cloudtrail will use"
    Value: !GetAtt SourceBucket.Arn
```

```
    Export:
      Name: SourceBucketARN
```

JSON

```
"Resources": {
    "SourceBucket": {
        "Type": "AWS::S3::Bucket",
        "Properties": {
            "VersioningConfiguration": {
                "Status": "Enabled"
            }
        }
    },
    "CodePipelineArtifactStoreBucket": {
        "Type": "AWS::S3::Bucket"
    },
    "CodePipelineArtifactStoreBucketPolicy": {
        "Type": "AWS::S3::BucketPolicy",
        "Properties": {
            "Bucket": {
                "Ref": "CodePipelineArtifactStoreBucket"
            },
            "PolicyDocument": {
                "Version": "2012-10-17",
                "Statement": [
                    {
                        "Sid": "DenyUnEncryptedObjectUploads",
                        "Effect": "Deny",
                        "Principal": "*",
                        "Action": "s3:PutObject",
                        "Resource": {
                            "Fn::Join": [
                                "",
                                [
                                    {
                                        "Fn::GetAtt": [
                                            "CodePipelineArtifactStoreBucket",
                                            "Arn"
                                        ]
                                    },
                                    "/*"
                                ]
```

```
                            ]
                        },
                        "Condition": {
                            "StringNotEquals": {
                                "s3:x-amz-server-side-encryption": "aws:kms"
                            }
                        }
                    },
                    {
                        "Sid": "DenyInsecureConnections",
                        "Effect": "Deny",
                        "Principal": "*",
                        "Action": "s3:*",
                        "Resource": {
                            "Fn::Join": [
                                "",
                                [
                                    {
                                        "Fn::GetAtt": [
                                            "CodePipelineArtifactStoreBucket",
                                            "Arn"
                                        ]
                                    },
                                    "/*"
                                ]
                            ]
                        },
                        "Condition": {
                            "Bool": {
                                "aws:SecureTransport": false
                            }
                        }
                    }
                ]
            }
        }
    },
    "CodePipelineServiceRole": {
        "Type": "AWS::IAM::Role",
        "Properties": {
            "AssumeRolePolicyDocument": {
                "Version": "2012-10-17",
                "Statement": [
                    {
```

```
                        "Effect": "Allow",
                        "Principal": {
                            "Service": [
                                "codepipeline.amazonaws.com"
                            ]
                        },
                        "Action": "sts:AssumeRole"
                    }
                ]
            },
            "Path": "/",
            "Policies": [
                {
                    "PolicyName": "AWS-CodePipeline-Service-3",
                    "PolicyDocument": {
                        "Version": "2012-10-17",
                        "Statement": [
                            {
                                "Effect": "Allow",
                                "Action": [
                                    "codecommit:CancelUploadArchive",
                                    "codecommit:GetBranch",
                                    "codecommit:GetCommit",
                                    "codecommit:GetUploadArchiveStatus",
                                    "codecommit:UploadArchive"
                                ],
                                "Resource": "resource_ARN"
                            },
                            {
                                "Effect": "Allow",
                                "Action": [
                                    "codedeploy:CreateDeployment",
                                    "codedeploy:GetApplicationRevision",
                                    "codedeploy:GetDeployment",
                                    "codedeploy:GetDeploymentConfig",
                                    "codedeploy:RegisterApplicationRevision"
                                ],
                                "Resource": "resource_ARN"
                            },
                            {
                                "Effect": "Allow",
                                "Action": [
                                    "codebuild:BatchGetBuilds",
                                    "codebuild:StartBuild"
```

```
            ],
            "Resource": "resource_ARN"
        },
        {
            "Effect": "Allow",
            "Action": [
                "devicefarm:ListProjects",
                "devicefarm:ListDevicePools",
                "devicefarm:GetRun",
                "devicefarm:GetUpload",
                "devicefarm:CreateUpload",
                "devicefarm:ScheduleRun"
            ],
            "Resource": "resource_ARN"
        },
        {
            "Effect": "Allow",
            "Action": [
                "lambda:InvokeFunction",
                "lambda:ListFunctions"
            ],
            "Resource": "resource_ARN"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": "resource_ARN"
        },
        {
            "Effect": "Allow",
            "Action": [
                "elasticbeanstalk:*",
                "ec2:*",
                "elasticloadbalancing:*",
                "autoscaling:*",
                "cloudwatch:*",
                "s3:*",
                "sns:*",
                "cloudformation:*",
                "rds:*",
                "sqs:*",
                "ecs:*"
```

```
                                ],
                                "Resource": "resource_ARN"
                            }
                        ]
                    }
                }
            ]
        }
    },
    "AppPipeline": {
        "Type": "AWS::CodePipeline::Pipeline",
        "Properties": {
            "Name": "s3-events-pipeline",
            "RoleArn": {
                "Fn::GetAtt": [
                    "CodePipelineServiceRole",
                    "Arn"
                ]
            },
            "Stages": [
                {
                    "Name": "Source",
                    "Actions": [
                        {
                            "Name": "SourceAction",
                            "ActionTypeId": {
                                "Category": "Source",
                                "Owner": "AWS",
                                "Version": 1,
                                "Provider": "S3"
                            },
                            "OutputArtifacts": [
                                {
                                    "Name": "SourceOutput"
                                }
                            ],
                            "Configuration": {
                                "S3Bucket": {
                                    "Ref": "SourceBucket"
                                },
                                "S3ObjectKey": {
                                    "Ref": "SourceObjectKey"
                                },
                                "PollForSourceChanges": false
```

```
                            },
                            "RunOrder": 1
                        }
                    ]
                },
                {
                    "Name": "Beta",
                    "Actions": [
                        {
                            "Name": "BetaAction",
                            "InputArtifacts": [
                                {
                                    "Name": "SourceOutput"
                                }
                            ],
                            "ActionTypeId": {
                                "Category": "Deploy",
                                "Owner": "AWS",
                                "Version": 1,
                                "Provider": "CodeDeploy"
                            },
                            "Configuration": {
                                "ApplicationName": {
                                    "Ref": "ApplicationName"
                                },
                                "DeploymentGroupName": {
                                    "Ref": "BetaFleet"
                                }
                            },
                            "RunOrder": 1
                        }
                    ]
                }
            ],
            "ArtifactStore": {
                "Type": "S3",
                "Location": {
                    "Ref": "CodePipelineArtifactStoreBucket"
                }
            }
        }
    },
    "EventRole": {
        "Type": "AWS::IAM::Role",
```

```
            "Properties": {
                "AssumeRolePolicyDocument": {
                    "Version": "2012-10-17",
                    "Statement": [
                        {
                            "Effect": "Allow",
                            "Principal": {
                                "Service": [
                                    "events.amazonaws.com"
                                ]
                            },
                            "Action": "sts:AssumeRole"
                        }
                    ]
                },
                "Path": "/",
                "Policies": [
                    {
                        "PolicyName": "eb-pipeline-execution",
                        "PolicyDocument": {
                            "Version": "2012-10-17",
                            "Statement": [
                                {
                                    "Effect": "Allow",
                                    "Action": "codepipeline:StartPipelineExecution",
                                    "Resource": {
                                        "Fn::Join": [
                                            "",
                                            [
                                                "arn:aws:codepipeline:",
                                                {
                                                    "Ref": "AWS::Region"
                                                },
                                                ":",
                                                {
                                                    "Ref": "AWS::AccountId"
                                                },
                                                ":",
                                                {
                                                    "Ref": "AppPipeline"
                                                }
                                            ]
                                        ]
                                    }
```

```
                            }
                        ]
                    }
                }
            ]
        }
    },
    "EventRule": {
        "Type": "AWS::Events::Rule",
        "Properties": {
            "EventPattern": {
                "source": [
                    "aws.s3"
                ],
                "detail-type": [
                    "AWS API Call via CloudTrail"
                ],
                "detail": {
                    "eventSource": [
                        "s3.amazonaws.com"
                    ],
                    "eventName": [
                        "PutObject",
                        "CompleteMultipartUpload"
                    ],
                    "resources": {
                        "ARN": [
                            {
                                "Fn::Join": [
                                    "",
                                    [
                                        {
                                            "Fn::GetAtt": [
                                                "SourceBucket",
                                                "Arn"
                                            ]
                                        },
                                        "/",
                                        {
                                            "Ref": "SourceObjectKey"
                                        }
                                    ]
                                ]
                            }
```

```
                        ]
                    }
                }
            },
            "Targets": [
                {
                    "Arn": {
                        "Fn::Join": [
                            "",
                            [
                                "arn:aws:codepipeline:",
                                {
                                    "Ref": "AWS::Region"
                                },
                                ":",
                                {
                                    "Ref": "AWS::AccountId"
                                },
                                ":",
                                {
                                    "Ref": "AppPipeline"
                                }
                            ]
                        ]
                    },
                    "RoleArn": {
                        "Fn::GetAtt": [
                            "EventRole",
                            "Arn"
                        ]
                    },
                    "Id": "codepipeline-AppPipeline"
                }
            ]
        }
    }
},
"Outputs" : {
    "SourceBucketARN" : {
        "Description" : "S3 bucket ARN that Cloudtrail will use",
        "Value" : { "Fn::GetAtt": ["SourceBucket", "Arn"] },
        "Export" : {
            "Name" : "SourceBucketARN"
        }
```

```
        }
    }
}


...
```

將第 1 GitHub 版來源動作的輪詢管道移轉至連線

您可以移轉第 1 GitHub 版來源動作，以使用外部存放庫的連線。對於具有 GitHub版本 1 來源動作的管線，建議使用這種變更偵測方法。

對於具有第 1 GitHub 版來源動作的管道，我們建議您修改管道以使用第 2 GitHub 版動作，以便透過「AWS CodeStar 連線」自動執行變更偵測。若要取得有關使用連接的更多資訊，請參閱GitHub 連接。

建立連線至 GitHub (主控台)

您可以使用控制台建立連線 GitHub。

步驟 1：取代您的版本 1 GitHub 動作

使用管線編輯頁面，將您的版本 1 GitHub 動作取代為第 2 版 GitHub 動作。

若要取代您的版本 1 GitHub 動作

1. 登入 CodePipeline 主控台。
2. 選擇您的管道，然後選擇 「編輯」。在來源階段中選擇「編輯階段」。會顯示一則訊息，建議您更新動作。
3. 在 [動作提供者] 中，選擇 [GitHub 版本 2]。
4. 執行下列任意一項：

   - 在 [連線] 下方，如果您尚未建立與提供者的連線，請選擇 [Connect 線至] GitHub。繼續執行步驟 2：建立連線 GitHub。
   - 在 [連線] 下方，如果您已建立與提供者的連線，請選擇連線。繼續執行步驟 3：儲存連線的來源動作。

步驟 2：建立連線 GitHub

選擇建立連線之後，會顯示 [Connect 線至 GitHub] 頁面。

若要建立連線 GitHub

1. 在「GitHub 連線設定」 下，您的連線名稱會顯示在「連線名稱」中。

   在 [GitHub 應用程式] 下方，選擇應用程式安裝，或選擇 [安裝新的應用程式] 來建立

   > **Note**
   > 您可以為您連至特定供應商的所有連線安裝一個應用程式。如果您已經安裝了該 GitHub 應用程序，請選擇它並跳過此步驟。

2. 如果 GitHub 顯示的授權頁面，請使用您的認證登入，然後選擇繼續。
3. 在應用程式安裝頁面上，會有訊息顯示AWS CodeStar應用程式正在嘗試連線到您的 GitHub 帳戶。

   > **Note**
   > 您只能為每個 GitHub 帳戶安裝一次該應用程序。如果您先前已安裝應用程式，可以選擇 Configure (設定)，繼續前往應用程式安裝的修改頁面，或者您可以使用上一步按鈕返回主控台。

4. 在 [安裝 AWS CodeStar] 頁面上，選擇 [安裝]。
5. 在 [Connect 線至 GitHub] 頁面上，會顯示新安裝的連線 ID。選擇 Connect (連線)。

步驟 3：儲存您的 GitHub 來源動作

在 「編輯」動作頁面上完成更新，以儲存新的來源動作。

若要儲存 GitHub 來源動作

1. 在存放庫中，輸入第三方存放庫的名稱。在「分支」中，輸入您希望管線偵測來源變更的分支。

   > **Note**
   > 在存放庫中，輸入`owner-name/repository-name`如下範例所示：

```
my-account/my-repository
```

2. 在 [輸出成品格式] 中，選擇成品的格式。

   - 若要使用預設方法儲存 GitHub 動作的輸出成品，請選擇CodePipeline預設值。此動作會從存 GitHub 放庫存取檔案，並將人工因素儲存在管線人工因素存放區中的 ZIP 檔案中。
   - 若要存放包含儲存庫 URL 參考的 JSON 檔案，以便下游動作可以直接執行 Git 命令，請選擇 Full clone (完整複製)。此選項只能由 CodeBuild 下游動作使用。

     如果您選擇此選項，則需要更新 CodeBuild 專案服務角色的權限，如中所示新增連線至 Bitbucket GitHub、 GitHub 企業伺服器或 GitLab .com 的 CodeBuild GitClone 權限。如需說明如何使用「完整複製」選項的教學課程，請參閱教學課程：搭 GitHub 配管線來源使用完整複製。

3. 在輸出人工因素中，您可以保留此動作的輸出成品名稱，例如SourceArtifact。選擇「完成」以關閉「編輯」動作頁面。
4. 選擇「完成」以關閉階段編輯頁面。選擇「儲存」以關閉配管編輯頁面。

建立與 GitHub (CLI) 的連線

您可以使用 AWS Command Line Interface (AWS CLI) 建立與的連接 GitHub。

若要這麼做，請使用 create-connection 命令。

> **Important**
>
> 依預設，透過 AWS CLI 或 AWS CloudFormation 建立的連線會處於 PENDING 狀態。建立連至 CLI 或 AWS CloudFormation 的連線後，請使用主控台編輯連線，將其狀態設為 AVAILABLE。

若要建立連線 GitHub

1. 開啟終端機 (Linux、macOS 或 Unix) 或命令提示 (Windows)。使用 AWS CLI 來執行 create-connection 命令，為連線指定 `--connection-name` 和 `--provider-type`。在此範例中，第三方供應商名稱為 `GitHub`，而指定的連線名稱為 `MyConnection`。

```
aws codestar-connections create-connection --provider-type GitHub --connection-name
 MyConnection
```

如果成功，此命令會傳回類似下列內容的連線 ARN 資訊。

```
{
    "ConnectionArn": "arn:aws:codestar-connections:us-west-2:account_id:connection/
aEXAMPLE-8aad-4d5d-8878-dfcab0bc441f"
}
```

2. 使用主控台完成連線。

將第 1 GitHub 版來源動作的輪詢管道遷移至 Webhook

您可以遷移管道以使用 webhook 來偵測 GitHub 來源儲存庫中的變更。此遷移到網絡掛鉤僅適用於 GitHub 版本 1 的操作。

- 控制台：將輪詢管道遷移到 Webhook（GitHub 版本 1 源操作）（控制台）
- CLI：將輪詢管線移轉至 Webhook (第 1 GitHub 版來源動作) (CLI)
- AWS CloudFormation: 更新推送事件的管道 (第 1 GitHub 版來源動作) (AWS CloudFormation範本)

將輪詢管道遷移到 Webhook（GitHub 版本 1 源操作）（控制台）

您可以使用主 CodePipeline 控台來更新管道，以使用 webhook 偵測 CodeCommit 來源儲存庫中的變更。

請遵循下列步驟來編輯正在使用輪詢 (定期檢查) 來 EventBridge 改用的管線。如果您想要建立管道，請參閱在中建立管線 CodePipeline。

使用主控台時，會為您變更管道的 `PollForSourceChanges` 參數。 GitHub 網絡掛鉤已為您創建並註冊。

編輯管道來源階段

1. 請登入AWS Management Console並開啟 CodePipeline 主控台，網址為 http://console.aws.amazon.com/codesuite/codepipeline/home。

   所有與您的 AWS 帳戶相關的管道名稱都會顯示。

2. 在 Name (名稱) 中，選擇您想編輯的管道名稱。這會開啟管道的詳細檢視，包含管道中各階段的每項動作之每項狀態。
3. 在管道詳細資訊頁面上，選擇 Edit (編輯)。
4. 在 Edit (編輯) 階段，選擇來源動作上的編輯圖示。
5. 展開變更偵測選項，然後選擇使用 Amazon E CloudWatch vents 以在發生變更時自動啟動我的管道 (建議使用)。

   將顯示一條消息，建議您在中 CodePipeline 創建一個 webhook GitHub 以檢測源更改：AWS CodePipeline 將為您創建一個 webhook。您可以在以下選項中選擇退出。選擇 Update (更新)。除了網絡掛鉤之外，還 CodePipeline 創建以下內容：

   - 一個密碼，隨機生成並用於授權連接 GitHub。
   - Webhook URL，這是使用區域的公有端點所產生。

   CodePipeline 使 GitHub用註冊網絡掛鉤。這將訂閱 URL 以接收儲存庫事件。
6. 當您完成管道編輯後，請選擇 Save pipeline changes (儲存管道變更) 以返回摘要頁面。

   訊息顯示要為管道建立的 Webhook 名稱。選擇 Save and continue (儲存並繼續)。
7. 若要測試您的動作，請使用 AWS CLI 發佈變更，將變更遞交至管道的來源階段中指定的來源。

將輪詢管線移轉至 Webhook (第 1 GitHub 版來源動作) (CLI)

請遵循下列步驟來編輯正在使用定期檢查的管道，以改用 Webhook。如果您想要建立管道，請參閱在中建立管線 CodePipeline。

若要建置事件驅動型管道，您可以編輯管道的 `PollForSourceChanges` 參數，然後手動建立以下資源：

- GitHub 網絡掛鉤和授權參數

建立和註冊您的 Webhook

> **Note**
>
> 當您使用 CLI 或 AWS CloudFormation 建立管道並新增 Webhook 時，您必須停用定期檢查。若要停用定期檢查，您必須明確新增 `PollForSourceChanges` 參數，並將其設為 false，如以下最終程序中所詳述。否則，CLI 或 AWS CloudFormation 管道的預設

值是將 PollForSourceChanges 預設為 true，並且不會顯示在管道結構輸出中。如需 PollForSourceChanges 預設值的更多資訊，請參閱 PollForSourceChanges 參數的預設設定。

1. 在文字編輯器中，為您想建立的 webhook 建立並儲存一個 JSON 檔案。為名為 my-webhook的 Webhook 使用此範例檔案：

```
{
    "webhook": {
        "name": "my-webhook",
  "targetPipeline": "pipeline_name",
  "targetAction": "source_action_name",
  "filters": [{
      "jsonPath": "$.ref",
      "matchEquals": "refs/heads/{Branch}"
  }],
  "authentication": "GITHUB_HMAC",
  "authenticationConfiguration": {
      "SecretToken": "secret"
  }
    }
}
```

2. 呼叫 put-webhook 命令，並包含 --cli-input 和 --region 參數。

   以下命令範例會利用 webhook_json JSON 檔案建立一個 Webhook。

```
aws codepipeline put-webhook --cli-input-json file://webhook_json.json --region
 "eu-central-1"
```

3. 在此範例顯示的輸出中，會針對名為 my-webhook 的 Webhook 傳回 URL 與 ARN。

```
{
    "webhook": {
        "url": "https://webhooks.domain.com/
trigger111111111EXAMPLE11111111111111111",
        "definition": {
            "authenticationConfiguration": {
                "SecretToken": "secret"
            },
            "name": "my-webhook",
            "authentication": "GITHUB_HMAC",
```

```
            "targetPipeline": "pipeline_name",
            "targetAction": "Source",
            "filters": [
                {
                    "jsonPath": "$.ref",
                    "matchEquals": "refs/heads/{Branch}"
                }
            ]
        },
        "arn": "arn:aws:codepipeline:eu-central-1:ACCOUNT_ID:webhook:my-webhook"
    },
    "tags": [{
      "key": "Project",
      "value": "ProjectA"
    }]
}
```

此範例透過在 Webhook 上包含 `Project` 標籤索引鍵和 `ProjectA` 值，將標籤新增到 Webhook。如需有關在中標記資源的更多資訊 CodePipeline，請參閱標記 資源。

4. 呼叫 register-webhook-with-third-party 命令並加入 `--webhook-name` 參數。

   以下範例命令會註冊名為 `my-webhook` 的 Webhook。

```
aws codepipeline register-webhook-with-third-party --webhook-name my-webhook
```

若要編輯管線的 PollForSourceChanges參數

> **Important**
>
> 當您使用這個方法建立管道時，如果沒有明確設為 false，則 `PollForSourceChanges` 參數會預設為 true。當新增基於事件的變更偵測時，您必須將該參數新增到輸出，並將其設為 false 以停用輪詢。否則，您的管道會針對單一來源變更啟動兩次。如需詳細資訊，請參閱 PollForSourceChanges 參數的預設設定。

1. 執行 get-pipeline 命令，將管道結構複製到 JSON 檔案。例如，針對名為 `MyFirstPipeline` 的管道，您會輸入下列命令：

```
aws codepipeline get-pipeline --name MyFirstPipeline >pipeline.json
```

此命令不會傳回任何內容，但您建立的檔案應該會顯示在您執行命令的目錄中。

2. 在任何純文字編輯器中開啟 JSON 檔案，然後變更或新增 PollForSourceChanges 參數，來編輯來源階段。在這個範例中，對於名為 UserGitHubRepo 的儲存庫，該參數會設為 false。

   我為什麼要進行這項變更？ 變更此參數會關閉定期檢查，以便您只能使用以事件為基礎的變更偵測。

```
"configuration": {
    "Owner": "name",
    "Repo": "UserGitHubRepo",
    "PollForSourceChanges": "false",
    "Branch": "main",
    "OAuthToken": "****"
},
```

3. 如果您使用的是使用 get-pipeline 命令擷取的管道結構，則必須從檔案移除 metadata 行，以編輯 JSON 檔案中的結構。否則，update-pipeline 命令無法使用它。從 JSON 檔案中的管道結構移除 "metadata" 區段，包含 { }，以及 "created"、"pipelineARN" 和 "updated" 欄位。

   例如，從結構中移除下列幾行：

```
"metadata": {
    "pipelineArn": "arn:aws:codepipeline:region:account-ID:pipeline-name",
    "created": "date",
    "updated": "date"
},
```

   儲存檔案。

4. 若要套用您的變更，請執行 update-pipeline 命令，並指定管道 JSON 檔案，與下面類似：

> **Important**
>
> 請確認在檔案名稱之前包含 file://。這是此命令必要項目。

```
aws codepipeline update-pipeline --cli-input-json file://pipeline.json
```

   此命令會傳回所編輯管道的整個結構。

> **Note**
>
> update-pipeline 命令將終止管道。若在您執行 update-pipeline 命令時有修訂正在透過管道執行，該執行將停止。您必須手動啟動管道，以透過更新的管道執行該修訂。使用 start-pipeline-execution 命令來手動啟動您的管道。

更新推送事件的管道 (第 1 GitHub 版來源動作) (AWS CloudFormation範本)

請按照以下步驟使用 webhook 將您的管道（帶有 GitHub 源）從定期檢查（輪詢）更新為基於事件的更改檢測。

要使用構建事件驅動的管道AWS CodeCommit，您可以編輯管道的`PollForSourceChanges`參數，然後將 GitHub webhook 資源添加到模板中。

如果您使用 AWS CloudFormation 來建立和管理您的管道，則您的範本具有如下的內容。

> **Note**
>
> 請注意，來源階段中的 `PollForSourceChanges` 組態屬性。如果您的範本未包含該屬性，則 `PollForSourceChanges` 會預設為 `true`。

YAML

```
Resources:
  AppPipeline:
    Type: AWS::CodePipeline::Pipeline
    Properties:
      Name: github-polling-pipeline
      RoleArn:
        !GetAtt CodePipelineServiceRole.Arn
      Stages:
        -
          Name: Source
          Actions:
            -
              Name: SourceAction
              ActionTypeId:
                Category: Source
```

```
                Owner: ThirdParty
                Version: 1
                Provider: GitHub
              OutputArtifacts:
                - Name: SourceOutput
              Configuration:
                Owner: !Ref GitHubOwner
                Repo: !Ref RepositoryName
                Branch: !Ref BranchName
                OAuthToken:
 {{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}
 PollForSourceChanges: true
              RunOrder: 1


...
```

JSON

```
        "AppPipeline": {
            "Type": "AWS::CodePipeline::Pipeline",
            "Properties": {
                "Name": "github-polling-pipeline",
                "RoleArn": {
                    "Fn::GetAtt": [
                        "CodePipelineServiceRole",
                        "Arn"
                    ]
                },
                "Stages": [
                    {
                        "Name": "Source",
                        "Actions": [
                            {
                                "Name": "SourceAction",
                                "ActionTypeId": {
                                    "Category": "Source",
                                    "Owner": "ThirdParty",
                                    "Version": 1,
                                    "Provider": "GitHub"
                                },
                                "OutputArtifacts": [
                                    {
```

```
                                    "Name": "SourceOutput"
                                }
                            ],
                            "Configuration": {
                                "Owner": {
                                    "Ref": "GitHubOwner"
                                },
                                "Repo": {
                                    "Ref": "RepositoryName"
                                },
                                "Branch": {
                                    "Ref": "BranchName"
                                },
                                "OAuthToken":
 "{{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}",
                                "PollForSourceChanges": true
                            },
                            "RunOrder": 1
                        }
                    ]
                },


...
```

在您的範本中新增參數並建立 Webhook

我們強烈建議您使用 AWS Secrets Manager 來儲存您的憑證。如果您使用 Secrets Manager，則您必須已在 Secrets Manager 中設定並存放秘密參數。這個範例會使用 Secrets Manager 的動態參考來取得網路掛接的 GitHub認證。如需詳細資訊，請參閱使用動態參考來指定範本值。

> **Important**
>
> 傳遞機密參數時，請勿在範本中直接輸入值。此值渲染為純文字，因此為可讀取。基於安全因素，請勿在您的 AWS CloudFormation 範本中使用純文字儲存您的憑證。

當您使用 CLI 或 AWS CloudFormation 建立管道並新增 Webhook 時，您必須停用定期檢查。

> **Note**
>
> 若要停用定期檢查，您必須明確新增 PollForSourceChanges 參數，並將其設為 false，如以下最終程序中所詳述。否則，CLI 或 AWS CloudFormation 管道的預設值是將 PollForSourceChanges 預設為 true，並且不會顯示在管道結構輸出中。如需 PollForSourceChanges 預設值的更多資訊，請參閱 PollForSourceChanges 參數的預設設定。

1. 在範本的 Resources 下，新增您的參數：

   YAML

   ```
   Parameters:
           GitHubOwner:
             Type: String

   ...
   ```

   JSON

   ```
   {
    "Parameters": {
     "BranchName": {
      "Description": "GitHub branch name",
      "Type": "String",
      "Default": "main"
     },
     "GitHubOwner": {
      "Type": "String"
     },

   ...
   ```

2. 使用 AWS::CodePipeline::Webhook AWS CloudFormation 資源來新增 Webhook。

   > **Note**
   >
   > 您指定的 TargetAction 必須符合管道中定義之來源動作的 Name 屬性。

如果設定RegisterWithThirdParty為true，請確定與之相關聯的使用者OAuthToken可以在中設定所需的範圍 GitHub。令牌和網絡掛鉤需要以下 GitHub 範圍：

- repo - 用於完全控制將成品從公有和私有儲存庫讀取和提取至管道。
- admin:repo_hook - 用於完全控制儲存庫勾點。

否則， GitHub 返回 404。如需有關傳回之 404 的詳細資訊，請參閱https://help.github.com/articles/about-webhooks。

YAML

```
 AppPipelineWebhook:
   Type: AWS::CodePipeline::Webhook
   Properties:
     Authentication: GITHUB_HMAC
     AuthenticationConfiguration:
       SecretToken:
{{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}
     Filters:
       -
         JsonPath: "$.ref"
         MatchEquals: refs/heads/{Branch}
     TargetPipeline: !Ref AppPipeline
     TargetAction: SourceAction
     Name: AppPipelineWebhook
     TargetPipelineVersion: !GetAtt AppPipeline.Version
     RegisterWithThirdParty: true


...
```

JSON

```
"AppPipelineWebhook": {
    "Type": "AWS::CodePipeline::Webhook",
    "Properties": {
        "Authentication": "GITHUB_HMAC",
        "AuthenticationConfiguration": {
            "SecretToken":
 "{{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}"
```

```
        },
        "Filters": [{
            "JsonPath": "$.ref",
            "MatchEquals": "refs/heads/{Branch}"
        }],
        "TargetPipeline": {
            "Ref": "AppPipeline"
        },
        "TargetAction": "SourceAction",
        "Name": "AppPipelineWebhook",
        "TargetPipelineVersion": {
            "Fn::GetAtt": [
              "AppPipeline",
              "Version"
            ]
        },
        "RegisterWithThirdParty": true
    }
},

...
```

3. 將更新的範本儲存到本機電腦，然後開啟 AWS CloudFormation 主控台。
4. 選擇您的堆疊，然後選擇 Create Change Set for Current Stack (建立目前堆疊的變更集)。
5. 上傳範本，然後檢視 AWS CloudFormation 中所列的變更。這些是會針對堆疊進行的變更。您應該會在清單中看到新資源。
6. 選擇 Execute (執行)。

若要編輯管線的 PollForSourceChanges參數

> **Important**
>
> 當您使用這個方法建立管道時，如果沒有明確設為 false，則 `PollForSourceChanges` 參數會預設為 true。當新增基於事件的變更偵測時，您必須將該參數新增到輸出，並將其設為 false 以停用輪詢。否則，您的管道會針對單一來源變更啟動兩次。如需詳細資訊，請參閱 PollForSourceChanges 參數的預設設定。

- 在範本中，將 `PollForSourceChanges` 變更為 `false`。如果您並未在管道定義中包含 `PollForSourceChanges`，請新增它，並將其設為 false。

我為什麼要進行這項變更？ 將此參數變更為 `false` 會關閉定期檢查，因此您只能使用事件型變更偵測。

YAML

```
          Name: Source
          Actions:
            -
              Name: SourceAction
              ActionTypeId:
                Category: Source
                Owner: ThirdParty
                Version: 1
                Provider: GitHub
              OutputArtifacts:
                - Name: SourceOutput
              Configuration:
                Owner: !Ref GitHubOwner
                Repo: !Ref RepositoryName
                Branch: !Ref BranchName
                OAuthToken:
{{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}
                PollForSourceChanges: false
              RunOrder: 1
```

JSON

```
{
   "Name": "Source",
   "Actions": [{
 "Name": "SourceAction",
 "ActionTypeId": {
     "Category": "Source",
     "Owner": "ThirdParty",
     "Version": 1,
     "Provider": "GitHub"
},
"OutputArtifacts": [{
    "Name": "SourceOutput"
}],
"Configuration": {
    "Owner": {
```

```
   "Ref": "GitHubOwner"
     },
     "Repo": {
   "Ref": "RepositoryName"
     },
     "Branch": {
         "Ref": "BranchName"
     },
     "OAuthToken":
 "{{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}",
     PollForSourceChanges: false
 },
 "RunOrder": 1
    }]
```

Example

當您使用建立這些資源時AWS CloudFormation，會在指定的 GitHub 儲存庫中建立定義的 webhook。遞交時會觸發您的管道。

YAML

```
Parameters:
  GitHubOwner:
    Type: String

Resources:
  AppPipelineWebhook:
    Type: AWS::CodePipeline::Webhook
    Properties:
      Authentication: GITHUB_HMAC
      AuthenticationConfiguration:
        SecretToken: {{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}
      Filters:
        -
          JsonPath: "$.ref"
          MatchEquals: refs/heads/{Branch}
      TargetPipeline: !Ref AppPipeline
      TargetAction: SourceAction
      Name: AppPipelineWebhook
      TargetPipelineVersion: !GetAtt AppPipeline.Version
      RegisterWithThirdParty: true
```

```
  AppPipeline:
    Type: AWS::CodePipeline::Pipeline
    Properties:
      Name: github-events-pipeline
      RoleArn:
        !GetAtt CodePipelineServiceRole.Arn
      Stages:
        -
          Name: Source
          Actions:
            -
              Name: SourceAction
              ActionTypeId:
                Category: Source
                Owner: ThirdParty
                Version: 1
                Provider: GitHub
              OutputArtifacts:
                - Name: SourceOutput
              Configuration:
                Owner: !Ref GitHubOwner
                Repo: !Ref RepositoryName
                Branch: !Ref BranchName
                OAuthToken:
 {{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}
                PollForSourceChanges: false
              RunOrder: 1

...
```

JSON

```
{
    "Parameters": {
        "BranchName": {
            "Description": "GitHub branch name",
            "Type": "String",
            "Default": "main"
        },
        "RepositoryName": {
            "Description": "GitHub repository name",
            "Type": "String",
            "Default": "test"
```

```
        },
        "GitHubOwner": {
            "Type": "String"
        },
        "ApplicationName": {
            "Description": "CodeDeploy application name",
            "Type": "String",
            "Default": "DemoApplication"
        },
        "BetaFleet": {
            "Description": "Fleet configured in CodeDeploy",
            "Type": "String",
            "Default": "DemoFleet"
        }
    },
    "Resources": {

...

        },
        "AppPipelineWebhook": {
            "Type": "AWS::CodePipeline::Webhook",
            "Properties": {
                "Authentication": "GITHUB_HMAC",
                "AuthenticationConfiguration": {
                    "SecretToken": {

 "{{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}"
                    }
                },
                "Filters": [
                    {
                        "JsonPath": "$.ref",
                        "MatchEquals": "refs/heads/{Branch}"
                    }
                ],
                "TargetPipeline": {
                    "Ref": "AppPipeline"
                },
                "TargetAction": "SourceAction",
                "Name": "AppPipelineWebhook",
                "TargetPipelineVersion": {
                    "Fn::GetAtt": [
                        "AppPipeline",
```

```
                    "Version"
                ]
            },
            "RegisterWithThirdParty": true
        }
    },
    "AppPipeline": {
        "Type": "AWS::CodePipeline::Pipeline",
        "Properties": {
            "Name": "github-events-pipeline",
            "RoleArn": {
                "Fn::GetAtt": [
                    "CodePipelineServiceRole",
                    "Arn"
                ]
            },
            "Stages": [
                {
                    "Name": "Source",
                    "Actions": [
                        {
                            "Name": "SourceAction",
                            "ActionTypeId": {
                                "Category": "Source",
                                "Owner": "ThirdParty",
                                "Version": 1,
                                "Provider": "GitHub"
                            },
                            "OutputArtifacts": [
                                {
                                    "Name": "SourceOutput"
                                }
                            ],
                            "Configuration": {
                                "Owner": {
                                    "Ref": "GitHubOwner"
                                },
                                "Repo": {
                                    "Ref": "RepositoryName"
                                },
                                "Branch": {
                                    "Ref": "BranchName"
                                },
```

```
                                    "OAuthToken":
 "{{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}",
                                    "PollForSourceChanges": false
                                },
                                "RunOrder": 1

...
```

建立 CodePipeline 服務角色

當您建立管道時，須建立服務角色或使用現有的服務角色。

您可以使用CodePipeline主控台或AWS CLI建立CodePipeline服務角色。建立管道需要用到服務角色，而且管道一律會關聯到該服務角色。

使用 AWS CLI 建立管線之前，必須為管線建立CodePipeline服務角色。如需具有指定服務角色和原則的AWS CloudFormation範例範本，請參閱中的教學課程教學：建立透過 AWS CloudFormation 部署動作使用變數的管道。

服務角色不是AWS受管理的角色，但最初是為了建立管線而建立的，然後在將新權限新增至服務角色原則時，您可能需要更新管線的服務角色。一旦使用服務角色建立管道，便無法套用不同的服務角色到該管道。將建議的政策連接至服務角色。

如需服務角色的詳細資訊，請參閱管理服 CodePipeline 務角色。

建立 CodePipeline 服務角色 (主控台)

使用主控台建立管線時，您可以使用管線建立精靈建立CodePipeline服務角色。

1. 請登入AWS Management Console並開啟CodePipeline主控台，網址為 http://console.aws.amazon.com/codesuite/codepipeline/home。

   選擇 Create pipeline (建立管道) 並完成管道建立精靈中的 Step 1: Choose pipeline settings (步驟 1：選擇管道設定) 頁面。

   > **Note**
   >
   > 在您建立管道後，便無法更改其名稱。如需其他限制的相關資訊，請參閱 AWS CodePipeline 中的配額。

2. 在 [服務角色] 中，選擇 [新增服務角色] CodePipeline 以允許在 IAM 中建立新的服務角色。
3. 完成管道建立。您可以在 IAM 角色清單中檢視管線服務角色，您可以透過 AWS CLI 執行`get-pipeline`命令來檢視與管道相關聯的服務角色 ARN。

建立 CodePipeline 服務角色 (CLI)

在使用 AWS CLI 建立管線之前AWS CloudFormation，或者必須為管線建立CodePipeline服務角色，並附加服務角色原則和信任原則。若要使用 CLI 建立您的服務角色，請使用下列步驟先建立信任原則 JSON 和角色原則 JSON，做為要執行 CLI 命令的目錄中的個別檔案。

> **Note**
>
> 建議您只允許系統管理使用者建立任何服務角色。具有建立角色和連接任何政策之許可的人員可以提升自己的許可。反之，建立一個政策，讓他們只建立所需的角色，或讓系統管理員代表他們建立服務角色。

1. 在終端機視窗中，輸入下列命令以建立名為的檔案`TrustPolicy.json`，您將在其中貼上角色原則 JSON。此範例使用 VIM。

```
vim TrustPolicy.json
```

2. 將下列 JSON 貼入檔案中。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "codepipeline.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

要保存並退出文件，請輸入以下 VIM 命令：

```
:wq
```

3. 在終端機視窗中，輸入下列命令以建立名為的檔案RolePolicy.json，您將在其中貼上角色原則 JSON。此範例使用 VIM。

```
vim RolePolicy.json
```

4. 將下列 JSON 貼入檔案中。在政策聲明Resource欄位中新增管道的 Amazon 資源名稱 (ARN)，確保盡可能縮減許可範圍。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codecommit:CancelUploadArchive",
        "codecommit:GetBranch",
        "codecommit:GetCommit",
        "codecommit:GetUploadArchiveStatus",
        "codecommit:UploadArchive"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "codedeploy:CreateDeployment",
        "codedeploy:GetApplicationRevision",
        "codedeploy:GetDeployment",
        "codedeploy:GetDeploymentConfig",
        "codedeploy:RegisterApplicationRevision"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "codebuild:BatchGetBuilds",
        "codebuild:StartBuild"
      ],
      "Resource": "*"
```

```
    },
    {
      "Effect": "Allow",
      "Action": [
        "devicefarm:ListProjects",
        "devicefarm:ListDevicePools",
        "devicefarm:GetRun",
        "devicefarm:GetUpload",
        "devicefarm:CreateUpload",
        "devicefarm:ScheduleRun"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "lambda:InvokeFunction",
        "lambda:ListFunctions"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "iam:PassRole"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "elasticbeanstalk:*",
        "ec2:*",
        "elasticloadbalancing:*",
        "autoscaling:*",
        "cloudwatch:*",
        "s3:*",
        "sns:*",
        "cloudformation:*",
        "rds:*",
        "sqs:*",
        "ecs:*"
      ],
      "Resource": "resource_ARN"
```

```
    }
  ]
}
```

要保存並退出文件，請輸入以下 VIM 命令：

```
:wq
```

5. 輸入下列命令以建立角色，並連接信任角色政策。政策名稱格式通常與角色名稱格式相同。此範例使用角色名稱MyRole和建立TrustPolicy為個別檔案的策略。

```
aws iam create-role --role-name MyRole --assume-role-policy-document file://
TrustPolicy.json
```

6. 輸入下列命令以建立政策，並將它連接至角色。政策名稱格式通常與角色名稱格式相同。此範例使用角色名稱MyRole和建立MyRole為個別檔案的策略。

```
aws iam put-role-policy --role-name MyRole --policy-name RolePolicy --policy-
document file://RolePolicy.json
```

7. 若要檢視建立的角色名稱和信任原則，請為名稱的角色輸入下列命令MyRole：

```
aws iam get-role --role-name MyRole
```

8. 使用 AWS CLI 或AWS CloudFormation建立管線時，請使用服務角色 ARN。

在 CodePipeline 中標記管道

標籤是與 AWS 資源關聯的金鑰值對。您可以在 CodePipeline 中將標籤套用到您的管道。如需 CodePipeline 資源標記、使用案例、標籤索引鍵和值限制條件的相關資訊，以及支援的資源類型，請參標記 資源。

建立管道時，您可以使用 CLI 指定標籤。您可以使用主控台或 CLI 來新增或移除標籤，並更新管道中標籤的值。您可以對新增最多 50 個標籤到各管道。

主題

- 標記管道 (主控台)
- 標記管道 (CLI)

標記管道 (主控台)

您可以使用主控台或 CLI 以標記資源。管道是唯一可用控制台或 CLI 管理的 CodePipeline 資源。

主題

新增標籤到管道 (主控台)

您可以使用主控台，將標籤新增到現有管道。

1. 登入AWS Management Console，然後 CodePipeline 往http://console.aws.amazon.com/codesuite/codepipeline/home。
2. 在 Pipelines (管道) 頁面，選擇您要新增標籤的管道。
3. 從導覽窗格中，選擇 Settings (設定)。
4. 在 Pipeline tags (管道標籤) 中，選擇 Edit (編輯)。
5. 在 Key (索引鍵) 和ValueValue (值) 欄中，在你想新增的各組標籤中輸入金鑰對。(Value (值) 欄為選用。) 例如，在 Key (索引鍵) 中輸入 **`Project`**。在 Value (值) 中輸入 **`ProjectA`**。
6. (選用) 選擇 Add tag (新增標籤)，新增更多列，然後輸入更多標籤。
7. 選擇 Submit (提交)。標籤列在管道設定之下。

檢視管道標籤 (主控台)

您可以使用主控台列出現有管道的標籤。

1. 登入AWS Management Console，然後 CodePipeline 往http://console.aws.amazon.com/codesuite/codepipeline/home。
2. 在 Pipelines (管道) 頁面，選擇您要檢視標籤的管道。
3. 從導覽窗格中，選擇 Settings (設定)。
4. 在 Pipeline tags (管道標籤)下，檢視 Key (金鑰) 和 Value (值) 欄下的管道標籤。

編輯管道標籤 (主控台)

您可以使用主控台來編輯已新增到管道的標籤。

1. 登入AWS Management Console，然後 CodePipeline 往http://console.aws.amazon.com/codesuite/codepipeline/home。
2. 在 Pipelines (管道) 頁面，選擇您想更新標籤的管道。
3. 從導覽窗格中，選擇 Settings (設定)。
4. 在 Pipeline tags (管道標籤) 中，選擇 Edit (編輯)。
5. 在 Key (金鑰) 和 Value (加值) 欄，視需要更新每個欄位的值。例如，針對 **`Project`** 索引鍵，在 Value (值) 中將 **`ProjectA`** 變為 **`ProjectB`**。
6. 選擇 Submit (提交)。

從管道移除標籤 (主控台)

您可以使用主控台，從管道刪除標籤。當您從關聯的資源移除標籤時，將會刪除這些標籤。

1. 登入AWS Management Console，然後 CodePipeline 往http://console.aws.amazon.com/codesuite/codepipeline/home。
2. 在 管道 (Pipelines) 頁面，選擇您要移除標籤的管道。
3. 從導覽窗格中，選擇 Settings (設定)。
4. 在 Pipeline tags (管道標籤) 中，選擇 Edit (編輯)。
5. 在您要刪除的金鑰和值的每個標籤旁邊，選擇 移除標籤 (Remove tag)。
6. 選擇 Submit (提交)。

標記管道 (CLI)

您可以使用 CLI 標籤資源。您必須使用主控台，管理管道中的標籤。

主題

- 新增標籤到管道 (CLI)
- 檢視管道標籤 (CLI)
- 編輯管道標籤 (CLI)

- 從管道移除標籤 (CLI)

新增標籤到管道 (CLI)

您可以使用主控台或 AWS CLI 標記管道。

若要在建立時將標籤新增到管道，請參閱 在中建立管線 CodePipeline。

在這些步驟中，我們假設您已經安裝新版 AWS CLI 或更新到最新版本。如需詳細資訊，請參閱安裝 AWS Command Line Interface。

在終端機或命令列，執行 tag-resource 命令，指定您要新增的管道 Amazon Resource Name (ARN)，和您想新增標籤的金鑰和值。您可以新增多個標籤到管道。例如，若要標記名為*MyPipeline*具有兩個標籤，一個名為*DeploymentEnvironment*，標籤值為*##*，以及一個名為*IscontainerBased*，標籤值為*##*：

```
aws codepipeline tag-resource --resource-arn arn:aws:codepipeline:us-west-2:account-id:MyPipeline --tags key=Project,value=ProjectA key=IscontainerBased,value=true
```

如果成功，此命令不會傳回任何內容。

檢視管道標籤 (CLI)

請遵循下列步驟，使用 AWS CLI 檢視管道的 AWS 標籤。如果沒有新增標籤，將會傳回空的清單。

在終端機或命令列上執行 list-tags-for-resource 命令。例如，若要檢視名為的管道的標籤索引鍵和標籤值列表*MyPipeline*使用arn:aws:codepipeline:*us-west-2*:*account-id*:*MyPipeline*ARN 值：

```
aws codepipeline list-tags-for-resource --resource-arn arn:aws:codepipeline:us-west-2:account-id:MyPipeline
```

如果成功，此命令會傳回類似如下的資訊：

```
{
    "tags": {
        "Project": "ProjectA",
        "IscontainerBased": "true"
    }
```

```
}
```

編輯管道標籤 (CLI)

請遵循下列步驟，使用 AWS CLI 編輯管道標籤。您可以變更現有索引鍵的值或新增其他索引鍵。您也可以從管道移除標籤，如以下部分所示。

在終端機或命令列，執行 tag-resource 命令，指定您要更新標籤的管道 ARN，並指定標籤金鑰和標籤值：

```
aws codepipeline tag-resource --resource-arn arn:aws:codepipeline:us-west-2:account-id:MyPipeline --tags key=Project,value=ProjectA
```

如果成功，此命令不會傳回任何內容。

從管道移除標籤 (CLI)

依照以下步驟使用 AWS CLI 從管道移除標籤。當您從關聯的資源移除標籤時，將會刪除這些標籤。

> **Note**
>
> 如果您刪除管道，所有標籤關聯會從已刪除的管道中移除。刪除刪除管道後就不需要移除標籤了。

在終端機或命令列，執行 untag-resource 命令，指定您想移除標籤的管道 ARN，和您想移除的標籤的標籤金鑰。例如，若要移除名為*MyPipeline*標籤鍵*##*和*IscontainerBased*：

```
aws codepipeline untag-resource --resource-arn arn:aws:codepipeline:us-west-2:account-id:MyPipeline --tag-keys Project IscontainerBased
```

如果成功，此命令不會傳回任何內容。若要驗證與管道相關的標籤，請執行 list-tags-for-resource 命令。

建立通知規則

您可以使用通知規則向使用者通知有重要的變更，例如管道開始執行時。通知規則指定用於傳送通知的事件和 Amazon SNS 主題。如需詳細資訊，請參閱什麼是通知？

您可以使用主控台或 AWS CLI 為 AWS CodePipeline 建立通知規則。

建立通知規則 (主控台)

1. 請登入AWS Management Console並開啟 CodePipeline 主控台，網址為 https://console.aws.amazon.com/codepipeline/。
2. 選擇 Pipelines (管道)，然後選擇您要新增通知的管道。
3. 在管道頁面上，選擇 Notify (通知)，然後選擇 Create notification rule (建立通知規則)。您也可以移至管道的 Settings (設定) 頁面，然後選擇 Create notification rule (建立通知規則)。
4. 在 Notification name (通知名稱) 中，輸入規則的名稱。
5. 如果您只想要提供給 Amazon 的資訊 EventBridge 包含在通知中，請在 [詳細資料類型] 中選擇 [基本]。如果您想要包含提供給 Amazon 的資訊以 EventBridge 及可能由 CodePipeline 或通知管理員提供的資訊，請選擇「完整」。

   如需詳細資訊，請參閱了解通知內容與安全性。
6. 在 Events that trigger notifications (觸發通知的事件) 中，選取您要傳送通知的事件。如需詳細資訊，請參閱管道上通知規則的事件。
7. 在 Targets (目標) 中，執行下列其中一個動作：

   - 如果您已設定要與通知搭配使用的資源，請在 Choose target type (選擇目標類型) 中，選擇 AWS Chatbot (Slack) 或 SNS topic (SNS 主題)。在選擇目標中，選擇 Amazon SNS 主題的用戶端名稱 (針對在中設定的 Slack 用戶端AWS Chatbot) 或 Amazon 資源名稱 (ARN) (適用於已設定通知所需政策的 Amazon SNS 主題)。
   - 如果您尚未設定要與通知搭配使用的資源，請選擇 Create target (建立目標)，然後選擇 SNS topic (SNS 主題)。在 codestar-notifications- 之後，提供主題名稱，然後選擇 Create (建立)。

> **Note**
>
> - 如果您在建立通知規則的過程中建立 Amazon SNS 主題，將會為您套用允許通知功能將事件發佈至主題的政策。使用針對通知規則建立的主題，有助於確保您只訂閱需要接收此資源相關通知的使用者。
> - 您無法在建立通知規則時建立 AWS Chatbot 用戶端。如果您選擇 AWS Chatbot (Slack)，您會看到一個按鈕，指示您在 AWS Chatbot 中設定用戶端。選擇該選項會開啟 AWS Chatbot 主控台。如需詳細資訊，請參閱設定通知和 AWS Chatbot 之間的整合。

- 如果您想要使用現有的 Amazon SNS 主題作為目標，除了該主題可能存在的任何其他政策之外，還必須新增AWS CodeStar通知的必要政策。如需詳細資訊，請參閱為通知設定 Amazon SNS 主題和了解通知內容與安全性。

8. 若要完成建立規則，請選擇 Submit (提交)。
9. 您必須先訂閱使用者訂閱規則的 Amazon SNS 主題，才能收到通知。如需詳細資訊，請參閱訂閱使用者訂閱屬於目標的 Amazon SNS 主題。您也可以設定通知之間的整合，並AWS Chatbot將通知傳送到 Amazon Chime 聊天室或 Slack 頻道。如需詳細資訊，請參閱設定通知和 AWS Chatbot 之間的整合。

建立通知規則 (AWS CLI)

1. 在終端機或命令提示字元中，執行 create-notification rule 命令以產生 JSON 架構：

```
aws codestar-notifications create-notification-rule --generate-cli-skeleton
 > rule.json
```

您可以將檔案命名為任何您想要的名稱。在此範例中，檔案命名為 *rule.json*。

2. 在純文字編輯器中開啟 JSON 檔案，並編輯成包含您想要用於規則的資源、事件類型和目標。下列範例顯示了AWS針對識別碼**MyNotificationRule**為 *123* 456789012 的帳戶*MyDemoPipeline*中名為的管線命名的通知規則。管道執行開始MyNotificationTopic時，會將通知與完整詳細資料類型一起傳送至名為 *codestar-##*的 Amazon SNS 主題：

```
{
    "Name": "MyNotificationRule",
    "EventTypeIds": [
        "codepipeline-pipeline-pipeline-execution-started"
    ],
    "Resource": "arn:aws:codebuild:us-east-2:123456789012:MyDemoPipeline",
    "Targets": [
        {
            "TargetType": "SNS",
            "TargetAddress": "arn:aws:sns:us-east-2:123456789012:codestar-
notifications-MyNotificationTopic"
        }
    ],
    "Status": "ENABLED",
    "DetailType": "FULL"
```

```
}
```

儲存檔案。

3. 在終端機或命令列中，再次執行 create-notification-rule 命令，使用您剛編輯的檔案建立通知規則：

```
aws codestar-notifications create-notification-rule --cli-input-json
 file://rule.json
```

4. 如果成功，此命令會傳回通知規則的 ARN，如下所示：

```
{
    "Arn": "arn:aws:codestar-notifications:us-east-1:123456789012:notificationrule/
dc82df7a-EXAMPLE"
}
```

使用中的觸發器 CodePipeline

觸發器可讓您將管線設定為在特定事件類型或篩選的事件類型上啟動，例如偵測到特定分支或提取要求的變更時。可針對使用中 CodePipeline動作的連線 (例如 Bitbucket 和 GitLab.) 的來源`CodeStarSourceConnection`動作 GitHub設定觸發程序。

來源動作 (例如 CodeCommit 和 S3) 會使用變更偵測，如本節有關啟動管道的詳細說明。

您可以將觸發器新增至管道，並設定觸發器以篩選特定事件

您可以使用控制台或 CLI 指定觸發器。

篩選程式碼推送或提取要求的觸發程序

您可以設定管線觸發程序的篩選器，以針對不同的 Git 事件 (例如標籤或分支推送、特定檔案路徑中的變更、開啟到特定分支的提取要求等) 啟動管線執行。您可以使用AWS CodePipeline控制台或中的create-pipeline和update-pipeline命令AWS CLI來配置觸發器的篩選器。

您可以為下列觸發器類型指定篩選器：

- 推送

  當變更推送至來源儲存庫時，推送觸發程序會啟動管道。執行將使用您正在推送到的分支（即目標分支）的提交。您可以篩選分支、檔案路徑或 Git 標籤上的推送觸發程序。
- 拉取請求

  在來源儲存庫中開啟、更新或關閉提取要求時，提取要求觸發程序會啟動管道。執行將使用您從中提取的源分支（即源分支）中的提交。您可以在分支和文件路徑上過濾提取請求觸發器。

  > **Note**
  >
  > 提取要求的支援事件類型會開啟、更新或關閉 (合併)。所有其他提取請求事件都會被忽略。

管線定義可讓您在相同的推送觸發器組態中組合不同的篩選器。如需有關管線定義的詳細資訊，請參閱在管線 JSON (CLI) 中觸發篩選。有效的組合有：

- 標籤

- 分支機構
- 分支 + 文件路徑

您可以依下列方式指定篩選類型：

- 沒有篩選

  此觸發器組態會在任何推送至指定為動作組態一部分的預設分支時啟動您的管道。

- 指定篩選

  您可以添加一個過濾器，該過濾器可以在特定的過濾器上啟動管道，例如在代碼推送的分支名稱上，並獲取確切的提交。這也會將管線設定為不會在任何變更時自動啟動。

- 不偵測變更

  這不會新增觸發程序，且管線不會在任何變更時自動啟動。

下表提供每個事件類型的有效篩選選項。此表格也會顯示動作組態中自動變更偵測時，哪些觸發程序組態預設為 true 或 false。

觸發器配置	事件類型	篩選條件選項	偵測變更
新增觸發器 — 無篩選	無	無	true
添加觸發器 — 在代碼推送時進行過濾	推送事件	Git 標籤，分支，文件路徑	false
新增觸發程式 — 篩選提取要求	提取請求	分支, 檔案路徑	false
無觸發器 — 未偵測	無	無	false

> **Note**
>
> 此觸發類型使用自動變更偵測 (作為Webhook觸發類型)。使用此觸發器類型的來源動作提供者是針對程式碼推送 (Bitbucket Cloud、 GitHub 企業伺服器 GitHub、 GitLab .com 和 GitLab自我管理) 設定的連線。

對於過濾，支持 glob 格式的正則表達式模式，如中在語法中使用 glob 模式所述。

> **Note**
>
> 在某些情況下，對於具有在檔案路徑上篩選之觸發程序的管線，當第一次建立具有檔案路徑篩選器的分支時，管線可能不會啟動。如需詳細資訊，請參閱 具有使用依檔案路徑觸發程序篩選的連線的管線可能無法在分支建立時啟動。

主題

觸發器濾波器的考量

使用觸發程序時，必須考量下列事項。

- 對於具有分支和文件路徑過濾器的觸發器，第一次推送分支時，管道將不會運行，因為無法訪問為新創建的分支更改的文件列表。
- 合併提取請求可能會觸發兩個管道執行，在推送 (分支篩選器) 和提取要求 (分支篩選器) 觸發組態相交的情況下。

觸發器濾波器的範例

對於具有推送和提取要求事件類型之篩選器的 Git 組態，指定的篩選器可能會相互衝突。以下是推送與提取要求事件的有效篩選器組合範例。

當客戶在單一組態物件中結合篩選器時，這些篩選器會使用 AND 作業，這表示只有完全符合才會啟動管線。下面的例子顯示了 Git 的配置：

```
{
```

```
  "filePaths": {
    "includes": ["common/**/*.js"]
  },
  "branches": {
    "includes": ["feature/**"]
  }
}
```

透過上述 Git 設定，這個範例會顯示一個事件，因為 AND 作業成功而啟動管線執行。

```
{
  "ref": "refs/heads/feature/triggers",
  ...
  "commits": [
    {
      ...
      "modified": [
        "common/app.js"
      ]
      ...
    }
  ]
}
```

此範例顯示不會啟動管線執行的事件，因為分支能夠篩選，但檔案路徑不是。

```
{
   "ref": "refs/heads/feature/triggers",
  ...
  "commits": [
    {
      ...
      "modified": [
        "src/Main.java"
      ]
      ...
    }
  ]
}
```

同時，推送陣列內的觸發器配置物件會使用 OR 作業。這可讓您配置多個觸發器，以啟動同一管線的執行。如需 JSON 結構中欄位定義的清單，請參閱在管線 JSON (CLI) 中觸發篩選。

篩選推送事件 (主控台)

您可以使用主控台為推送事件新增篩選器，並包含或排除分支或檔案路徑。

篩選推送事件 (主控台)

1. 請登入AWS Management Console並開啟 CodePipeline 主控台，網址為 http://console.aws.amazon.com/codesuite/codepipeline/home。

   所有與您的 AWS 帳戶相關的管道名稱與狀態都會顯示。
2. 在 Name (名稱) 中，選擇您想編輯的管道名稱。否則，請在管線建立精靈中使用這些步驟。
3. 在管道詳細資訊頁面上，選擇 Edit (編輯)。
4. 在 「編輯」 頁面上，選擇您要編輯的來源動作。選擇 [編輯觸發器] 選擇「指定篩選」。
5. 在 [事件類型] 中，從下列選項中選擇 [推送]。

   - 選擇推送以在將變更推送至來源儲存區域時啟動管線。選擇此選項可讓欄位指定分支和檔案路徑或 Git 標籤的篩選條件。
   - 選擇「提取請求」以在來源儲存庫中開啟、更新或關閉提取要求時啟動管線。選擇此選項可讓欄位指定目標分支和檔案路徑的篩選條件。
6. 在篩選類型中，選擇下列其中一個選項。

   - 選擇「分支」以指定觸發程式監視的來源儲存庫中的分支，以便知道何時開始執行工作流程。在 In clu de 中，以 glob 格式輸入您要為觸發程序組態指定的分支名稱模式，以便在指定分支中的變更時啟動管線。在 Exclude 中，輸入 glob 格式的分支名稱正則表達式模式，您想要為觸發器配置指定要忽略，並且不要在指定分支中的更改時啟動管道。如需更多資訊，請參閱在語法中使用 glob 模式。

     > **Note**
     >
     > 如果包含和排除兩者具有相同的模式，則預設為排除該模式。

     您可以使用 glob 格式的正則表達式模式來定義分支名稱。例如，使用`main.*`來比對以開頭的所有分支`main.*`。如需更多資訊，請參閱在語法中使用 glob 模式。

     對於推送觸發器，請指定要推送到的分支，也就是目標分支。對於提取請求觸發程序，請指定要打開拉取請求的目的地分支。

- (選擇性) 在 [檔案路徑] 下，指定觸發器的檔案路徑。視需要在「包含」與「排除」中輸入名稱。

  您可以使用 glob 格式的正則表達式模式來定義文件路徑名。例如，使用`prod.*`來比對開頭為的所有檔案路徑`prod.*`。如需更多資訊，請參閱在語法中使用 glob 模式。

- 選擇「標籤」，將管線觸發程序組態設定為以 Git 標籤開始。在 In clu de 中，輸入 glob 格式的標籤名稱模式，您要為觸發程序組態指定這些模式，以便在發行指定的一或多個標籤時啟動管線。在排除中，輸入 glob 格式的標籤名稱正則表達式模式，您想要為觸發程序配置指定要忽略，並且不要在釋放指定的標籤或標籤時啟動管道。如果包含和排除兩者具有相同的標籤模式，則預設為排除標籤模式。

篩選提取要求 (主控台)

您可以使用控制台為具有指定事件的提取請求添加過濾器，並包含或排除分支或文件路徑。

篩選提取要求 (主控台)

1. 請登入AWS Management Console並開啟 CodePipeline 主控台，網址為 http://console.aws.amazon.com/codesuite/codepipeline/home。

   所有與您的 AWS 帳戶相關的管道名稱與狀態都會顯示。
2. 在 Name (名稱) 中，選擇您想編輯的管道名稱。否則，請在管線建立精靈中使用這些步驟。
3. 在管道詳細資訊頁面上，選擇 Edit (編輯)。
4. 在 「編輯」 頁面上，選擇您要編輯的來源動作。選擇 [編輯觸發器] 選擇「指定篩選」。
5. 在 [事件類型] 中，從下列選項中選擇 [提取要求]。

   - 選擇推送以在將變更推送至來源儲存區域時啟動管線。選擇此選項可讓欄位指定分支和檔案路徑或 Git 標籤的篩選條件。
   - 選擇 [提取要求]，以在開啟、更新或關閉指定目標分支的提取要求時啟動管線。選擇此選項可讓欄位指定分支和檔案路徑的篩選條件。

     您可以選擇性地指定下列提取要求事件進行篩選：

     - 已建立拉取請求
     - 新的修訂被提取請求
     - 拉取請求已關閉
6. 在篩選類型中，選擇下列其中一個選項。

- 選擇「分支」以指定觸發程式監視的來源儲存庫中的分支，以便知道何時開始執行工作流程。在 In clu de 中，以 glob 格式輸入您要為觸發程序組態指定的分支名稱模式，以便在指定分支中的變更時啟動管線。在 Exclude 中，輸入 glob 格式的分支名稱正則表達式模式，您想要為觸發器配置指定要忽略，並且不要在指定分支中的更改時啟動管道。如需更多資訊，請參閱在語法中使用 glob 模式。

  > **Note**
  >
  > 如果包含和排除兩者具有相同的模式，則預設為排除該模式。

  您可以使用 glob 格式的正則表達式模式來定義分支名稱。例如，使用`main.*`來比對以開頭的所有分支`main.*`。如需更多資訊，請參閱在語法中使用 glob 模式。

  對於推送觸發器，請指定要推送到的分支，也就是目標分支。對於提取請求觸發程序，請指定要打開拉取請求的目的地分支。

- (選擇性) 在 [檔案路徑] 下，指定觸發器的檔案路徑名稱。視需要在「包含」與「排除」中輸入名稱。

  您可以使用 glob 格式的正則表達式模式來定義文件路徑名。例如，使用`prod.*`來比對開頭為的所有檔案路徑`prod.*`。如需更多資訊，請參閱在語法中使用 glob 模式。

在管線 JSON (CLI) 中觸發篩選

您可以更新管線 JSON 以新增觸發器的篩選器。

若要使用AWS CLI建立或更新管線，請使用`create-pipeline`或`update-pipeline`指令。

下列範例 JSON 結構提供下欄位定義的參考`create-pipeline`。

```
{
    "pipeline": {
        "name": "MyServicePipeline",
        "triggers": [
            {
                "provider": "Connection",
                "gitConfiguration": {
                    "sourceActionName": "ApplicationSource",
                    "push": [
```

```
                    {
                        "filePaths": {
                            "includes": [
                                "projectA/**",
                                "common/**/*.js"
                            ],
                            "excludes": [
                                "**/README.md",
                                "**/LICENSE",
                                "**/CONTRIBUTING.md"
                            ]
                        },
                        "branches": {
                            "includes": [
                                "feature/**",
                                "release/**"
                            ],
                            "excludes": [
                                "mainline"
                            ]
                        },
                        "tags": {
                            "includes": [
                                "release-v0", "release-v1"
                            ],
                            "excludes": [
                                "release-v2"
                            ]
                        }
                    }
                ],
                "pullRequest": [
                    {
                        "events": [
                            "CLOSED"
                        ],
                        "branches": {
                            "includes": [
                                "feature/**",
                                "release/**"
                            ],
                            "excludes": [
                                "mainline"
                            ]
```

```
                            },
                            "filePaths": {
                                "includes": [
                                    "projectA/**",
                                    "common/**/*.js"
                                ],
                                "excludes": [
                                    "**/README.md",
                                    "**/LICENSE",
                                    "**/CONTRIBUTING.md"
                                ]
                            }
                        }
                    ]
                }
            }
        ],
        "stages": [
            {
                "name": "Source",
                "actions": [
                    {
                        "name": "ApplicationSource",
                        "configuration": {
                            "BranchName": "mainline",
                            "ConnectionArn": "arn:aws:codestar-connections:eu-
central-1:111122223333:connection/fe9ff2e8-ee25-40c9-829e-65f8EXAMPLE",
                            "FullRepositoryId": "monorepo-example",
                            "OutputArtifactFormat": "CODE_ZIP"
                        }
                    }
                ]
            }
        ]
    }
}
```

JSON 結構中的欄位定義如下：

- sourceActionName：具有 Git 配置的管道源操作的名稱。
- push：使用篩選推送事件。這些事件會在不同的推送篩選器和篩選器內的 AND 作業之間使用 OR 作業。

  - `branches`：要篩選的分支。分支在包含和排除之間使用 AND 運算。
    - `includes`：過濾要包含的分支的模式。包括使用 OR 操作。
    - `excludes`：要針對要排除的分支進行篩選的模式。排除使用 OR 操作。
  - `filePaths`：要篩選的檔案路徑名稱。
    - `includes`：要篩選的檔案路徑所要包含的模式。包括使用 OR 操作。
    - `excludes`：要篩選出要排除的檔案路徑的模式。排除使用 OR 操作。
  - `tags`：要篩選的標籤名稱。
    - `includes`：要篩選的樣式以取得要包含的標籤。包括使用 OR 操作。
    - `excludes`：要篩選的模式，以取得要排除的標籤。排除使用 OR 操作。
- `pullRequest`：提取請求事件，並在提取請求事件和提取請求過濾器上進行過濾。
  - `events`：根據指定的打開，更新或關閉的提取請求事件過濾器。
  - `branches`：要篩選的分支。分支在包含和排除之間使用 AND 運算。
    - `includes`：過濾要包含的分支的模式。包括使用 OR 操作。
    - `excludes`：要針對要排除的分支進行篩選的模式。排除使用 OR 操作。
  - `filePaths`：要篩選的檔案路徑名稱。
    - `includes`：要篩選的檔案路徑所要包含的模式。包括使用 OR 操作。
    - `excludes`：要篩選出要排除的檔案路徑的模式。排除使用 OR 操作。

AWS CloudFormation範本中的觸發篩選

您可以更新中的管線資源AWS CloudFormation以新增觸發器篩選。

下列範例範本程式碼片段提供觸發程序欄位定義的 YAML 參考。如需欄位定義的清單，請參閱在管線 JSON (CLI) 中觸發篩選。

```
pipeline:
  name: MyServicePipeline
  executionMode: PARALLEL
  triggers:
    - provider: CodeConnection
      gitConfiguration:
        sourceActionName: ApplicationSource
        push:
          - filePaths:
              includes:
```

```
                - projectA/**
                - common/**/*.js
              excludes:
                - '**/README.md'
                - '**/LICENSE'
                - '**/CONTRIBUTING.md'
            branches:
              includes:
                - feature/**
                - release/**
              excludes:
                - mainline
          - tags:
              includes:
                - release-v0
                - release-v1
              excludes:
                - release-v2
        pullRequest:
          - events:
              - CLOSED
            branches:
              includes:
                - feature/**
                - release/**
              excludes:
                - mainline
            filePaths:
              includes:
                - projectA/**
                - common/**/*.js
              excludes:
                - '**/README.md'
                - '**/LICENSE'
                - '**/CONTRIBUTING.md'
  stages:
    - name: Source
      actions:
        - name: ApplicationSource
          configuration:
            BranchName: mainline
            ConnectionArn: arn:aws:codestar-connections:eu-
central-1:111122223333:connection/fe9ff2e8-ee25-40c9-829e-65f85EXAMPLE
            FullRepositoryId: monorepo-example
```

```
            OutputArtifactFormat: CODE_ZIP
```

管理執行 CodePipeline

若要分析管線進度，您可以檢視錯誤記錄、檢視管線和動作執行歷程記錄，以及重試失敗的階段或動作。

主題

檢視執行 CodePipeline

您可以使用AWS CodePipeline主控台或檢視執行狀態、檢視執行歷史記錄，以及重試失敗的階段或動作。AWS CLI

主題

檢視管道執行歷程記錄 (主控台)

您可以使用 CodePipeline 主控台來檢視帳戶中所有管道的清單。您也可以檢視每個管道的詳細資訊，包括管道中最後一個動作的執行時間、階段之間的轉換是否啟用或停用、任何動作是否失敗等其他資訊。您也可以檢視歷程記錄頁面，該頁面中將顯示所有記錄在歷程記錄中的管道執行詳細資訊。執行歷程記錄會保留長達 12 個月。

> **Note**
>
> 在 2019 年 2 月 21 日當日或之後執行的作業，都有提供詳細的執行歷程記錄。

您可以使用主控台來檢視管道中的執行歷程記錄，包括每項執行的狀態、來源修訂，以及計時詳細資訊。

1. 請登入AWS Management Console並開啟 CodePipeline 主控台，網址為 http://console.aws.amazon.com/codesuite/codepipeline/home。

   所有與您的 AWS 帳戶相關的管道名稱，以及其狀態都會一併顯示。
2. 在 Name (名稱) 中，選擇管道的名稱。
3. 選擇 View history (檢視歷程記錄)。
4. 檢視與每次管道執行相關的狀態、來源修訂、變更詳細資訊及觸發。
5. 選擇一個執行。詳細檢視會顯示執行詳細資訊、時間軸標籤、視覺化標籤和變數標籤。管線層級變數的變數值會在管線執行時解析，並可在每次執行的執行歷史記錄中檢視。

> **Note**
>
> 您可以在每個動作執行的歷史記錄下的「輸出變數」標籤上檢視管線動作的輸出變數。

檢視執行狀態 (主控台)

您可以在執行歷程記錄頁面上的 Status (狀態) 中檢視管道狀態。選擇執行 ID 連結，然後檢視動作狀態。

以下是管道、階段和動作的有效狀態：

> **Note**
>
> 下列管線狀態也適用於作為輸入執行的管線執行。若要檢視輸入執行及其狀態，請參閱檢視輸入執行 (主控台)。

管道層級狀態

管道狀態	描述
InProgress	管道執行目前正在執行。
Stopping (正在停止)	由於要求停止並等待或停止並捨棄管道執行，因此管道執行正在停止中。

管道狀態	描述
已停止	停止程序已完成，且管道執行已停止。
Succeeded	管道執行已成功完成。
已取代	雖然此管道執行等待下一個階段完成，但較新的管道執行已改為透過管道前進並繼續。
失敗	管道執行未成功完成。

階段層級狀態

階段狀態	描述
InProgress	階段目前正在執行。
Stopping (正在停止)	由於要求停止並等待或停止並捨棄管道執行，因此階段執行正在停止中。
已停止	停止程序已完成，且階段執行已停止。
Succeeded	階段已成功完成。
失敗	階段未成功完成。

動作層級狀態

動作狀態	描述
InProgress	動作目前正在執行。
已捨棄	由於要求停止並捨棄管道執行，因而捨棄動作。
Succeeded	動作已成功完成。
失敗	對於核准動作，FAILED (失敗) 狀態表示檢閱者拒絕動作，或因動作組態不正確而失敗。

檢視輸入執行 (主控台)

您可以使用主控台來檢視輸入執行的狀態和詳細資料。啟用轉換或階段可用時，會InProgress繼續進入階段的輸入執行。具有Stopped狀態的入站執行不會進入階段。Failed如果編輯管線，輸入執行狀態會變更為。編輯管線時，不會繼續所有進行中的執行，且執行狀態會變更為。Failed

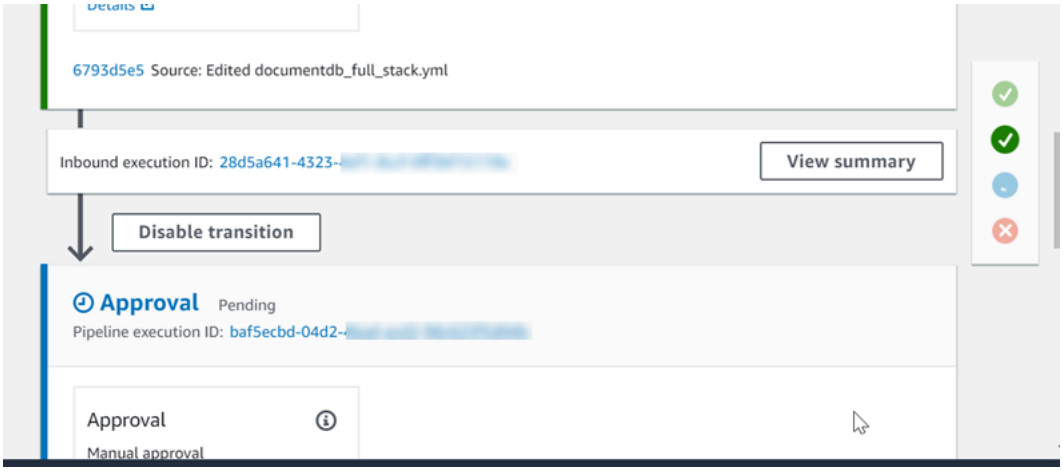
如果您沒有看到輸入執行，則在停用的階段轉換中沒有擱置的執行。

1. 請登入AWS Management Console並開啟 CodePipeline 主控台，網址為 http://console.aws.amazon.com/codesuite/codepipeline/home。

   將會顯示所有與您 AWS 帳戶建立關聯的管道名稱。

2. 選擇您要檢視其輸入執行項目的管線名稱，請執行下列其中一項作業：

   - 選擇 View (檢視)。在管線圖中，您可以在停用轉換前面的「輸入執行 ID」欄位中檢視輸入執行 ID。



   選擇檢視摘要來查看執行詳細資訊，例如執行 ID、來源觸發程式以及下一個階段的名稱。

   - 選擇管線，然後選擇檢視歷史記錄。

檢視管道執行來源修訂 (主控台)

您可以檢視管道執行中所用來源成品的詳細資訊 (源自管道第一個階段的輸出成品)。詳細資訊包括識別符，例如遞交 ID、簽入註解、以及使用 CLI 時的管道建置動作版本號碼。針對某些修訂類型，您可以檢視並開啟遞交的 URL。來源修訂包括下列項目：

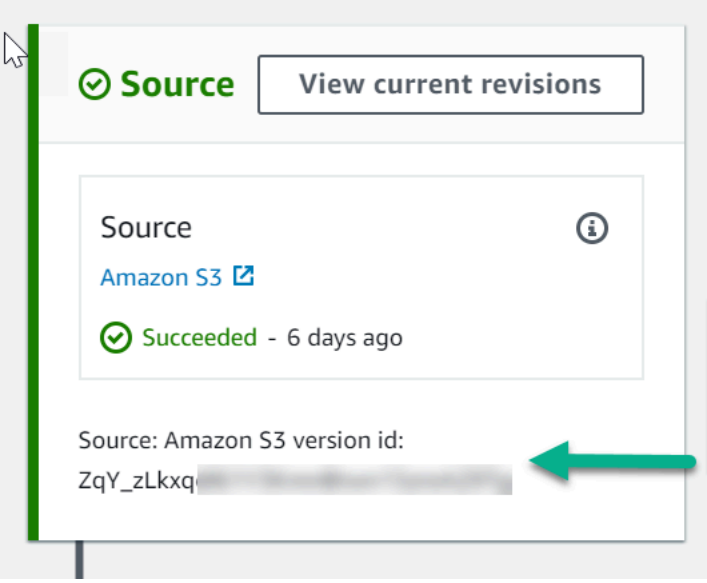
- 摘要：有關成品最新修訂版的摘要資訊。對於 GitHub 和 CodeCommit 存儲庫，提交消息。對於 Amazon S3 儲存貯體或動作，則是使用者提供的物件中繼資料中指定的 codepipeline-artifact-revision-summary 金鑰內容。
- revisionUrl：成品修訂版的修訂 URL (例如，外部儲存庫 URL)。
- revisionId：成品修訂版的修訂 ID。例如，對於 CodeCommit 或 GitHub 儲存庫中的來源變更，這是提交 ID。對於儲存在 GitHub 或儲存 CodeCommit 庫中的成品，提交 ID 會連結至提交詳細資訊頁面。

1. 請登入AWS Management Console並開啟 CodePipeline 主控台，網址為 http://console.aws.amazon.com/codesuite/codepipeline/home。

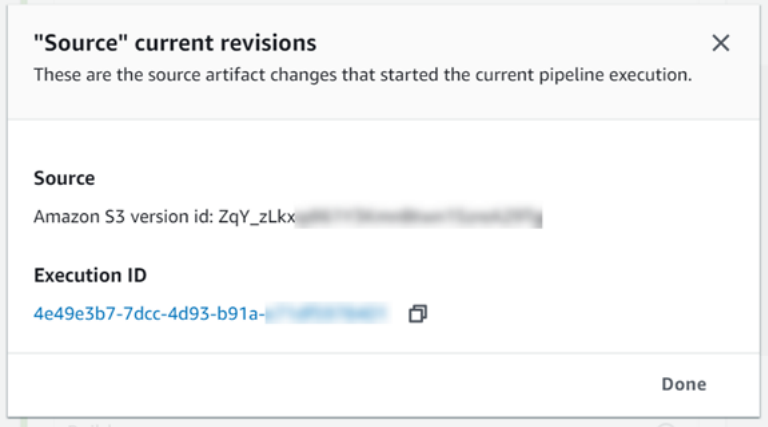
   AWS 帳戶將顯示與您關聯的所有配管的名稱。

2. 選擇您要檢視其來源修訂詳細資訊的管道名稱。執行下列任意一項：

   - 選擇 View history (檢視歷程記錄)。在 Source revisions (來源修訂) 中，會列出每項執行的來源變更。
   - 找出您要檢視其來源修訂詳細資訊的動作，然後找到其階段底部的修訂資訊：



   選擇 View current revisions (檢視目前修訂) 以檢視來源資訊。除了存放在 Amazon S3 儲存貯體的成品外，此資訊詳細資料檢視中的提交 ID 等識別碼會連結至成品的來源資訊頁面。



檢視動作執行 (主控台)

您可以檢視管道的動作詳細資訊，例如動作執行 ID，輸入成品、輸出成品和狀態。您可以在主控台中選擇管道，然後選擇執行 ID，來檢視動作詳細資訊。

> **Note**
>
> 在 2019 年 2 月 21 日當日或之後執行的作業，都有提供詳細的執行歷程記錄。

1. 請登入AWS Management Console並開啟 CodePipeline 主控台，網址為 http://console.aws.amazon.com/codesuite/codepipeline/home。

   所有與您的 AWS 帳戶相關的管道名稱都會顯示。
2. 選擇您要檢視動作詳細資訊的管道名稱，然後選擇 View history (檢視歷程記錄)。
3. 在 Execution ID (執行 ID) 中，選擇您要檢視動作執行詳細資訊的執行 ID。
4. 您可以在 Timeline (時間軸) 標籤上檢視以下資訊：

   a. 在 Action name (動作名稱) 中，選擇連結以開啟動作的詳細資訊頁面，您可以在這裡檢視狀態、階段名稱、動作名稱、組態資料和成品資訊。

   b. 在 Provider (供應商) 中，選擇連結以檢視動作供應商詳細資訊。例如，在前面的範例管線 CodeDeploy 中，如果您選擇「暫存」或「實際執行」階段，則會顯示為該階段設定之 CodeDeploy 應用程式的 CodeDeploy主控台頁面。

檢視動作成品和成品存放區資訊 (主控台)

您可以檢視動作的輸入和輸出成品動作詳細資訊。您也可以選擇連結，前往該動作的成品資訊。由於成品存放區使用版本控制，因此每個動作執行都有唯一的輸入和輸出成品位置。

1. 請登入AWS Management Console並開啟 CodePipeline 主控台，網址為 http://console.aws.amazon.com/codesuite/codepipeline/home。

   所有與您的 AWS 帳戶相關的管道名稱都會顯示。
2. 選擇您要檢視動作詳細資訊的管道名稱，然後選擇 View history (檢視歷程記錄)。
3. 在 Execution ID (執行 ID) 中，選擇您要檢視動作詳細資訊的執行 ID。
4. 在 Timeline (時間軸) 標籤上的 Action name (動作名稱)，選擇連結以開啟動作的詳細資訊頁面。
5. 在詳細資訊頁面的 [執行] 索引標籤上，檢視動作執行的狀態和時間。
6. 在 [組態] 索引標籤上，檢視動作的資源組態 (例如，組 CodeBuild 建專案名稱)。
7. 在「人工因素」標籤上，檢視人工因 Artifact 類型與人工因 Artifact 提供者中的人工因素 選擇 Artifact name (成品名稱) 下的連結，以檢視成品存放區中的成品。
8. 在「輸出變數」標籤上，從管線中的動作檢視已解析的變數，以執行動作。

檢視管道詳細資訊與歷程記錄 (CLI)

您可以執行下列命令來檢視關於管道與管道執行的詳細資訊：

- list-pipelines 命令，用以檢視所有與您的 AWS 帳戶相關之管道摘要。
- get-pipeline 命令，用以審閱單一管道的詳細資訊。
- list-pipeline-executions 以檢視最近期管道執行的摘要。
- get-pipeline-execution 以檢視管道執行相關資訊，包含成品詳細資訊、管道執行 ID、管道名稱、版本及狀態。
- get-pipeline-state 命令以檢視管道、階段和動作狀態。
- list-action-executions 以檢視管道的動作執行詳細資訊。

主題

- 檢視執行歷程記錄 (CLI)
- 檢視執行狀態 (CLI)

- 檢視輸入執行狀態 (CLI)
- 檢視來源修訂 (CLI)
- 檢視動作執行 (CLI)

檢視執行歷程記錄 (CLI)

您可以檢視管道執行歷程記錄。

- 若要檢視管道過去的執行相關詳細資訊，請執行 list-pipeline-executions 命令，指定管道的獨特名稱。例如，若要檢視名為 *MyFirstPipeline* 之管道目前狀態的詳細資訊，請輸入下列內容：

```
aws codepipeline list-pipeline-executions --pipeline-name MyFirstPipeline
```

此指令會傳回關於所有管道執行的摘要資訊；這些管道執行的歷程記錄已記錄的內容。摘要包含開始與結束時間、持續時間與狀態。

以下範例顯示名為 *MyFirstPipeline* 的管道所傳回的資料，其中有三項執行：

```
{
    "pipelineExecutionSummaries": [
        {
            "lastUpdateTime": 1496380678.648,
            "pipelineExecutionId": "7cf7f7cb-3137-539g-j458-d7eu3EXAMPLE",
            "startTime": 1496380258.243,
            "status": "Succeeded"
        },
        {
            "lastUpdateTime": 1496591045.634,
            "pipelineExecutionId": "3137f7cb-8d494hj4-039j-d84l-d7eu3EXAMPLE",
            "startTime": 1496590401.222,
            "status": "Succeeded"
        },
        {
            "lastUpdateTime": 1496946071.6456,
            "pipelineExecutionId": "4992f7jf-7cf7-913k-k334-d7eu3EXAMPLE",
            "startTime": 1496945471.5645,
            "status": "Succeeded"
        }
    ]
}
```

若要檢視管道執行的更多詳細資訊，請執行 get-pipeline-execution，指定管道執行的獨特 ID。例如，若要檢視前一範例中關於首次執行的更多詳細資訊，請輸入以下內容：

```
aws codepipeline get-pipeline-execution --pipeline-name MyFirstPipeline --pipeline-execution-id 7cf7f7cb-3137-539g-j458-d7eu3EXAMPLE
```

此命令會傳回管道執行相關的摘要資訊，包含成品詳細資訊、管道執行 ID、管道名稱、版本和狀態。

下列範例顯示名為 *MyFirstPipeline* 的管道所傳回的資料：

```
{
    "pipelineExecution": {
        "pipelineExecutionId": "3137f7cb-7cf7-039j-s83l-d7eu3EXAMPLE",
        "pipelineVersion": 2,
        "pipelineName": "MyFirstPipeline",
        "status": "Succeeded",
        "artifactRevisions": [
            {
                "created": 1496380678.648,
                "revisionChangeIdentifier": "1496380258.243",
                "revisionId": "7636d59f3c461cEXAMPLE8417dbc6371",
                "name": "MyApp",
                "revisionSummary": "Updating the application for feature 12-4820"
            }
        ]
    }
}
```

檢視執行狀態 (CLI)

您可以使用 CLI 來檢視管線、階段和動作狀態。

- 若要檢視管道目前狀態的相關詳細資訊，請執行 get-pipeline-state 命令，指定管道的獨特名稱。例如，若要檢視名為 *MyFirstPipeline* 之管道目前狀態的詳細資訊，請輸入下列內容：

  ```
  aws codepipeline get-pipeline-state --name MyFirstPipeline
  ```

  此命令會傳回管道中所有階段的目前狀態，以及此類階段中動作的狀態。

下列範例顯示名為 *MyFirstPipeline* 的三階段管道資料，其中前兩個階段與動作顯示成功、第三個則顯示失敗，而第二與第三階段間的轉換已停用。

```
{
    "updated": 1427245911.525,
    "created": 1427245911.525,
    "pipelineVersion": 1,
    "pipelineName": "MyFirstPipeline",
    "stageStates": [
        {
            "actionStates": [
                {
                    "actionName": "Source",
                    "entityUrl": "https://console.aws.amazon.com/s3/home?#",
                    "latestExecution": {
                        "status": "Succeeded",
                        "lastStatusChange": 1427298837.768
                    }
                }
            ],
            "stageName": "Source"
        },
        {
            "actionStates": [
                {
                    "actionName": "Deploy-CodeDeploy-Application",
                    "entityUrl": "https://console.aws.amazon.com/codedeploy/home?
#",
                    "latestExecution": {
                        "status": "Succeeded",
                        "lastStatusChange": 1427298939.456,
                        "externalExecutionUrl": "https://console.aws.amazon.com/?
#",
                        "externalExecutionId": ""c53dbd42-This-Is-An-Example"",
                        "summary": "Deployment Succeeded"
                    }
                }
            ],
            "inboundTransitionState": {
                "enabled": true
            },
            "stageName": "Staging"
```

```
        },
        {
            "actionStates": [
                {
                    "actionName": "Deploy-Second-Deployment",
                    "entityUrl": "https://console.aws.amazon.com/codedeploy/home?
#",
                    "latestExecution": {
                        "status": "Failed",
                        "errorDetails": {
                            "message": "Deployment Group is already deploying
 deployment ...",
                            "code": "JobFailed"
                        },
                        "lastStatusChange": 1427246155.648
                    }
                }
            ],
            "inboundTransitionState": {
                "disabledReason": "Disabled while I investigate the failure",
                "enabled": false,
                "lastChangedAt": 1427246517.847,
                "lastChangedBy": "arn:aws:iam::80398EXAMPLE:user/CodePipelineUser"
            },
            "stageName": "Production"
        }
    ]
}
```

檢視輸入執行狀態 (CLI)

您可以使用 CLI 檢視輸入執行狀態。啟用轉換或階段可用時，會InProgress繼續進入階段的輸入執行。具有Stopped狀態的入站執行不會進入階段。Failed如果編輯管線，輸入執行狀態會變更為。編輯管線時，不會繼續所有進行中的執行，且執行狀態會變更為。Failed

- 若要檢視管道目前狀態的相關詳細資訊，請執行 get-pipeline-state 命令，指定管道的獨特名稱。例如，若要檢視名為 *MyFirstPipeline* 之管道目前狀態的詳細資訊，請輸入下列內容：

```
aws codepipeline get-pipeline-state --name MyFirstPipeline
```

此命令會傳回管道中所有階段的目前狀態，以及此類階段中動作的狀態。輸出也會顯示每個階段中的管線執行 ID，以及停用轉移的階段是否有輸入執行 ID。

下列範例顯示名為的兩階段管線傳回資料 *MyFirstPipeline*，其中第一個階段顯示已啟用的轉移和成功的管線執行，而第二個名Beta為的階段則顯示停用的轉移和輸入執行 ID。入站執行可以具有InProgressStopped、或FAILED狀態。

```
{
    "pipelineName": "MyFirstPipeline",
    "pipelineVersion": 2,
    "stageStates": [
        {
            "stageName": "Source",
            "inboundTransitionState": {
                "enabled": true
            },
            "actionStates": [
                {
                    "actionName": "SourceAction",
                    "currentRevision": {
                        "revisionId": "PARcnxX_u0SMRBnKh83pHL09.zPRLLMu"
                    },
                    "latestExecution": {
                        "actionExecutionId": "14c8b311-0e34-4bda-EXAMPLE",
                        "status": "Succeeded",
                        "summary": "Amazon S3 version id: PARcnxX_u0EXAMPLE",
                        "lastStatusChange": 1586273484.137,
                        "externalExecutionId": "PARcnxX_u0EXAMPLE"
                    },
                    "entityUrl": "https://console.aws.amazon.com/s3/home?#"
                }
            ],
            "latestExecution": {
                "pipelineExecutionId": "27a47e06-6644-42aa-EXAMPLE",
                "status": "Succeeded"
            }
        },
        {
            "stageName": "Beta",
            "inboundExecution": {
                "pipelineExecutionId": "27a47e06-6644-42aa-958a-EXAMPLE",
                "status": "InProgress"
```

```
            },
            "inboundTransitionState": {
                "enabled": false,
                "lastChangedBy": "USER_ARN",
                "lastChangedAt": 1586273583.949,
                "disabledReason": "disabled"
            },
                    "currentRevision": {
            "actionStates": [
                {
                    "actionName": "BetaAction",
                    "latestExecution": {
                        "actionExecutionId": "a748f4bf-0b52-4024-98cf-EXAMPLE",
                        "status": "Succeeded",
                        "summary": "Deployment Succeeded",
                        "lastStatusChange": 1586272707.343,
                        "externalExecutionId": "d-KFGF3EXAMPLE",
                        "externalExecutionUrl": "https://us-
west-2.console.aws.amazon.com/codedeploy/home?#/deployments/d-KFGF3WTS2"
                    },
                    "entityUrl": "https://us-west-2.console.aws.amazon.com/
codedeploy/home?#/applications/my-application"
                }
            ],
            "latestExecution": {
                "pipelineExecutionId": "f6bf1671-d706-4b28-EXAMPLE",
                "status": "Succeeded"
            }
        }
    ],
    "created": 1585622700.512,
    "updated": 1586273472.662
}
```

檢視來源修訂 (CLI)

您可以檢視管道執行中所用來源成品的相關詳細資訊 (源自管道第一個階段的輸出成品)。詳細資訊包括識別符，例如遞交 ID、簽入註解、建立或更新成品之後的時間，以及使用 CLI 時的建置動作版本號碼。針對某些修訂類型，您可以檢視並開啟成品版本的遞交 URL。來源修訂包括下列項目：

- 摘要：有關成品最新修訂版的摘要資訊。對於 GitHub 和AWS CodeCommit存儲庫，提交消息。對於 Amazon S3 儲存貯體或動作，則是使用者提供的物件中繼資料中指定的 codepipeline-artifact-revision-summary 金鑰內容。
- revisionUrl：成品修訂版的遞交 ID。對於儲存在 GitHub 或儲存AWS CodeCommit庫中的成品，提交 ID 會連結至提交詳細資訊頁面。

您可以執行 get-pipeline-execution 命令，以檢視管道執行中所含之最新來源修訂的相關資訊。在您初次執行 get-pipeline-state 命令以取得管道中所有階段的詳細資訊之後，即可識別執行 ID 以套用至您想要其來源修訂詳細資訊的階段。然後，您將執行 ID 用於 get-pipeline-execution 命令。(由於管道中的階段可能在不同的管道執行期間最後已成功完成，因此可能有不同的執行 ID)。

換言之，如果您想要檢視目前在 Staging 階段中成品的詳細資訊，請執行 get-pipeline-state 命令，並識別 Staging 階段的目前執行 ID，然後使用該執行 ID 來執行 get-pipeline-execution 命令。

檢視管道中的來源修訂

1. 開啟終端機 (Linux、macOS 或 Unix) 或命令提示字元 (視窗)，然後使用AWS CLI來執行get-pipeline-state指令。針對名為 *MyFirstPipeline* 的管道，請輸入：

   ```
   aws codepipeline get-pipeline-state --name MyFirstPipeline
   ```

   此命令會傳回管道的最新狀態 (包含每個階段的最新管道執行 ID)。

2. 若要檢視管道執行的詳細資訊，請執行 get-pipeline-execution 命令，並指定管道的唯一名稱以及您想要檢視其成品詳細資訊之執行的管道執行 ID。例如，若要檢視名為 *MyFirstPipeline* 且執行 ID 為 3137f7cb-7cf7-039j-s83l-d7eu3EXAMPLE 之管道的執行詳細資訊，您可以輸入下列命令：

   ```
   aws codepipeline get-pipeline-execution --pipeline-name MyFirstPipeline --pipeline-execution-id 3137f7cb-7cf7-039j-s83l-d7eu3EXAMPLE
   ```

   此命令會傳回屬於管道執行一部分之每個來源修訂的資訊，以及管道的識別資訊。只會包含該執行中所含管道階段的資訊。管道中可能會有不屬於該管道執行一部分的其他階段。

   下列範例顯示管線部分的傳回資料 *MyFirstPipeline*，其中名為 "MyApp" 的成品儲存在儲存 GitHub 庫中：

3.
   ```
   {
       "pipelineExecution": {
   ```

```
        "artifactRevisions": [
            {
                "created": 1427298837.7689769,
                "name": "MyApp",
                "revisionChangeIdentifier": "1427298921.3976923",
                "revisionId": "7636d59f3c461cEXAMPLE8417dbc6371",
                "revisionSummary": "Updating the application for feature 12-4820",
                "revisionUrl": "https://api.github.com/repos/anycompany/MyApp/git/
commits/7636d59f3c461cEXAMPLE8417dbc6371"
            }
            //More revisions might be listed here
        ],
        "pipelineExecutionId": "3137f7cb-7cf7-039j-s83l-d7eu3EXAMPLE",
        "pipelineName": "MyFirstPipeline",
        "pipelineVersion": 2,
        "status": "Succeeded"
    }
}
```

檢視動作執行 (CLI)

您可以檢視管道的動作執行詳細資訊，例如動作執行 ID，輸入成品、輸出成品、執行結果和狀態。您提供執行 ID 篩選條件，以傳回管道執行中的動作清單：

> **Note**
>
> 在 2019 年 2 月 21 日當日或之後執行的作業，都有提供詳細的執行歷程記錄。

- 若要檢視管道的動作執行，請執行下列其中一項：
  - 若要檢視管道中所有動作執行的詳細資訊，請執行 list-action-executions 命令，指定管道的獨特名稱。例如，若要檢視名為 *MyFirstPipeline* 管道中的動作執行，請輸入下列內容：

    ```
    aws codepipeline list-action-executions --pipeline-name MyFirstPipeline
    ```

    以下顯示此命令的部分範例輸出：

    ```
    {
        "actionExecutionDetails": [
    ```

```
        {
            "actionExecutionId": "ID",
            "lastUpdateTime": 1552958312.034,
            "startTime": 1552958246.542,
            "pipelineExecutionId": "Execution_ID",
            "actionName": "Build",
            "status": "Failed",
            "output": {
                "executionResult": {
                    "externalExecutionUrl": "Project_ID",
                    "externalExecutionSummary": "Build terminated with state:
 FAILED",
                    "externalExecutionId": "ID"
                },
                "outputArtifacts": []
            },
            "stageName": "Beta",
            "pipelineVersion": 8,
            "input": {
                "configuration": {
                    "ProjectName": "java-project"
                },
                "region": "us-east-1",
                "inputArtifacts": [
                    {
                        "s3location": {
                            "bucket": "codepipeline-us-east-1-ID",
                            "key": "MyFirstPipeline/MyApp/Object.zip"
                        },
                        "name": "MyApp"
                    }
                ],
                "actionTypeId": {
                    "version": "1",
                    "category": "Build",
                    "owner": "AWS",
                    "provider": "CodeBuild"
                }
            }
        },

. . .
```

- 若要檢視管道執行中所有動作執行，請執行 list-action-executions 命令，指定管道的獨特名稱和執行 ID。例如，若要檢視*## ID* 的動作執行，請輸入下列內容：

```
aws codepipeline list-action-executions --pipeline-name MyFirstPipeline --filter
 pipelineExecutionId=Execution_ID
```

- 以下顯示此命令的部分範例輸出：

```
{
    "actionExecutionDetails": [
        {
            "stageName": "Beta",
            "pipelineVersion": 8,
            "actionName": "Build",
            "status": "Failed",
            "lastUpdateTime": 1552958312.034,
            "input": {
                "configuration": {
                    "ProjectName": "java-project"
                },
                "region": "us-east-1",
                "actionTypeId": {
                    "owner": "AWS",
                    "category": "Build",
                    "provider": "CodeBuild",
                    "version": "1"
                },
                "inputArtifacts": [
                    {
                        "s3location": {
                            "bucket": "codepipeline-us-east-1-ID",
                            "key": "MyFirstPipeline/MyApp/Object.zip"
                        },
                        "name": "MyApp"
                    }
                ]
            },


. . .
```

設定管線執行模式

您可以設定管線的執行模式，以指定處理多個執行的方式。

如需有關管線執行模式的詳細資訊，請參閱管道執行的運作方式。

> **Important**
>
> 對於「平行」模式下的配管，將配管執行模式編輯為「佇列」或「已取代」時，配管狀態不會將更新狀態顯示為「平行」。如需詳細資訊，請參閱 從平行模式變更的管線會顯示先前的執行模式。

> **Important**
>
> 對於平行模式下的配管，將配管執行模式編輯為 QUEED 或已取代時，不會更新每個模式下配管的配管定義。如需詳細資訊，請參閱 如果在變更為「佇列」或「已取代」模式時進行編輯，則「平行」模式下的管線定義已過期。

設定管線執行模式 (主控台)

您可以使用控制台來設定管線執行模式。

1. 請登入 AWS Management Console 並開啟 CodePipeline 主控台，網址為 http://console.aws.amazon.com/codesuite/codepipeline/home。

   會顯示與您 AWS 帳戶相關聯的所有管道的名稱和狀態。
2. 在 Name (名稱) 中，選擇您想編輯的管道名稱。
3. 在管道詳細資訊頁面上，選擇 Edit (編輯)。
4. 在 「編輯」 頁面上，選擇編輯：配管屬性。
5. 選擇管道的模式。

   - 已取代
   - 已佇列 (需要配管類型 V2)
   - 平行 (需要配管類型 V2)
6. 在 [編輯] 頁面上，選擇 [完成]。

設定管線執行模式 (CLI)

若要使用 AWS CLI 來設定配管執行模式，請使用`create-pipeline`或`update-pipeline`指令。

1. 開啟終端機工作階段 (Linux、macOS 或 Unix) 或命令提示字元 (Windows)，然後執行get-pipeline指令，將管線結構複製到 JSON 檔案中。例如，針對名為 **MyFirstPipeline** 的管道輸入下列命令：

   ```
   aws codepipeline get-pipeline --name MyFirstPipeline >pipeline.json
   ```

   此命令不會傳回任何內容，但您建立的檔案應該會顯示在您執行命令的目錄中。

2. 在任何純文字編輯器中開啟 JSON 檔案，並修改檔案結構以反映您要設定的管線執行模式，例如 QUEED。

   ```
   "executionMode": "QUEUED"
   ```

   下列範例顯示如何在具有兩個階段的範例管線中將執行模式設定為 QUEED。

   ```
   {
       "pipeline": {
           "name": "MyPipeline",
           "roleArn": "arn:aws:iam::111122223333:role/service-role/
   AWSCodePipelineServiceRole-us-east-1-dkpippe",
           "artifactStore": {
               "type": "S3",
               "location": "bucket"
           },
           "stages": [
               {
                   "name": "Source",
                   "actions": [
                       {
                           "name": "Source",
                           "actionTypeId": {
                               "category": "Source",
                               "owner": "AWS",
                               "provider": "CodeCommit",
                               "version": "1"
                           },
                           "runOrder": 1,
                           "configuration": {
   ```

```
                        "BranchName": "main",
                        "OutputArtifactFormat": "CODE_ZIP",
                        "PollForSourceChanges": "true",
                        "RepositoryName": "MyDemoRepo"
                    },
                    "outputArtifacts": [
                        {
                            "name": "SourceArtifact"
                        }
                    ],
                    "inputArtifacts": [],
                    "region": "us-east-1",
                    "namespace": "SourceVariables"
                }
            ]
        },
        {
            "name": "Build",
            "actions": [
                {
                    "name": "Build",
                    "actionTypeId": {
                        "category": "Build",
                        "owner": "AWS",
                        "provider": "CodeBuild",
                        "version": "1"
                    },
                    "runOrder": 1,
                    "configuration": {
                        "ProjectName": "MyBuildProject"
                    },
                    "outputArtifacts": [
                        {
                            "name": "BuildArtifact"
                        }
                    ],
                    "inputArtifacts": [
                        {
                            "name": "SourceArtifact"
                        }
                    ],
                    "region": "us-east-1",
                    "namespace": "BuildVariables"
                }
```

```
                ]
            }
        ],
        "version": 1,
        "executionMode": "QUEUED"
    }
}
```

3. 如果您使用的是使用 get-pipeline 命令擷取的管道結構，您必須在 JSON 檔案中修改結構。您必須從檔案移除 `metadata` 行，以讓 update-pipeline 命令可以使用它。從 JSON 檔案的管道結構移除此區段 (`"metadata": { }` 行以及 `"created"`、`"pipelineARN"` 和 `"updated"` 欄位)。

   例如，從結構中移除下列幾行：

```
"metadata": {
  "pipelineArn": "arn:aws:codepipeline:region:account-ID:pipeline-name",
  "created": "date",
  "updated": "date"
  }
```

   儲存檔案。

4. 若要套用您的變更，請執行 update-pipeline 命令、指定管道 JSON 檔案：

> **Important**
>
> 請確認在檔案名稱之前包含 `file://`。這是此命令必要項目。

```
aws codepipeline update-pipeline --cli-input-json file://pipeline.json
```

此命令會傳回所編輯管道的整個結構。

> **Note**
>
> update-pipeline 命令將終止管道。若在您執行 update-pipeline 命令時有修訂正在透過管道執行，該執行將停止。您必須手動啟動管道，以透過更新的管道執行該修訂。

重試階段中失敗的階段或失敗的動作

您可以重試失敗的階段，而不必從頭開始再次執行管線。您可以透過在階段中重試失敗的動作，或重試階段中的所有動作，從階段中的第一個動作開始執行此操作。當您重試階段中的失敗動作時，所有仍在進行中的動作會繼續運作，而失敗的動作會再次觸發。當您從階段中的第一個動作重試失敗的階段時，階段不會有任何正在進行中的動作。在可以重試階段之前，它必須讓所有動作都失敗，或是某些動作失敗且某些動作成功。

> **Important**
>
> 重試失敗的階段會從階段中的第一個動作重試階段中的所有動作，重試失敗的動作會重試階段中所有失敗的動作。這會覆寫相同執行中先前成功動作的輸出成品。
> 雖然可能會覆寫人工因素，但仍會保留先前成功動作的執行歷程記錄。

如果您使用主控台來檢視管線，則可重試階段按鈕或重試失敗動作按鈕會出現在可重試的階段上。

如果您使用 AWS CLI，可以使用get-pipeline-state命令來判斷是否有任何動作失敗。

> **Note**
>
> 在下列情況下，您可能無法重試階段：
>
> - 階段中的所有動作都成功，因此階段不會處於失敗狀態。
> - 階段失敗後，整體配管結構已變更。
> - 階段中的另一個重試正在處理中。

主題

重試失敗的階段 (主控台)

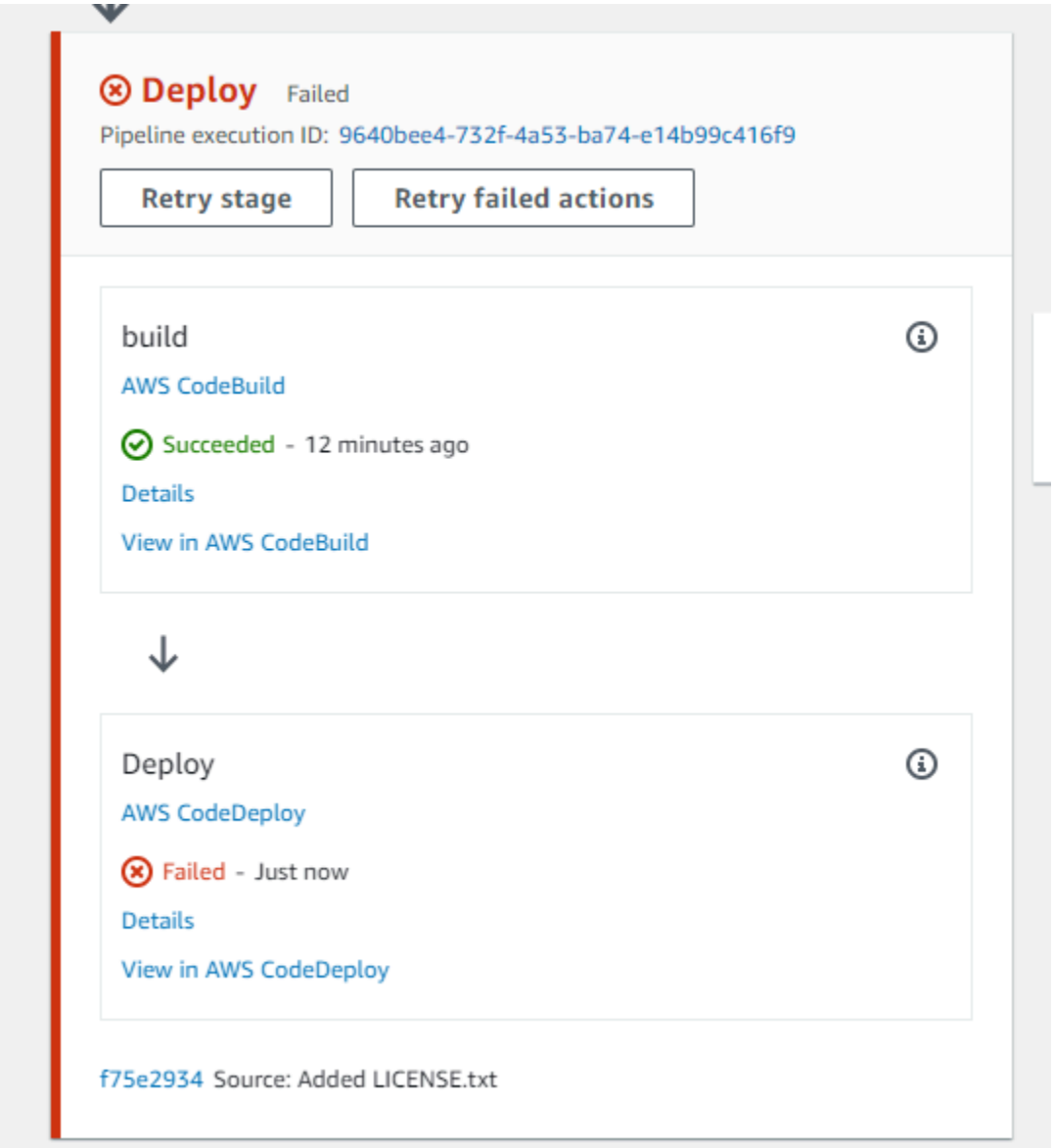
在階段-主控台中重試失敗的階段或失敗的動作

1. 請登入AWS Management Console並開啟 CodePipeline 主控台，網址為 http://console.aws.amazon.com/codesuite/codepipeline/home。

所有與您的 AWS 帳戶相關的管道名稱都會顯示。

2. 在 Name (名稱) 中，選擇管道的名稱。
3. 找出具有失敗動作的階段，然後選擇下列其中一項：

   - 若要重試階段中的所有動作，請選擇「重試階段」。
   - 若要僅重試階段中失敗的動作，請選擇「重試失敗的動作」。



若階段中所有重試動作皆已成功完成，管道將持續執行。

重試失敗的階段 (CLI)

若要在階段中重試失敗的階段或失敗的動作-CLI

若要使用重AWS CLI試所有動作或所有失敗的動作，請使用下列參數執行retry-stage-execution命令：

```
--pipeline-name <value>
--stage-name <value>
--pipeline-execution-id <value>
--retry-mode ALL_ACTIONS/FAILED_ACTIONS
```

> **Note**
>
> 您可以使用的值`retry-mode`是`FAILED_ACTIONS`和`ALL_ACTIONS`。

1. 在終端機 (Linux、macOS 或 Unix) 或命令提示字元 (Windows) 上，執行retry-stage-execution命令，如下列範例所示，針對名稱為的管線`MyPipeline`。

   ```
   aws codepipeline retry-stage-execution --pipeline-name MyPipeline --stage-name
    Deploy --pipeline-execution-id b59babff-5f34-EXAMPLE --retry-mode FAILED_ACTIONS
   ```

   輸出會傳回執行 ID：

   ```
   {
       "pipelineExecutionId": "b59babff-5f34-EXAMPLE"
   }
   ```

2. 您也可以使用 JSON 輸入檔案來執行命令。首先您必須建立 JSON 檔案，此檔案指出管道、包含失敗動作的階段，以及該階段中最新的管道執行。接著使用 `--cli-input-json` 參數來執行 retry-stage-execution 命令。若要擷取 JSON 檔案所需的詳細資訊，使用 get-pipeline-state 命令是最簡單的方法。

   a. 在終端機 (Linux、macOS 或 Unix) 或命令提示字元 (Windows) 上，在管線上執行get-pipeline-state命令。例如，對於名為的管線 MyFirstPipeline，您可以鍵入類似下列內容的內容：

      ```
      aws codepipeline get-pipeline-state --name MyFirstPipeline
      ```

對於命令的回應包括各階段的管道狀態資訊。在以下範例中，回應表示在預備 (Staging) 階段中失敗的一個或多個動作。

```
{
    "updated": 1427245911.525,
    "created": 1427245911.525,
    "pipelineVersion": 1,
    "pipelineName": "MyFirstPipeline",
    "stageStates": [
        {
            "actionStates": [...],
            "stageName": "Source",
            "latestExecution": {
                "pipelineExecutionId": "9811f7cb-7cf7-SUCCESS",
                "status": "Succeeded"
            }
        },
        {
            "actionStates": [...],
            "stageName": "Staging",
            "latestExecution": {
                "pipelineExecutionId": "3137f7cb-7cf7-EXAMPLE",
                "status": "Failed"
            }
        }
    ]
}
```

b. 在純文字編輯器中，以 JSON 格式來建立您將用於記錄下列內容的檔案：

- 包含失敗動作之管道的名稱
- 包含失敗動作之階段的名稱
- 該階段中最新管道執行的 ID
- 重試模式。

在前面的 MyFirstPipeline 例子中，您的文件看起來像這樣：

```
{
    "pipelineName": "MyFirstPipeline",
    "stageName": "Staging",
```

```
    "pipelineExecutionId": "3137f7cb-7cf7-EXAMPLE",
    "retryMode": "FAILED_ACTIONS"
}
```

c. 以類似 **retry-failed-actions.json** 的名稱儲存檔案。

d. 呼叫您執行 retry-stage-execution 命令時建立的檔案。例如：

> **Important**
>
> 請確認在檔案名稱之前包含 `file://`。這是此命令必要項目。

```
aws codepipeline retry-stage-execution --cli-input-json file://retry-failed-
actions.json
```

e. 若要檢視重試嘗試的結果，請開啟 CodePipeline 主控台並選擇包含失敗動作的管線，或再次使用get-pipeline-state命令。如需詳細資訊，請參閱 在中檢視管線和詳細資訊 CodePipeline。

使用中的動作 CodePipeline

在 AWS CodePipeline 中，動作是管道階段中序列的一部分。它是對該階段中成品所執行的任務。管道動作會以階段組態中決定的指定順序 (連續或平行) 發生。

CodePipeline 提供六種動作類型的支援：

- 來源
- 組建
- 測試
- 部署
- 核准
- Invoke

如需有關可根據動作類型整合到管道中的和合作夥伴產品和服務的資訊，請參閱與 CodePipeline 動作類型的整合。AWS 服務

主題

使用動作類型

動作類型是您身為提供者的預先設定動作，可在中使用其中一個支援的整合模型，為客戶建立這些動作 AWS CodePipeline。

您可以要求、檢視和更新動作類型。如果動作類型是以擁有者身分為您的帳戶建立的，您可以使用AWS CLI來檢視或更新動作類型內容和結構。如果您是動作類型的提供者或擁有者，您的客戶可以選擇該動作，並在中提供動作後將其新增至其管道CodePipeline。

> **Note**
>
> 您可以在`owner`欄位`custom`中建立要與工作 Worker 一起執行的動作。您不會使用整合模型來建立它們。如需有關自訂動作的資訊，請參閱在 CodePipeline 中建立和新增自訂動作。

動作類型元件

下列元件構成動作類型。

- 動作類型 ID — ID 由類別、擁有者、提供者和版本組成。下列範例顯示具有擁有者、類別`ThirdParty`、名為`TestProvider`的`Test`提供者以及版本的動作類型 ID`1`。

```
{
    "Category": "Test",
    "Owner": "ThirdParty",
    "Provider": "TestProvider",
    "Version": "1"
},
```

- 執行程式組態 — 建立動作時指定的整合模型或動作引擎。當您指定動作類型的執行程式時，您可以選擇以下兩種類型之一：
  - Lambda：動作類型擁有者將整合寫入為 Lambda 函數，CodePipeline只要有動作可用的工作，就會叫用該函數。
  - JobWorker：動作類型擁有者將整合寫入為工作工作者，該工作者會輪詢客戶管線上的可用作業。然後，工作工作者會執行工作，並使用CodePipeline API 將CodePipeline作業結果提交回。

    > **Note**
    >
    > 工作者整合模型不是偏好的整合模型。

- 輸入與輸出人工因素：動作類型擁有者為動作客戶指定的人工因素限制。
- 權限：指定可存取協力廠商動作類型之客戶的權限策略。可用的權限策略取決於動作類型選擇的整合模型。

- URL：客戶可以與之互動的資源的深層連結，例如動作類型擁有者的設定頁面。

主題

請求動作類型

當第三方提供者要求新CodePipeline動作類型時，會針對中的動作類型擁有者建立動作類型 CodePipeline，且擁有者可以管理和檢視動作類型。

動作類型可以是私人動作或公開動作。當您建立動作類型時，它是私有的。若要請求將動作類型變更為公開動作，請聯絡CodePipeline服務團隊。

在您為CodePipeline小組建立動作定義檔案、執行程式資源和動作類型要求之前，您必須選擇整合模型。

步驟 1：選擇您的整合模式

選擇您的整合模型，然後建立該模型的組態。選擇整合模型之後，您必須設定整合資源。

- 對於 Lambda 整合模型，您可以建立 Lambda 函數並新增許可。將權限新增至整合商 Lambda 函數，以便為CodePipeline服務提供使用服CodePipeline務主體叫用該函數的許可：`codepipeline.amazonaws.com`。可以使用AWS CloudFormation或命令列來新增權限。
  - 使用以AWS CloudFormation下方式新增權限的範例

```
  CodePipelineLambdaBasedActionPermission:
    Type: 'AWS::Lambda::Permission'
    Properties:
      Action: 'lambda:invokeFunction'
      FunctionName: {"Fn::Sub": "arn:${AWS::Partition}:lambda:${AWS::Region}:
${AWS::AccountId}:function:function-name"}
      Principal: codepipeline.amazonaws.com
```

  - 命令列的文件

- 對於工作 Worker 整合模型，您可以使用允許的帳戶清單建立整合，其中工作者會輪詢使用 CodePipeline API 的工作。

步驟 2：建立動作類型定義檔案

您可以使用 JSON 在動作類型定義檔案中定義動作類型。在檔案中，您可以包括動作類別、用於管理動作類型的整合模型，以及組態屬性。

> **Note**
>
> 建立公用動作之後，您無法將下方的動作類型屬性`properties`從變更`optional`為`required`。您也無法變更`owner`.

如需有關動作類型定義檔案參數的詳細資訊，請參閱 CodePipelineAPI 參考UpdateActionType中的ActionTypeDeclaration和。

動作類型定義檔案中有八個區段：

- `description`：要更新之動作類型的說明。
- `executor`：使用支援整合模型 (Lambda或) 建立之動作類型之執行程式的相關資訊`job worker`。您只能根據您的執行人類型提供`jobWorkerExecutorConfiguration`或`lambdaExecutorConfiguration`。
  - `configuration`：動作類型組態的資源，以選擇的整合模型為基礎。對於 Lambda 整合模型，請使用 Lambda 函數 ARN。對於工作 Worker 整合模型，請使用工作 Worker 執行來源的帳戶或帳戶清單。
  - `jobTimeout`：作業的逾時 (以秒為單位)。一個動作執行可以由多個工作組成。這是單一工作的逾時，而不是整個動作執行。

    > **Note**
    >
    > 對於 Lambda 整合模型，逾時時間上限為 15 分鐘。

  - `policyStatementsTemplate`：指定客戶帳CodePipeline戶中成功執行動作所需權限的政策陳述式。
  - `type`：用來建立和更新動作類型 (Lambda或) 的整合模型`JobWorker`。
- `id`：動作類型的類別、擁有者、提供者和版本 ID：

  - category：在階段中可採用的動作種類：來源、建置、部署、測試、叫用或核准。
  - provider：所呼叫動作類型的提供者，例如提供者公司或產品名稱。建立動作類型時會提供提供者名稱。
  - owner：所呼叫之動作類型的建立者：AWS或ThirdParty。
  - version：用來版本化動作類型的字串。對於第一個版本，將版本號碼設定為 1。
- inputArtifactDetails：管道中上一個階段的預期成品數量。
- outputArtifactDetails：動作類型階段的結果所預期的成品數目。
- permissions：識別具有使用動作類型之權限之帳戶的詳細資訊。
- properties：完成專案作業所需的參數。
  - description：顯示給使用者的屬性說明。
  - optional：是否配置屬性是可選的。
  - noEcho：是否從日誌中省略客戶輸入的字段值。如果true，則當使用GetPipeline API 請求傳回時，會編輯該值。
  - key：組態屬性是否為索引鍵。
  - queryable：該屬性是否與輪詢一起使用。動作類型最多可具有一個可查詢的屬性。如果有，則該屬性必須是必要的，而且不是私密。
  - name：顯示給使用者的屬性名稱。
- urls：URL 清單會CodePipeline顯示給您的使用者。
  - entityUrlTemplate：動作類型的外部資源的 URL，例如配置頁面。
  - executionUrlTemplate：最近一次執行動作的詳細資訊的 URL。
  - revisionUrlTemplate：客戶可以更新或變更外部動作組態CodePipeline主控台中的 URL。
  - thirdPartyConfigurationUrl：頁面的 URL，使用者可以在其中註冊外部服務，並執行該服務所提供之動作的初始設定。

下列程式碼顯示動作類型定義檔案的範例。

```
{
   "actionType": {
      "description": "string",
      "executor": {
         "configuration": {
            "jobWorkerExecutorConfiguration": {
               "pollingAccounts": [ "string" ],
               "pollingServicePrincipals": [ "string" ]
```

```
            },
            "lambdaExecutorConfiguration": {
               "lambdaFunctionArn": "string"
            }
         },
         "jobTimeout": number,
         "policyStatementsTemplate": "string",
         "type": "string"
      },
      "id": {
         "category": "string",
         "owner": "string",
         "provider": "string",
         "version": "string"
      },
      "inputArtifactDetails": {
         "maximumCount": number,
         "minimumCount": number
      },
      "outputArtifactDetails": {
         "maximumCount": number,
         "minimumCount": number
      },
      "permissions": {
         "allowedAccounts": [ "string" ]
      },
      "properties": [
         {
            "description": "string",
            "key": boolean,
            "name": "string",
            "noEcho": boolean,
            "optional": boolean,
            "queryable": boolean
         }
      ],
      "urls": {
         "configurationUrl": "string",
         "entityUrlTemplate": "string",
         "executionUrlTemplate": "string",
         "revisionUrlTemplate": "string"
      }
   }
```

```
}
```

步驟 3：註冊您的整合CodePipeline

要向註冊您的操作類型CodePipeline，請聯繫CodePipeline服務團隊提出您的請求。

CodePipeline服務團隊通過在服務代碼庫中進行更改來註冊新的操作類型集成。CodePipeline註冊兩個新行動：公共行動和私人行動。您使用私人操作進行測試，然後在準備就緒時激活公共行動以服務客戶流量。

若要註冊 Lambda 整合的請求

- 使用以下表單向CodePipeline服務團隊發送請求。

```
This issue will track the onboarding of [Name] in CodePipeline.


[Contact engineer] will be the primary point of contact for this integration.

Name of the action type as you want it to appear to customers: Example.com Testing

Initial onboard checklist:

1. Attach an action type definition file in JSON format. This includes the schema
 for the action type

2. A list of test accounts for the allowlist which can access the new action type
 [{account, account_name}]

3. The Lambda function ARN

4. List of AWS ## where your action will be available

5. Will this be available as a public action?
```

若要註冊工作者整合的請求

- 使用以下表單向CodePipeline服務團隊發送請求。

```
This issue will track the onboarding of [Name] in CodePipeline.
```

```
[Contact engineer] will be the primary point of contact for this integration.


Name of the action type as you want it to appear to customers: Example.com Testing

Initial onboard checklist:

1. Attach an action type definition file in JSON format. This includes the schema
 for the action type.

2. A list of test accounts for the allowlist which can access the new action type
 [{account, account_name}]

3. URL information:
Website URL: https://www.example.com/%TestThirdPartyName%/%TestVersionNumber%

Example URL pattern where customers will be able to review their configuration
 information for the action: https://www.example.com/%TestThirdPartyName%/
%customer-ID%/%CustomerActionConfiguration%

Example runtime URL pattern: https://www.example.com/%TestThirdPartyName%/
%customer-ID%/%TestRunId%

4. List of AWS ## where your action will be available

5. Will this be available as a public action?
```

步驟 4：啟用您的新整合

當您準備好公開使用新的整合時，請與CodePipeline服務團隊聯絡。

將可用的動作類型新增至管線 (主控台)

您可以將動作類型新增至管線，以便對其進行測試。您可以透過建立新配管或編輯現有配管來執行此操作。

> **Note**
>
> 如果您的動作類型是來源、組建或部署類別動作，您可以透過建立管線來新增它。如果您的動作類型位於測試類別中，則必須透過編輯現有管線來新增動作類型。

若要從CodePipeline主控台將動作類型新增至現有管線

1. 請登入AWS Management Console並開啟CodePipeline主控台，網址為 http://console.aws.amazon.com/codesuite/codepipeline/home。
2. 在配管清單中，選擇您要新增動作類型。
3. 在配管的摘要檢視頁面上，選擇 「編輯」(Edit)。
4. 選擇編輯階段。在您要新增動作類型的階段中，選擇 [新增動作群組]。「編輯」動作頁面隨即顯示。
5. 在 [編輯動作] 頁面上的 [動作名稱] 中，輸入動作的名稱。這是管線中階段所顯示的名稱。
6. 在 [動作提供者] 中，從清單中選擇您的動作類型。

   請注意，清單中的值是以動作類型定義檔案中`provider`指定的值為基礎。
7. 在輸入人工因素中，以下列格式輸入人工因素名稱：

   `Artifactname``::`*`FileName`*

   請注意，允許的最小和最大數量是根據動作類型定義檔案中`inputArtifactDetails`指定的數量來定義的。
8. 選擇「Connect 至」<Action_Name>。

   瀏覽器視窗隨即開啟，並連線至您針對動作類型建立的網站。
9. 以客戶身份登錄到您的網站，並完成客戶使用您的操作類型所採取的步驟。您的步驟會根據您的動作類別、網站和設定而有所不同，但通常會包含將客戶返回 「編輯動作」頁面的完成動作。
10. 在CodePipeline 「編輯」動作頁面中，會顯示動作的其他組態欄位。顯示的欄位是您在動作定義檔案中指定的組態特性。在針對您的動作類型自訂的欄位中輸入資訊。

    例如，如果動作定義檔案指定了名為的屬性`Host`，則標籤為「主機」(Host) 的欄位會顯示在您動作的「編輯」動作頁面上。
11. 在輸出人工因素中，以下列格式輸入人工因素名稱：

    `Artifactname``::`*`FileName`*

    請注意，允許的最小和最大數量是根據動作類型定義檔案中`outputArtifactDetails`指定的數量來定義的。
12. 選擇「完成」以返回管線詳細資訊頁面。

> **Note**
>
> 您的客戶可以選擇性地使用 CLI 將動作類型新增至其管道。

13. 若要測試您的動作，請將變更送至管線的來源階段中指定的來源，或遵循手動啟動管線中的步驟執行。

若要使用您的動作類型建立管道，請遵循中的步驟，在中建立管線 CodePipeline並從您要測試的階段中選擇動作類型。

檢視動作類型

您可以使用 CLI 檢視動作類型。使用指get-action-type令可檢視已使用整合模型建立的動作類型。

若要檢視動作類型

1. 創建一個輸入 JSON 文件並命名該文件`file.json`。以 JSON 格式新增動作類型 ID，如下範例所示。

```
{
    "category": "Test",
    "owner": "ThirdParty",
    "provider": "TestProvider",
    "version": "1"
}
```

2. 在終端機視窗或命令列中執行命get-action-type令。

```
aws codepipeline get-action-type --cli-input-json file://file.json
```

此指令會傳回動作類型的動作定義輸出。此範例顯示使用 Lambda 整合模型建立的動作類型。

```
{
    "actionType": {
        "executor": {
            "configuration": {
                "lambdaExecutorConfiguration": {
                    "lambdaFunctionArn": "arn:aws:lambda:us-west-2:<account-
id>:function:my-function"
                }
```

```
            },
            "type": "Lambda"
        },
        "id": {
            "category": "Test",
            "owner": "ThirdParty",
            "provider": "TestProvider",
            "version": "1"
        },
        "inputArtifactDetails": {
            "minimumCount": 0,
            "maximumCount": 1
        },
        "outputArtifactDetails": {
            "minimumCount": 0,
            "maximumCount": 1
        },
        "permissions": {
            "allowedAccounts": [
                "<account-id>"
            ]
        },
        "properties": []
    }
}
```

更新動作類型

您可以使用 CLI 編輯使用整合模型建立的動作類型。

對於公用動作類型，您無法更新擁有者、無法將選用屬性變更為必要屬性，而且只能新增新的選用屬性。

1. 使用`get-action-type`指令取得動作類型的結構。複製結構。
2. 創建一個輸入 JSON 文件並命名它`action.json`。將您在上一個步驟中複製的動作類型結構貼到其中。更新任何您要變更的參數。您也可以新增選用的參數。

   如需輸入檔案的參數的詳細資訊，請參閱中的動作定義檔案的描述步驟 2：建立動作類型定義檔案。

   下範例示範如何更新使用 Lambda 整合模型建立的動作類型。本範例進行下列變更：

- 將provider名稱變更為TestProvider1。
- 新增工作逾時限制為 900 秒。
- 新增名為的動作組態屬性Host，該屬性會使用動作顯示給客戶。

```
{
    "actionType": {
        "executor": {
            "configuration": {
                "lambdaExecutorConfiguration": {
                    "lambdaFunctionArn": "arn:aws:lambda:us-west-2:<account-
id>:function:my-function"
                }
            },
            "type": "Lambda",
            "jobTimeout": 900
        },
        "id": {
            "category": "Test",
            "owner": "ThirdParty",
            "provider": "TestProvider1",
            "version": "1"
        },
        "inputArtifactDetails": {
            "minimumCount": 0,
            "maximumCount": 1
        },
        "outputArtifactDetails": {
            "minimumCount": 0,
            "maximumCount": 1
        },
        "permissions": {
            "allowedAccounts": [
                "account-id"
            ]
        },
        "properties": {
         "description": "Owned build action parameter description",
         "optional": true,
         "noEcho": false,
         "key": true,
         "queryable": false,
         "name": "Host"
```

```
            }
        }
}
```

3. 在終端機或命令列中執行命update-action-type令

```
aws codepipeline update-action-type --cli-input-json file://action.json
```

此指令會傳回動作類型輸出，以符合您更新的參數。

在 CodePipeline 中建立和新增自訂動作

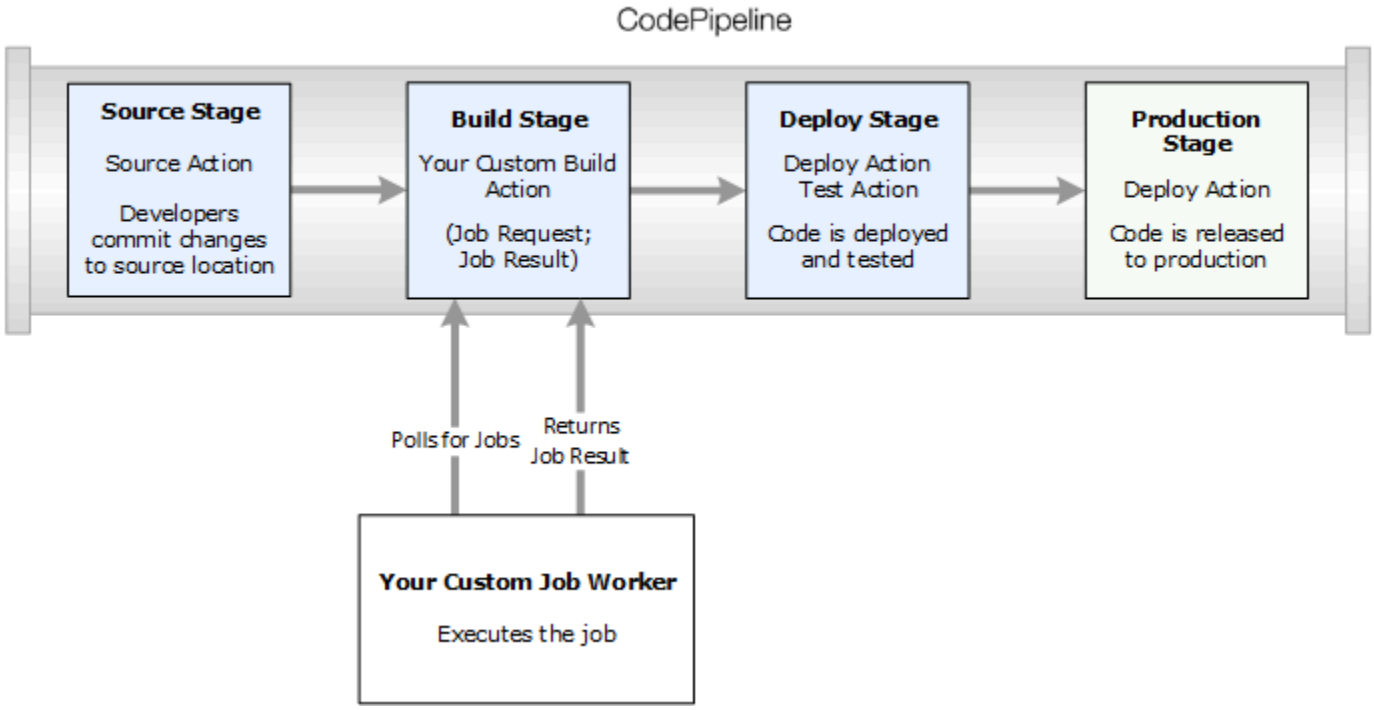
AWS CodePipeline 包含數個動作，可協助您為自動化發行程序設定建置、測試和部署資源。如果您的發行程序包含預設動作中未包含的活動 (例如內部開發的建置程序或測試套件)，您可以建立該用途的自訂動作，並將其包含在管道中。您可以使用 AWS CLI，以在與 AWS 帳戶建立關聯的管道中建立自訂動作。

您可以為下列 AWS CodePipeline 動作類別建立自訂動作：

- 建置或轉換項目的自訂建置動作
- 將項目部署至一或多個伺服器、網站或儲存庫的自訂部署動作
- 設定和執行自動化測試的自訂測試動作
- 執行函數的自訂呼叫動作

當您建立自訂動作時，您也必須建立工作 Worker，以輪詢CodePipeline此自訂動作的工作要求、執行工作，並將狀態結果傳回至CodePipeline。此任務工作者可以位於任何電腦或資源上，只要它可以存取 CodePipeline 的公有端點。若要輕鬆管理存取和安全性，請考慮在 Amazon EC2 執行個體上託管工作者。

下圖顯示管道的高階檢視，其中包含自訂建置動作：



當管線包含自訂動作做為階段的一部分時，管線會建立工作要求。自訂任務工作者偵測到該請求，並執行該任務 (在此範例中，為使用第三方建置軟體的自訂程序)。動作完成時，任務工作者會傳回成功結果或失敗結果。如果收到成功結果，管線會將修訂版本及其成品提供給下一個動作。如果傳回失敗，管線將不會為管線中的下一個動作提供修訂版本。

> **Note**
>
> 這些說明假設您已完成開始使用 CodePipeline 中的步驟。

主題

- 建立自訂動作
- 建立自訂動作的任務工作者
- 將自訂動作新增至管道

建立自訂動作

若要使用 AWS CLI 建立自訂動作

1. 開啟文字編輯器，並為您的自訂動作建立 JSON 檔案，其中包含動作類別、動作提供者，以及自訂動作所需的任何設定。例如，若要建立只需要一個屬性的自訂建置動作，您的 JSON 檔案可能如下所示：

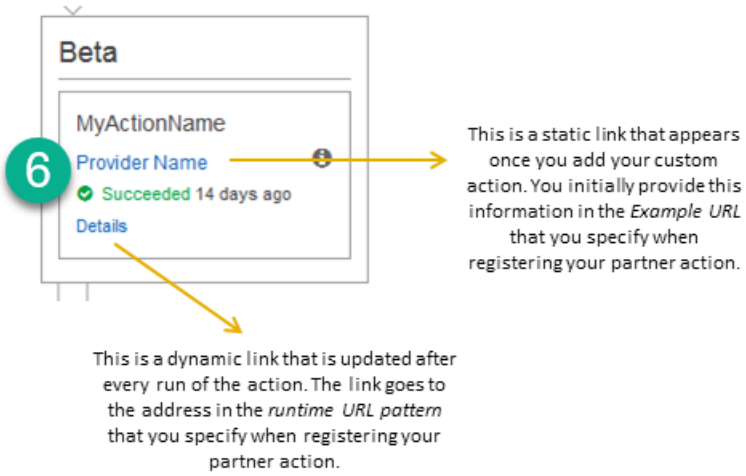
```
{
    "category": "Build",
    "provider": "My-Build-Provider-Name",
    "version": "1",
    "settings": {
        "entityUrlTemplate": "https://my-build-instance/job/{Config:ProjectName}/",
        "executionUrlTemplate": "https://my-build-instance/job/
{Config:ProjectName}/lastSuccessfulBuild/{ExternalExecutionId}/"
    },
    "configurationProperties": [{
        "name": "ProjectName",
        "required": true,
        "key": true,
        "secret": false,
        "queryable": false,
        "description": "The name of the build project must be provided when this
 action is added to the pipeline.",
        "type": "String"
    }],
    "inputArtifactDetails": {
        "maximumCount": integer,
        "minimumCount": integer
    },
    "outputArtifactDetails": {
        "maximumCount": integer,
        "minimumCount": integer
    },
    "tags": [{
      "key": "Project",
      "value": "ProjectA"
    }]
}
```

此範例透過在自訂動作上包含 Project 標籤索引鍵和 ProjectA 值，將標籤新增到自訂動作。如需有關標記 CodePipeline 資源的詳細資訊，請參閱 標記 資源。

JSON 檔案中包含兩個屬性：entityUrlTemplate 和 executionUrlTemplate。您可以遵循 {Config:*name*} 格式來參照 URL 範本內自訂動作組態屬性中的名稱，只要組態屬性同時為必要且不是秘密。例如，在上述範例中，entityUrlTemplate值參照組態屬性*ProjectName*。

- entityUrlTemplate：靜態連結，可提供動作服務提供者的相關資訊。在此範例中，建置系統包含每個建置專案的靜態連結。此連結格式會根據建置提供者而不同 (或者，如果您建立不同的動作類型 (例如測試)，則為其他服務提供者)。您必須提供此連結格式，在新增自訂動作時，使用者可以選擇此連結，以將瀏覽器開啟到您網站上提供建置專案 (或測試環境) 特性的頁面。
- executionUrlTemplate：動態連結，可使用目前或最近執行動作的相關資訊予以更新。當您的自訂任務工作者更新任務狀態 (例如，成功、失敗或進行中) 時，也會提供將用來完成連結的 externalExecutionId。此連結可以用來提供動作執行的詳細資訊。

例如，當您在管道中檢視動作時，請參閱下列兩個連結：



1

在您新增自訂動作並指向 entityUrlTemplate 中的地址 (於建立自訂動作時所指定) 之後，會出現此靜態連結。

2

在每次執行動作並指向 `executionUrlTemplate` 中的地址 (於建立自訂動作時所指定) 之後，會更新動態連結。

如需這些連結類型的詳細資訊，以及`RevisionURLTemplate`和`ThirdPartyURL`，請參閱《CodePipelineAPI 參考》CreateCustomActionType中的ActionTypeSettings和。如需動作結構需求以及如何建立動作的詳細資訊，請參閱 CodePipeline 配管結構參照。

2. 保存 JSON 文件並為其命名，您可以輕鬆記住（例如 *MyCustomAction*.json）。
3. 在已安裝 AWS CLI 的電腦上，開啟終端機工作階段 (Linux、OS X、Unix) 或命令提示字元 (Windows)。
4. 使用 AWS CLI 執行 aws codepipeline create-custom-action-type 命令，並指定您剛建立的 .JSON 檔案名稱。

   例如，若要建立組建自訂動作：

   > **Important**
   >
   > 請確認在檔案名稱之前包含 `file://`。這是此命令必要項目。

   ```
   aws codepipeline create-custom-action-type --cli-input-json
    file://MyCustomAction.json
   ```

5. 此命令會傳回您所建立自訂動作的整個結構，以及為您新增的 `JobList` 動作組態屬性。當您將自訂動作新增至管道時，可以使用 `JobList` 指定提供者中您可以輪詢任務的專案。如果您未設定此項目，則會在自訂任務工作者輪詢任務時傳回所有可用的任務。

   例如，上述命令可能會傳回與下列類似的結構：

   ```
   {
       "actionType": {
           "inputArtifactDetails": {
               "maximumCount": 1,
               "minimumCount": 1
          },
          "actionConfigurationProperties": [
               {
   ```

```
                "secret": false,
                "required": true,
                "name": "ProjectName",
                "key": true,
                "description": "The name of the build project must be provided when
 this action is added to the pipeline."
            }
        ],
        "outputArtifactDetails": {
            "maximumCount": 0,
            "minimumCount": 0
        },
        "id": {
            "category": "Build",
            "owner": "Custom",
            "version": "1",
            "provider": "My-Build-Provider-Name"
        },
        "settings": {
            "entityUrlTemplate": "https://my-build-instance/job/
{Config:ProjectName}/",
            "executionUrlTemplate": "https://my-build-instance/job/mybuildjob/
lastSuccessfulBuild/{ExternalExecutionId}/"
        }
    }
}
```

> **Note**
>
> 作為create-custom-action-type命令輸出的一部分，該id部分包括"owner": "Custom"。CodePipeline會自動指派Custom為自訂動作類型的擁有者。當您使用 create-custom-action-type 命令或 update-pipeline 命令時，無法指派或變更此值。

建立自訂動作的任務工作者

自訂動作需要工作者，該工作者將輪詢CodePipeline自訂動作的工作要求、執行工作並將狀態結果傳回至CodePipeline。任務工作者只要可以存取 CodePipeline 的公有端點，就可以位於任何電腦或資源上。

有多種方式可以設計任務工作者。下列各節提供一些用於開發 CodePipeline 自訂任務工作者的實用指導。

主題

- 選擇和設定任務工作者的許可管理策略
- 開發自訂動作的任務工作者
- 自訂任務工作者架構和範例

選擇和設定任務工作者的許可管理策略

若要開發 CodePipeline 中自訂動作的自訂任務工作者，您需要策略來整合使用者與許可管理。

最簡單的策略是建立具有 IAM 執行個體角色的 Amazon EC2 執行個體，以新增自訂任務工作者所需的基礎設施，讓您輕鬆擴展整合所需的資源。您可以使用與 AWS 的內建整合，以簡化自訂任務工作者與 CodePipeline 之間的互動。

設定 Amazon EC2 執行個執行個執行個

1. 進一步了解 Amazon EC2，並判斷它是否是您整合的正確選擇。如需詳細資訊，請參閱 Amazon EC2-虛擬伺服器託管。
2. 開始建立您的 Amazon EC2 執行個體。如需相關資訊，請參閱 Amazon EC2 執行個執行個執行個執行個

另一個需要考慮的策略是使用與 IAM 聯合身分識別來整合您現有的身分識別提供者系統和資源。如果您已有公司身分提供者，或已設定為使用 Web 身分提供者來支援使用者，則此策略特別有用。聯合身分可讓您授予AWS資源的安全存取權CodePipeline，包括無需建立或管理 IAM 使用者。您可以使用功能和政策以符合密碼安全要求和輪換登入資料。您可以使用範例應用程式做為您自己設計的範本。

設定聯合身分

1. 進一步了解 IAM 身分。如需資訊，請參閱管理聯合。
2. 檢閱授予臨時存取藍本中的範例，以識別最符合您自訂動作需求的臨時存取藍本。
3. 檢閱與基礎設施相關的聯合身分程式碼範例，例如：
   - 適用於 Active Directory 使用案例的聯合身分範例應用程式
4. 開始設定聯合身分。如需詳細資訊，請參閱 IAM 使用者指南中的身分識別提供者和聯

運行自定義操作和工作線程AWS 帳戶時，請執行以下其中一項以在您的下面使用。

若使用者想要與 AWS Management Console 之外的 AWS 互動，則需要程式設計存取權。授予程式設計存取權的方式取決於存取 AWS 的使用者類型。

若要授予使用者程式設計存取權，請選擇下列其中一個選項。

哪個使用者需要程式設計存取權？	若要	By
人力身分 (IAM Identity Center 中管理的使用者)	使用臨時憑證簽署對 AWS CLI、AWS SDK 或 AWS API 的程式請求。	請依照您要使用的介面所提供的指示操作。 • 關於 AWS CLI，請參閱AWS Command Line Interface使用者指南中的設定 AWS CLI 來使用 AWS IAM Identity Center。 • 關於 AWS SDK、工具和 AWS API，請參閱AWS《SDK 和工具參考指南》中的 IAM Identity Center 驗證。
IAM	使用臨時憑證簽署對 AWS CLI、AWS SDK 或 AWS API 的程式請求。	請遵循《IAM 使用者指南》中使用臨時憑證搭配 AWS 資源中的指示。
IAM	(不建議使用) 使用長期憑證簽署 AWS CLI 或 AWS API 的程式設計要求 (直接或使用 AWS SDK)。	請依照您要使用的介面所提供的指示操作。 • 如需相關資訊AWS CLI，請參閱《使用者指南》中的使用 IAM 使用者登入AWS Command Line Interface資料進行驗證。 • 關於AWS SDK 和工具，請參閱AWS《SDK 和工具參考

哪個使用者需要程式設計存取權？	若要	By
		指南》中的使用長期憑證進行驗證。 • 對於 AWS API，請參閱《IAM 使用者指南》中的管理 IAM 使用者的存取金鑰。

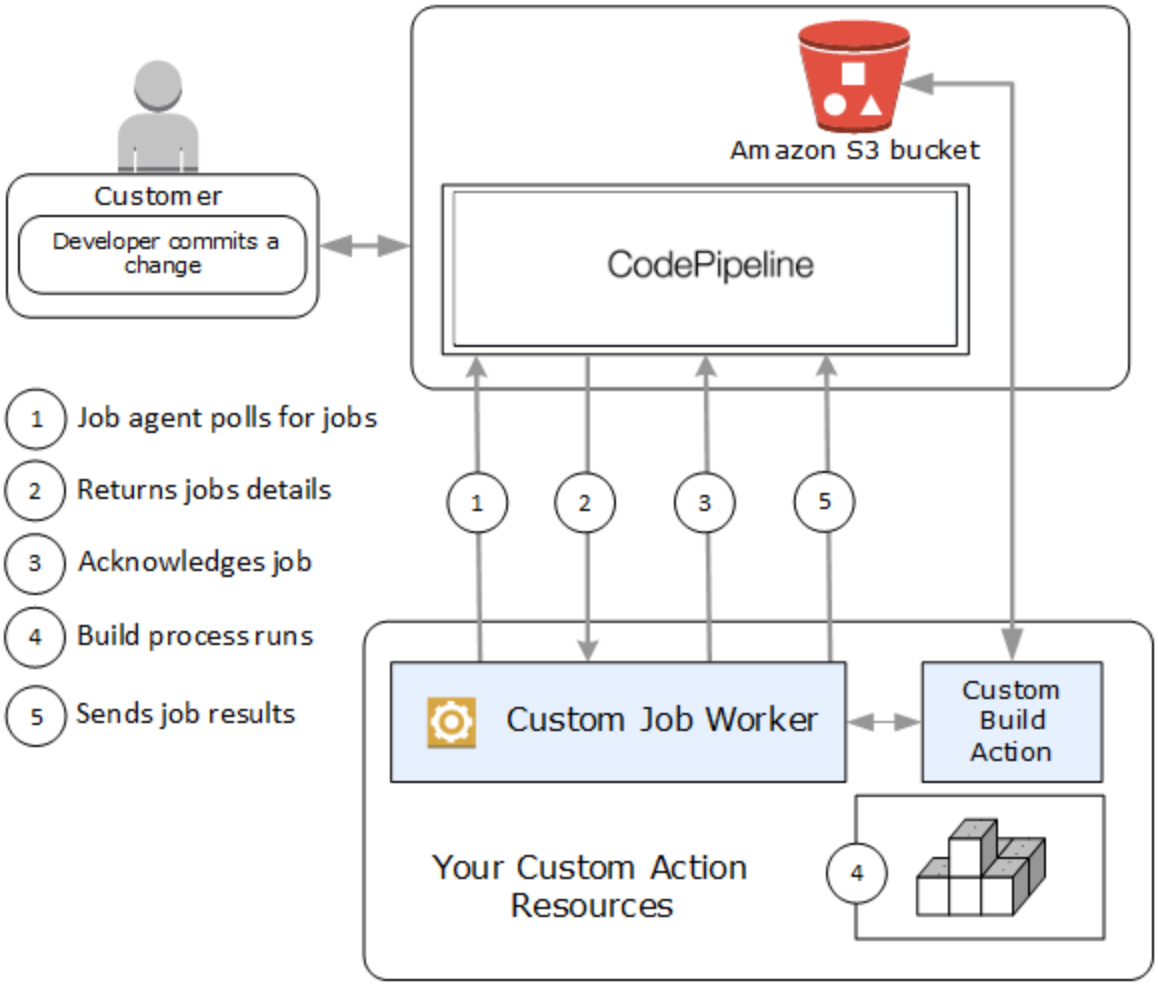
您可以建立下列範例政策，以與自訂任務工作者搭配使用。此政策僅做為範例使用，並以現狀提供。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codepipeline:PollForJobs",
        "codepipeline:AcknowledgeJob",
        "codepipeline:GetJobDetails",
        "codepipeline:PutJobSuccessResult",
        "codepipeline:PutJobFailureResult"
      ],
      "Resource": [
        "arn:aws:codepipeline:us-east-2::actionType:custom/Build/MyBuildProject/1/"
      ]
    }
  ]
}
```

> **Note**
>
> 請考慮使用受AWSCodePipelineCustomActionAccess管理的原則。

開發自訂動作的任務工作者

在您選擇許可管理策略之後，應該考慮任務工作者與 CodePipeline 的互動方式。下列高階圖表顯示建置程序的自訂動作和工作 Worker 的工作流程。



1. 您的任務工作者會使用 `PollForJobs` 輪詢 CodePipeline 是否有任務。
2. 管道來源階段中的變更觸發管道時 (例如，開發人員確認變更時)，即會開始自動化發行程序。程序會繼續進行，直到設定自訂動作的階段為止。當它到達您在此階段中的動作時，CodePipeline 會將任務排入佇列中。如果您的任務工作者再次呼叫 `PollForJobs` 以取得狀態，則會顯示此任務。從 `PollForJobs` 取得任務詳細資訊，並將它傳遞回任務工作者。
3. 工作人員打電話`AcknowledgeJob`給CodePipeline發送工作確認。CodePipeline傳回確認，指出工作 Worker 應該繼續工作 (`InProgress`)，或者，如果您有一個以上的工作 Worker 輪詢工作，而另

一個工作 Worker 已宣告該工作，則會傳回錯InvalidNonceException誤回應。在 InProgress 認可之後，CodePipeline 會等待傳回結果。

4. 工作 Worker 會在修訂版本上初始化您的自訂動作，然後執行您的動作。與任何其他動作一起，您的自訂動作會將結果傳回給工作 Worker。在建立自訂動作的範例中，動作會從 Amazon S3 儲存貯體提取成品、建立成品，然後將成功建置的成品推送回 Amazon S3 儲存貯體。
5. 執行動作時，工作工作者可以使PutJobSuccessResult用延續權杖 (工作者產生的作業狀態的序列化，例如 JSON 格式的建置識別碼或 Amazon S3 物件金鑰) 呼叫，以及將用於填入連結的ExternalExecutionId資訊executionUrlTemplate。這將使用指向特定操作詳細信息的工作鏈接更新管道的控制台視圖。雖然不需要，但為最佳實務，因為它可讓使用者檢視執行中自訂動作的狀態。

   呼叫 PutJobSuccessResult 之後，會將任務視為完成。在 CodePipeline 中建立包含接續字符的新任務。如果您的任務工作者再次呼叫 PollForJobs，則會顯示此任務。新的任務可以用來檢查動作的狀態，並以接續字符傳回，或在動作完成之後以不含接續字符傳回。

   > **Note**
   >
   > 如果您的任務工作者執行所有適用於自訂動作的工作，則應該考慮將您的任務工作者處理分為至少兩個步驟。第一個步驟會建立您動作的詳細資訊頁面。在您建立詳細資訊頁面之後，即可序列化任務工作者的狀態，並根據大小限制將它傳回為接續字符 (請參閱 AWS CodePipeline 中的配額)。例如，您可以將動作的狀態寫入用來做為接續字符的字串。任務工作者處理的第二個步驟 (和後續步驟) 會執行動作的實際工作。最後一個步驟會將成功或失敗傳回給 CodePipeline，但最後一個步驟上沒有接續字符。

   如需有關使用延續權杖的詳細資訊，請參閱 CodePipelineAPI 參考PutJobSuccessResult中的規格。
6. 自訂動作完成後，工作 Worker 會呼叫下列其中一個 API，將CodePipeline自訂動作的結果傳回至：
   - PutJobSuccessResult沒有延續令牌，這表示自定義操作已成功運行
   - PutJobFailureResult，表示自訂動作未成功執行

   根據結果，管道會繼續進行下一個動作 (成功) 或停止 (失敗)。

自訂任務工作者架構和範例

在您映射高階工作流程之後，即可建立任務工作者。雖然您自訂動作的特性最終會判斷您任務工作者所需的內容，但是自訂動作的大部分任務工作者都會包含下列功能：

- 使用 `PollForJobs` 從 CodePipeline 輪詢任務。
- 使用 `AcknowledgeJob`、`PutJobSuccessResult` 和 `PutJobFailureResult`，來認可任務並將結果傳回給 CodePipeline。
- 從管道擷取成品和/或將成品放入 Amazon S3 儲存貯體。若要從 Amazon S3 儲存貯體下載成品，您必須建立一個使用簽名版本 4 簽署 (Sig V4) 的 Amazon S3 用戶端。簽名 V4 是必需的AWS KMS.

  若要將成品上傳到 Amazon S3 儲存貯體，您必須另外設定 Amazon S3`PutObject` 請求以使用加密。目前僅支援AWS金鑰管理服務 (AWS KMS) 進行加密。 AWS KMS用途AWS KMS keys。為了知道是否使用AWS 受管金鑰或客戶管理的金鑰來上傳人工因素，您的自訂工作者必須查看工作資料並檢查加密金鑰屬性。如果已設定屬性，您應在設定時使用該客戶管理的金鑰 IDAWS KMS。如果索引鍵屬性為 null，您可以使用AWS 受管金鑰. CodePipeline除非另有配置，AWS 受管金鑰否則會使用。

  如需示範如何在 Java 或 .NET 中建立AWS KMS參數的範例，請參閱使用AWS開發套件AWS Key Management Service在 Amazon S3 中指定參數。如需有關的 Amazon S3 儲存貯體的詳細資訊 CodePipeline，請參閱CodePipeline 概念 。

在上提供更複雜的自訂工作 Worker 範例GitHub。此範例是開放原始碼，並以現狀提供。

- Job 背景工作者範例CodePipeline：從GitHub儲存庫下載範例。

將自訂動作新增至管道

擁有工作背景工作者之後，您可以透過建立新動作並在使用「建立管線」精靈時選擇自訂動作、編輯現有管道並新增自訂動作，或使用 SDK 或 APIAWS CLI，將自訂動作新增至管道。

> **Note**
>
> 如果自訂動作是建置或部署動作，則您可以在 Create Pipeline (建立管道) 精靈中建立包含該自訂動作的管道。如果您的自訂動作是在測試類別中，則您必須編輯現有管道予以新增。

主題

- 將自訂動作新增至現有管道 (CLI)

將自訂動作新增至現有管道 (CLI)

您可以使用AWS CLI將自訂動作新增至現有配管。

1. 開啟終端機工作 (Linux、macOS 或 Unix) 或命令提示 (Windows)，然後執行get-pipeline命令，然後執行命令，然後將要編輯的管線結構。例如，針對名為 **MyFirstPipeline** 的管道，您會輸入下列命令：

   ```
   aws codepipeline get-pipeline --name MyFirstPipeline >pipeline.json
   ```

   此命令不會傳回任何內容，但您建立的檔案應該會顯示在您執行命令的目錄中。
2. 使用任何文字編輯器開啟 JSON 檔案，並修改檔案的結構，以將自訂動作新增至現有階段。

   > **Note**
   > 如果您想要您的動作與該階段中的另一個動作平行執行，則請確保您將與該動作相同的 `runOrder` 值指派給它。

   例如，若要修改管道的結構以新增名為 Build 的階段，並將建置自訂動作新增至該階段，您可以先修改 JSON 以新增 Build 階段，再新增部署階段，如下所示：

   ```
   ,
       {
         "name": "MyBuildStage",
         "actions":  [
                 {
                   "inputArtifacts": [
                   {
                      "name": "MyApp"
                    }
                      ],
                       "name": "MyBuildCustomAction",
                       "actionTypeId": {
                           "category": "Build",
                           "owner": "Custom",
                           "version": "1",
                           "provider": "My-Build-Provider-Name"
   ```

```
                    },
                    "outputArtifacts": [
                        {
                          "name": "MyBuiltApp"
                        }
                    ],
                    "configuration": {
                        "ProjectName": "MyBuildProject"
                    },
                    "runOrder": 1
                }
            ]
        },
        {
            "name": "Staging",
            "actions": [
                    {
                        "inputArtifacts": [
                            {
                                "name": "MyBuiltApp"
                            }
                        ],
                        "name": "Deploy-CodeDeploy-Application",
                        "actionTypeId": {
                            "category": "Deploy",
                            "owner": "AWS",
                            "version": "1",
                            "provider": "CodeDeploy"
                        },
                        "outputArtifacts": [],
                        "configuration": {
                            "ApplicationName": "CodePipelineDemoApplication",
                            "DeploymentGroupName": "CodePipelineDemoFleet"
                        },
                        "runOrder": 1
                    }
                ]
             }
    ]
}
```

3. 若要套用您的變更，請執行 update-pipeline 命令，並指定管道 JSON 檔案，與下面類似：

> **Important**
>
> 請確認在檔案名稱之前包含 `file://`。這是此命令必要項目。

```
aws codepipeline update-pipeline --cli-input-json file://pipeline.json
```

此命令會傳回所編輯管道的整個結構。

4. 開啟 CodePipeline 主控台，並選擇您剛編輯的管道名稱。

此管道會顯示您的變更。下次您變更來源位置時，管道會透過修訂過的管道結構來執行該修訂版。

在 CodePipeline 中標記自訂動作

標籤是與 AWS 資源關聯的金鑰值對。您可以在 CodePipeline 中使用主控台或 CLI 將標籤套用到您的自訂動作。如需 CodePipeline 資源標記、使用案例、標籤索鍵和值限制條件的相關資訊，以及支援的資源類型，請參標記 資源。

您可以新增、刪除和更新自訂動作中的標籤值。您可以在每個自訂動作中新增最多 50 個標籤。

主題

- 新增標籤到自訂動作
- 檢視自訂動作的標籤
- 編輯自訂動作的標籤
- 從自訂動作移除標籤

新增標籤到自訂動作

依照以下步驟，使用 AWS CLI 將標籤新增到自訂動作。若要在建立自訂動作時，將標籤新增到自訂動作，請參閱 在 CodePipeline 中建立和新增自訂動作。

在這些步驟中，我們假設您已經安裝新版 AWS CLI 或更新到最新版本。如需詳細資訊，請參閱安裝 AWS Command Line Interface。

在終端機或命令列上執行 tag-resource 命令，為您要新增標籤及該標籤之索引鍵和值的自訂動作，指定 Amazon Resource Name (ARN)。您可以將多個標籤新增到自訂動作。例如，若要以兩個標籤來標

記自訂動作，一個標籤的索引鍵名為 *TestActionType*，標籤值為 *UnitTest*，另一個標籤的索引鍵名為 *ApplicationName*，標籤值為 *MyApplication*：

```
aws codepipeline tag-resource --resource-arn arn:aws:codepipeline:us-west-2:account-id:actiontype:Owner/Category/Provider/Version --tags key=TestActionType,value=UnitTest
 key=ApplicationName,value=MyApplication
```

如果成功，此命令不會傳回任何內容。

檢視自訂動作的標籤

依照以下步驟使用 AWS CLI 查看自訂動作的 AWS 標籤。如果沒有新增標籤，將會傳回空的清單。

在終端機或命令列上執行 list-tags-for-resource 命令。例如，以 ARN arn:aws:codepipeline:*us-west-2*:*account-id*:actiontype:*Owner*/*Category*/*Provider*/*Version* 檢視自訂動作的標籤索引鍵鍵和標籤值的清單：

```
aws codepipeline list-tags-for-resource --resource-arn arn:aws:codepipeline:us-west-2:account-id:actiontype:Owner/Category/Provider/Version
```

如果成功，此命令會傳回類似如下的資訊：

```
{
    "tags": {
        "TestActionType": "UnitTest",
        "ApplicationName": "MyApplication"
    }
}
```

編輯自訂動作的標籤

依照以下步驟使用 AWS CLI 編輯自訂動作的標籤。您可以變更現有索引鍵的值或新增其他索引鍵。您也可以從自訂動作中移除標籤，如下個部分所示。

在終端機或命令列，執行 tag-resource 命令，為您要更新標籤並指定其標籤索引鍵和標籤值的自訂動作，指定 Amazon Resource Name (ARN)：

```
aws codepipeline tag-resource --resource-arn arn:aws:codepipeline:us-west-2:account-id:actiontype:Owner/Category/Provider/Version --tags
 key=TestActionType,value=IntegrationTest
```

從自訂動作移除標籤

依照以下步驟使用 AWS CLI 從自訂動作中移除標籤。當您從關聯的資源移除標籤時，將會刪除這些標籤。

> **Note**
>
> 如果您刪除自訂動作，所有標籤關聯都會從已刪除的自訂動作中移除。您不需要在刪除自訂動作之前移除標籤。

在終端機或命令列，執行 untag-resource 命令，為您要移除的標籤及其標籤索引鍵的自訂動作指定 ARN。例如，移除具有標籤索引鍵 *TestActionType* 的自訂動作的標籤：

```
aws codepipeline untag-resource --resource-arn arn:aws:codepipeline:us-west-2:account-id:actiontype:Owner/Category/Provider/Version --tag-keys TestActionType
```

如果成功，此命令不會傳回任何內容。為了確認與自訂動作關聯的標籤，請執行 list-tags-for-resource 命令。

在 CodePipeline 中於管道中呼叫 AWS Lambda 函數

AWS Lambda 是一項運算服務，可讓您執行程式碼，無需佈建或管理伺服器。您可以建立 Lambda 函數，並將其新增為管道中的動作。由於 Lambda 可讓您撰寫函數以執行幾乎任何工作，因此您可以自訂管道的運作方式。

> **Important**
>
> 請勿記錄CodePipeline傳送至 Lambda 的 JSON 事件，因為這可能會導致使用者認證登入CloudWatch記錄。該CodePipeline角色使用 JSON 事件將臨時登入資料傳遞給`artifactCredentials`欄位中的 Lambda。如需範例事件，請參閱JSON 事件範例。

以下是可以在管道中使用 Lambda 函數的一些方法：

- 使用 AWS CloudFormation 在管道的某個階段中隨需建立資源，並在另一個階段中予以刪除。
- 使用交換 CNAME 值的 Lambda 函數來部署零停機時間的應用程式版本。AWS Elastic Beanstalk

- 若要部署到亞馬遜 ECS 泊塢視窗執行個體。
- 建立 AMI 快照以在建置或部署之前備份資源。
- 在管道新增與第三方產品的整合，例如發布訊息到 IRC 用戶端。

> **Note**
>
> 建立和執行 Lambda 函數可能會向您的AWS帳戶產生費用。如需詳細資訊，請參閱 定價。

本主題假設您熟悉AWS CodePipeline並AWS Lambda且知道如何建立管道、函數，以及它們所依賴的 IAM 政策和角色。本主題將說明如何：

- 建立 Lambda 函數來測試網頁是否已成功部署。
- 設定CodePipeline和 Lambda 執行角色，以及在管道中執行函數所需的權限。
- 編輯管道以將 Lambda 函數新增為動作。
- 手動釋出變更以測試動作。

> **Note**
>
> 在中使用跨區域 Lambda 叫用動作時CodePipeline，使用PutJobSuccessResult和的 lambda 執行狀態PutJobFailureResult應傳送至存在 Lambda 函數的AWS區域，而不是傳送至 CodePipeline存在 Lambda 函數的區域。

本主題包含範例函數，以示範在以下位置使用 Lambda 函數的彈性CodePipeline：

- Basic Lambda function
  - 建立要搭配使用的基本 Lambda 函數CodePipeline。
  - 將成功或失敗結果傳回動作CodePipeline的「詳細資訊」連結中。
- 使用 AWS CloudFormation 範本的 Python 函數範例
  - 使用 JSON 編碼使用者參數來傳遞多個組態值到函數 (`get_user_params`)。
  - 與成品儲存貯體中的 .zip 成品互動 (`get_template`)。
  - 使用連續字符來監控長時間執行的非同步處理 (`continue_job_later`)。這允許動作繼續，並且即使函數超過 15 分鐘的執行階段 (Lambda 中的限制) 也會成功。

每個範例函數皆包含關於您必須新增到角色的許可資訊。如需中限制的相關資訊AWS Lambda，請參閱AWS Lambda開發人員指南中的限制。

> **Important**
>
> 在此主題中的範本程式碼、角色與政策皆僅做為範例使用，並依原樣提供。

主題

步驟 1：建立管道

在此步驟中，將會建立管線來稍後將會建立 Lambda 函數。此管道與您在 CodePipeline 教程 中所建立的管道相同。如果該管道仍為您的帳戶設定，且位於您打算建立 Lambda 函數的相同區域中，則可以略過此步驟。

建立管道

1. 請按照中的前三個步驟教學：建立簡易管道 (S3 儲存貯體)來建立 Amazon S3 儲存貯體、CodeDeploy資源和兩階段管道。為您的執行個體類型選擇亞馬遜 Linux 選項。您可以使用任何您想要的管道名稱，但是在此主題中的步驟使用 MyLambdaTestPipeline。
2. 在管道的狀態頁面上的 CodeDeploy 動作中，請選擇 Details (詳細資訊)。請在部署群組的部署詳細資訊頁面上，從清單中選擇一個執行個體 ID。
3. 在 Amazon EC2 主控台的執行個體的 [詳細資料] 索引標籤上，將 IP 位址複製到公用 IPv4 地址 (例如**192.0.2.4**)。您會使用此地址做為 AWS Lambda 中函數的目標。

> **Note**
>
> 的預設服務角色原則CodePipeline包含叫用函數所需的 Lambda 許可。但是，若您已修改預設服務角色或選取不同角色，請確認該角色的政策允許 `lambda:InvokeFunction` 與 `lambda:ListFunctions` 許可。否則，包含 Lambda 動作的管道會失敗。

步驟 2：建立 Lambda 函數

在此步驟中，將會建立 Lambda 函數來發出 HTTP 請求並檢查網頁上的一行文字。在此步驟中，您還必須建立 IAM 政策和 Lambda 執行角色和 Lambda 執行角色。如需詳細資訊，請參閱AWS Lambda開發人員指南中的權限模型。

建立執行角色

1. 登入 AWS Management Console，並開啟位於 https://console.aws.amazon.com/iam/ 的 IAM 主控台。
2. 選擇 Policies (政策)，然後選擇 Create Policy (建立政策)。選擇 JSON (JSON) 索引標籤，然後將下列政策貼到欄位中。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "logs:*"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:logs:*:*:*"
    },
    {
      "Action": [
        "codepipeline:PutJobSuccessResult",
        "codepipeline:PutJobFailureResult"
        ],
        "Effect": "Allow",
        "Resource": "*"
     }
  ]
}
```

3. 選擇 Review policy (檢閱政策)。
4. 在 Review policy (檢閱政策) 頁面上的 Name (名稱) 中，輸入政策名稱 (例如 **`CodePipelineLambdaExecPolicy`**)。在 Description (說明) 中，輸入 **`Enables Lambda to execute code`**。

   選擇 建立政策 。

   > **Note**
   >
   > 這些是 Lambda 函數與CodePipeline亞馬遜互操作所需的最低許可CloudWatch。如果您想要擴充此原則以允許函數與其他AWS資源互動，則應修改此政策以允許這些 Lambda 函數所需的動作。

5. 在政策儀表板頁面上，選擇 Roles (角色)，然後選擇 Create role (建立角色)。
6. 在 [建立角色] 頁面上，選擇AWS 服務。選擇 Lambda (Lambda)，然後選擇 Next: Permissions (下一步：許可)。
7. 在 Attach permissions policies (附加許可政策) 頁面上，選取 CodePipelineLambdaExecPolicy 旁的核取方塊，然後選擇 Next: Tags (下一步：標籤)。選擇 下一步：檢閱 。
8. 在 Review (檢閱) 頁面上的 Role name (角色名稱) 中輸入名稱，然後選擇 Create role (建立角色)。

若要建立要搭配使用的 Lambda 函數範例CodePipeline

1. 請登入 AWS Management Console，並開啟位於 https://console.aws.amazon.com/lambda/ 的 AWS Lambda 主控台。
2. 在 Functions (函數) 頁面上，選擇 Create function (建立函數)。

   > **Note**
   >
   > 如果您看到「歡迎」頁面而非 Lambda 頁面，請選擇「立即開始使用」。

3. 在 Create function (建立函數) 頁面上，選擇 Author from scratch (從頭開始撰寫)。在函數名稱中，輸入 Lambda 函數的名稱 (例如**`MyLambdaFunctionForAWSCodePipeline`**)。在執行階段中，選擇 Node.js 14.x。

4. 在 Role (角色) 下，請選取 Choose an existing role (選擇現有的角色)。在 Existing role (現有角色) 中選擇您的角色，然後選擇 Create function (建立函數)。

   會開啟建立的函數詳細資訊頁面。

5. 複製下列程式碼到 Function code (函數程式碼) 方塊中：

> **Note**
>
> 事件物件，在CodePipeline .job 鍵下，包含工作詳細資訊。如需 JSON 事件CodePipeline 傳回至 Lambda 的完整範例，請參閱JSON 事件範例。

```
var assert = require('assert');
var AWS = require('aws-sdk');
var http = require('http');

exports.handler = function(event, context) {

    var codepipeline = new AWS.CodePipeline();

    // Retrieve the Job ID from the Lambda action
    var jobId = event["CodePipeline.job"].id;

    // Retrieve the value of UserParameters from the Lambda action configuration in
 CodePipeline, in this case a URL which will be
    // health checked by this function.
    var url =
 event["CodePipeline.job"].data.actionConfiguration.configuration.UserParameters;

    // Notify CodePipeline of a successful job
    var putJobSuccess = function(message) {
        var params = {
            jobId: jobId
        };
        codepipeline.putJobSuccessResult(params, function(err, data) {
            if(err) {
                context.fail(err);
            } else {
                context.succeed(message);
            }
        });
```

```
    };

    // Notify CodePipeline of a failed job
    var putJobFailure = function(message) {
        var params = {
            jobId: jobId,
            failureDetails: {
                message: JSON.stringify(message),
                type: 'JobFailed',
                externalExecutionId: context.awsRequestId
            }
        };
        codepipeline.putJobFailureResult(params, function(err, data) {
            context.fail(message);
        });
    };

    // Validate the URL passed in UserParameters
    if(!url || url.indexOf('http://') === -1) {
        putJobFailure('The UserParameters field must contain a valid URL address to
test, including http:// or https://');
        return;
    }

    // Helper function to make a HTTP GET request to the page.
    // The helper will test the response and succeed or fail the job accordingly
    var getPage = function(url, callback) {
        var pageObject = {
            body: '',
            statusCode: 0,
            contains: function(search) {
                return this.body.indexOf(search) > -1;
            }
        };
        http.get(url, function(response) {
            pageObject.body = '';
            pageObject.statusCode = response.statusCode;

            response.on('data', function (chunk) {
                pageObject.body += chunk;
            });

            response.on('end', function () {
                callback(pageObject);
```

```
            });

            response.resume();
        }).on('error', function(error) {
            // Fail the job if our request failed
            putJobFailure(error);
        });
    };

    getPage(url, function(returnedPage) {
        try {
            // Check if the HTTP response has a 200 status
            assert(returnedPage.statusCode === 200);
            // Check if the page contains the text "Congratulations"
            // You can change this to check for different text, or add other tests
 as required
            assert(returnedPage.contains('Congratulations'));

            // Succeed the job
            putJobSuccess("Tests passed.");
        } catch (ex) {
            // If any of the assertions failed then fail the job
            putJobFailure(ex);
        }
    });
};
```

6. 將 Handler (處理常式) 保留為預設值，並且將 Role (角色) 保留為預設值 **CodePipelineLambdaExecRole**。
7. 在 Basic settings (基本設定) 的 Timeout (逾時) 中，輸入 **20** 秒。
8. 選擇 儲存 。

步驟 3：將 Lambda 函數新增至CodePipeline主控台中的管道

在此步驟中，您將新階段新增至管線，然後新增 Lambda 動作，將函數呼叫至該階段。

新增階段

1. 請登入AWS Management Console並開啟CodePipeline主控台，網址為 http://console.aws.amazon.com/codesuite/codepipeline/home。
2. 在 Welcome (歡迎使用) 頁面上，選擇您建立的管道。

3. 在管道檢視頁面上，選擇 Edit (編輯)。
4. 在 Edit (編輯) 頁面上，選擇 + Add stage (+ 新增階段) 以在使用 CodeDeploy 動作部署階段後新增階段。輸入階段的名稱 (例如 **LambdaStage**)，然後選擇 Add stage (新增階段)。

> **Note**
>
> 您也可以選擇將 Lambda 動作新增至現有階段。為了示範目的，我們將 Lambda 函數新增為階段中唯一的動作，以便您在成品透過管道進行時輕鬆檢視其進度。

5. 選擇 + Add action group (+ 新增動作群組)。在編輯動作的動作名稱中，輸入 Lambda 動作的名稱 (例如**MyLambdaAction**)。在 Provider (提供者) 中，選擇 AWS Lambda。在函數名稱中，選擇或輸入 Lambda 函數的名稱或成名或名稱或成名或名稱或成名或成名或**MyLambdaFunctionForAWSCodePipeline** 在使用者參數中，指定先前複製之 Amazon EC2 執行個體的 IP 位址 (例如，**http://*192.0.2.4***)，然後選擇 [完成]。

> **Note**
>
> 此主題使用 IP 地址，但是在現實情況中，您可以提供已註冊的網站名稱 (例如 **http://*www.example.com***)。如需中的事件資料和處理常式的詳細資訊AWS Lambda，請參閱AWS Lambda開發人員指南中的程式設計模型。

6. 在 Edit action (編輯動作) 頁面中，選擇 Save (儲存)。

步驟 4：使用 Lambda 函數來測試管道

若要測試函數，請透過管道釋出最近一次的變更。

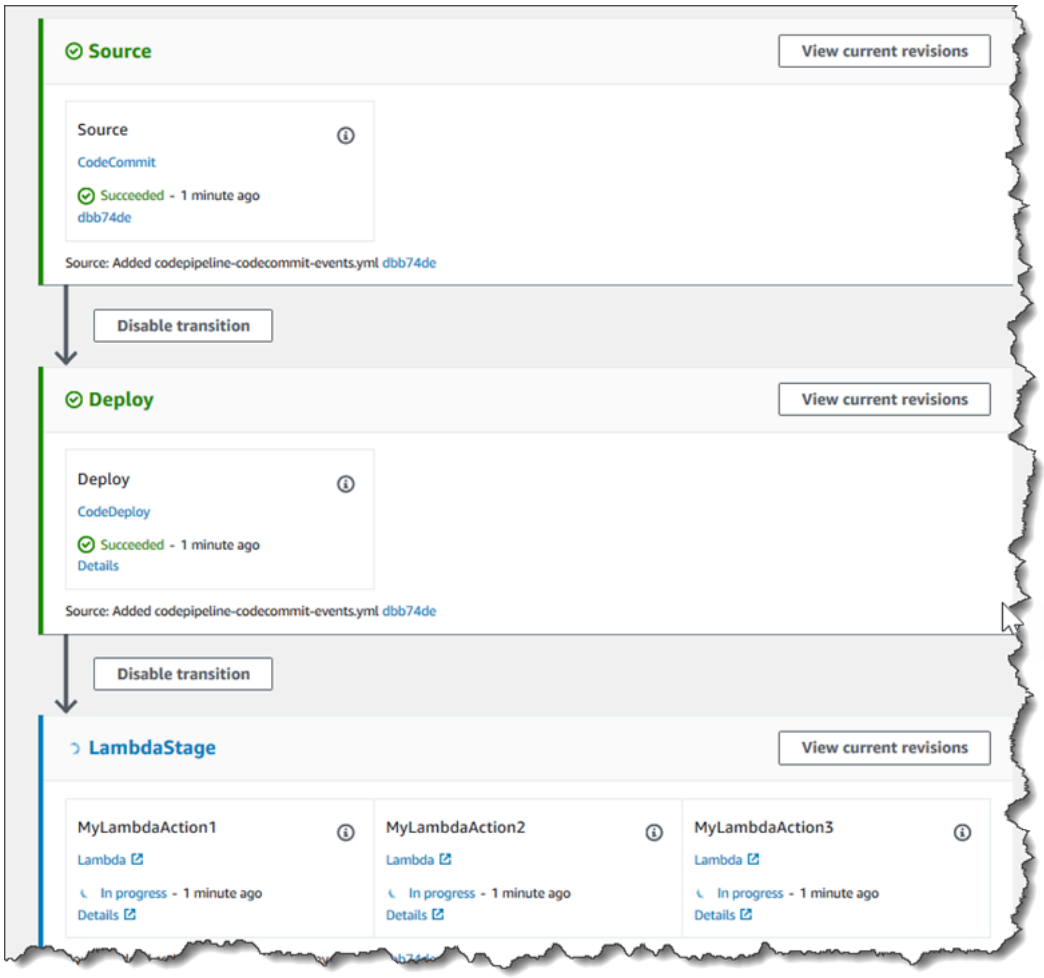
使用主控台來透過管道執行最新的成品版本

1. 在配管詳細資訊頁面上，選擇「發行變更」。這將會在管道的來源動作所指定的各個來源位置中執行最新的可用版本。
2. Lambda 動作完成後，選擇詳細資料連結以檢視 Amazon 中函數的日誌串流CloudWatch，包括事件的計費持續時間。若函數失敗，CloudWatch 日誌將提供關於原因的資訊。

步驟 5：後續步驟

現在您已成功建立 Lambda 函數，並將其新增為管線中的動作，您可以嘗試下列動作：

- 將更多 Lambda 動作新增至您的舞台，以檢查其他網頁。
- 修改 Lambda 函數以檢查其他文字字串。
- 探索 Lambda 函數，並建立自己的 Lambda 函數，並將其新增至管道。



完成 Lambda 函數的試驗後，請考慮將其從管道中移除、從中刪除AWS Lambda，然後從 IAM 刪除角色，以避免可能產生的費用。如需更多資訊，請查看 在中編輯配管 CodePipeline、在 CodePipeline 中刪除管道 以及刪除角色或執行個體描述檔。

JSON 事件範例

以下範例會顯示範例 JSON 事件來傳送至 Lambda 的範例 JSON 事件CodePipeline。此事件的結構類似於對 `GetJobDetails API` 的回應，但是不包含 `actionTypeId` 與 `pipelineContext` 資料類型。兩個動作組態詳細資訊，`FunctionName` 與 `UserParameters`，皆包含在 JSON 事件與對 `GetJobDetails` API 的回應中。使用`######`顯示的值為範例或說明，並非實際值。

```
{
    "CodePipeline.job": {
        "id": "11111111-abcd-1111-abcd-111111abcdef",
        "accountId": "111111111111",
        "data": {
            "actionConfiguration": {
                "configuration": {
                    "FunctionName": "MyLambdaFunctionForAWSCodePipeline",
                    "UserParameters": "some-input-such-as-a-URL"
                }
            },
            "inputArtifacts": [
                {
                    "location": {
                        "s3Location": {
                            "bucketName": "the name of the bucket configured as the
pipeline artifact store in Amazon S3, for example codepipeline-us-east-2-1234567890",
                            "objectKey": "the name of the application, for example
CodePipelineDemoApplication.zip"
                        },
                        "type": "S3"
                    },
                    "revision": null,
                    "name": "ArtifactName"
                }
            ],
            "outputArtifacts": [],
            "artifactCredentials": {
                "secretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
                "sessionToken": "MIICiTCCAfICCQD6m7oRw0uXOjANBgkqhkiG9w
0BAQUFADCBiDELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZ
WF0dGxlMQ8wDQYDVQQKEwZBbWF6b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIw
EAYDVQQDEwlUZXN0Q2lsYWMxHzAdBgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5
jb20wHhcNMTEwNDI1MjA0NTIxWhcNMTIwNDI0MjA0NTIxWjCBiDELMAkGA1UEBh
MCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBb
WF6b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMx
HzAdBgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5jb20wgZ8wDQYJKoZIhvcNAQE
BBQADgY0AMIGJAoGBAMaK0dn+a4GmWIWJ21uUSfwfEvySWtC2XADZ4nB+BLYgVI
k60CpiwsZ3G93vUEIO3IyNoH/f0wYK8m9TrDHudUZg3qX4waLG5M43q7Wgc/MbQ
ITxOUSQv7c7ugFFDzQGBzZswY6786m86gpEIbb3OhjZnzcvQAaRHhdlQWIMm2nr
AgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4nUhVVxYUntneD9+h8Mg9q6q+auN
KyExzyLwaxlAoo7TJHidbtS4J5iNmZgXL0FkbFFBjvSfpJIlJ00zbhNYS5f6Guo
EDmFJl0ZxBHjJnyp378OD8uTs7fLvjx79LjSTbNYiytVbZPQUQ5Yaxu2jXnimvw
```

```
 3rrszlaEXAMPLE=",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE"
            },
            "continuationToken": "A continuation token if continuing job",
            "encryptionKey": {
              "id": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
              "type": "KMS"
            }
        }
    }
}
```

其他函數範例

下列 Lambda 函數範例示範您可以在中用於管道的其他功能CodePipeline。若要使用這些函數，您可能必須修改 Lambda 執行角色的政策，如每個範例簡介中所述。

主題

- 使用 AWS CloudFormation 範本的 Python 函數範例

使用 AWS CloudFormation 範本的 Python 函數範例

以下範例顯示根據提供之 AWS CloudFormation 範本來建立或更新堆疊的函數。該範本會建立 Amazon S3 儲存貯體。僅做為示範用途，以將成本降至最低。在理想情況下，您應該在上傳任何內容到儲存貯體前刪除堆疊。若您上傳檔案到儲存貯體，將不能在刪除堆疊時刪除儲存貯體。您需要手動刪除儲存貯體中所有內容，才可刪除儲存貯體本身。

此 Python 範例假設您有一個使用 Amazon S3 儲存貯體做為來源動作的管道，或者您可以存取可與管道搭配使用的版本控制 Amazon S3 儲存貯體。您會建立 AWS CloudFormation 範本、壓縮該範本，並以 .zip 檔案上傳到該儲存貯體。接著您必須新增來源動作到自儲存貯體擷取此 .zip 檔案的管道。

> **Note**
>
> 當 Amazon S3 是管道的來源供應商時，您可以將一或多個來源檔案壓縮為單一 .zip，然後將 .zip 上傳到來源儲存貯體。您也可以上傳單一解壓縮檔案；不過，預期 .zip 檔案的下游動作會失敗。

此範例將示範：

- 使用 JSON 編碼使用者參數來傳遞多個組態值到函數 (`get_user_params`)。
- 與成品儲存貯體中的 .zip 成品互動 (`get_template`)。
- 使用連續字符來監控長時間執行的非同步處理 (`continue_job_later`)。這允許動作繼續，並且即使函數超過 15 分鐘的執行階段 (Lambda 中的限制) 也會成功。

若要使用此範例 Lambda 函數，Lambda 執行角色的政策必須具有 Amazon S3 中AWS CloudFormation的`Allow`許可CodePipeline，並且如下列範例政策所示：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "logs:*"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:logs:*:*:*"
        },
        {
            "Action": [
                "codepipeline:PutJobSuccessResult",
                "codepipeline:PutJobFailureResult"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Action": [
                "cloudformation:DescribeStacks",
                "cloudformation:CreateStack",
                "cloudformation:UpdateStack"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Action": [
                "s3:*"
            ],
            "Effect": "Allow",
            "Resource": "*"
```

```
        }
    ]
}
```

若要建立 AWS CloudFormation 範本，請開啟任何純文字編輯器，並複製和貼上下列程式碼：

```
{
  "AWSTemplateFormatVersion" : "2010-09-09",
  "Description" : "CloudFormation template which creates an S3 bucket",
  "Resources" : {
    "MySampleBucket" : {
      "Type" : "AWS::S3::Bucket",
      "Properties" : {
      }
    }
  },
  "Outputs" : {
    "BucketName" : {
      "Value" : { "Ref" : "MySampleBucket" },
      "Description" : "The name of the S3 bucket"
    }
  }
}
```

在名為 **template-package** 的目錄中將此另存為使用名稱 **template.json** 的 JSON 檔案。為此目錄和檔案建立名為的壓縮 (.zip) 檔案**template-package.zip**，並將壓縮檔上傳到版本控制的 Amazon S3 儲存貯體。若您已為管道設定儲存貯體，便可以使用。接著，編輯您的管道以新增擷取該 .zip 檔的來源動作。命名此動作的輸出*MyTemplate*。如需詳細資訊，請參閱在中編輯配管 CodePipeline。

> **Note**
>
> 範例 Lambda 函數需要這些檔案名稱和壓縮結構。但是，您也可以用自己的 AWS CloudFormation 範本來取代此範例。如果您使用自己的範本，請務必修改 Lambda 執行角色的政策，以允許AWS CloudFormation範本所需的任何其他功能。

將以下代碼添加為 Lambda 中的函數

1. 開啟 Lambda 主控台，然後選擇建立函數。

2. 在 Create function (建立函數) 頁面上，選擇 Author from scratch (從頭開始撰寫)。在函數名稱中，輸入 Lambda 函數的名稱。
3. 在 Runtime (執行時間) 中，選擇 Python 2.7。
4. 在 [選擇或建立執行角色] 底下，選取 [使用現有角色]。在 Existing role (現有角色) 中選擇您的角色，然後選擇 Create function (建立函數)。

   會開啟建立的函數詳細資訊頁面。
5. 複製下列程式碼到 Function code (函數程式碼) 方塊中：

```
from __future__ import print_function
from boto3.session import Session

import json
import urllib
import boto3
import zipfile
import tempfile
import botocore
import traceback

print('Loading function')

cf = boto3.client('cloudformation')
code_pipeline = boto3.client('codepipeline')

def find_artifact(artifacts, name):
    """Finds the artifact 'name' among the 'artifacts'

    Args:
        artifacts: The list of artifacts available to the function
        name: The artifact we wish to use
    Returns:
        The artifact dictionary found
    Raises:
        Exception: If no matching artifact is found

    """
    for artifact in artifacts:
        if artifact['name'] == name:
            return artifact

    raise Exception('Input artifact named "{0}" not found in event'.format(name))
```

```
def get_template(s3, artifact, file_in_zip):
    """Gets the template artifact

    Downloads the artifact from the S3 artifact store to a temporary file
    then extracts the zip and returns the file containing the CloudFormation
    template.

    Args:
        artifact: The artifact to download
        file_in_zip: The path to the file within the zip containing the template

    Returns:
        The CloudFormation template as a string

    Raises:
        Exception: Any exception thrown while downloading the artifact or unzipping
 it

    """
    tmp_file = tempfile.NamedTemporaryFile()
    bucket = artifact['location']['s3Location']['bucketName']
    key = artifact['location']['s3Location']['objectKey']

    with tempfile.NamedTemporaryFile() as tmp_file:
        s3.download_file(bucket, key, tmp_file.name)
        with zipfile.ZipFile(tmp_file.name, 'r') as zip:
            return zip.read(file_in_zip)

def update_stack(stack, template):
    """Start a CloudFormation stack update

    Args:
        stack: The stack to update
        template: The template to apply

    Returns:
        True if an update was started, false if there were no changes
        to the template since the last update.

    Raises:
        Exception: Any exception besides "No updates are to be performed."

    """
```

```
    try:
        cf.update_stack(StackName=stack, TemplateBody=template)
        return True

    except botocore.exceptions.ClientError as e:
        if e.response['Error']['Message'] == 'No updates are to be performed.':
            return False
        else:
            raise Exception('Error updating CloudFormation stack
 "{0}"'.format(stack), e)

def stack_exists(stack):
    """Check if a stack exists or not

    Args:
        stack: The stack to check

    Returns:
        True or False depending on whether the stack exists

    Raises:
        Any exceptions raised .describe_stacks() besides that
        the stack doesn't exist.

    """
    try:
        cf.describe_stacks(StackName=stack)
        return True
    except botocore.exceptions.ClientError as e:
        if "does not exist" in e.response['Error']['Message']:
            return False
        else:
            raise e

def create_stack(stack, template):
    """Starts a new CloudFormation stack creation

    Args:
        stack: The stack to be created
        template: The template for the stack to be created with

    Throws:
        Exception: Any exception thrown by .create_stack()
    """
```

```
    cf.create_stack(StackName=stack, TemplateBody=template)

def get_stack_status(stack):
    """Get the status of an existing CloudFormation stack

    Args:
        stack: The name of the stack to check

    Returns:
        The CloudFormation status string of the stack such as CREATE_COMPLETE

    Raises:
        Exception: Any exception thrown by .describe_stacks()

    """
    stack_description = cf.describe_stacks(StackName=stack)
    return stack_description['Stacks'][0]['StackStatus']

def put_job_success(job, message):
    """Notify CodePipeline of a successful job

    Args:
        job: The CodePipeline job ID
        message: A message to be logged relating to the job status

    Raises:
        Exception: Any exception thrown by .put_job_success_result()

    """
    print('Putting job success')
    print(message)
    code_pipeline.put_job_success_result(jobId=job)

def put_job_failure(job, message):
    """Notify CodePipeline of a failed job

    Args:
        job: The CodePipeline job ID
        message: A message to be logged relating to the job status

    Raises:
        Exception: Any exception thrown by .put_job_failure_result()

    """
```

```
    print('Putting job failure')
    print(message)
    code_pipeline.put_job_failure_result(jobId=job, failureDetails={'message':
 message, 'type': 'JobFailed'})

def continue_job_later(job, message):
    """Notify CodePipeline of a continuing job

    This will cause CodePipeline to invoke the function again with the
    supplied continuation token.

    Args:
        job: The JobID
        message: A message to be logged relating to the job status
        continuation_token: The continuation token

    Raises:
        Exception: Any exception thrown by .put_job_success_result()

    """

    # Use the continuation token to keep track of any job execution state
    # This data will be available when a new job is scheduled to continue the
 current execution
    continuation_token = json.dumps({'previous_job_id': job})

    print('Putting job continuation')
    print(message)
    code_pipeline.put_job_success_result(jobId=job,
 continuationToken=continuation_token)

def start_update_or_create(job_id, stack, template):
    """Starts the stack update or create process

    If the stack exists then update, otherwise create.

    Args:
        job_id: The ID of the CodePipeline job
        stack: The stack to create or update
        template: The template to create/update the stack with

    """
    if stack_exists(stack):
        status = get_stack_status(stack)
```

```
        if status not in ['CREATE_COMPLETE', 'ROLLBACK_COMPLETE',
 'UPDATE_COMPLETE']:
            # If the CloudFormation stack is not in a state where
            # it can be updated again then fail the job right away.
            put_job_failure(job_id, 'Stack cannot be updated when status is: ' +
 status)
            return

        were_updates = update_stack(stack, template)

        if were_updates:
            # If there were updates then continue the job so it can monitor
            # the progress of the update.
            continue_job_later(job_id, 'Stack update started')

        else:
            # If there were no updates then succeed the job immediately
            put_job_success(job_id, 'There were no stack updates')
    else:
        # If the stack doesn't already exist then create it instead
        # of updating it.
        create_stack(stack, template)
        # Continue the job so the pipeline will wait for the CloudFormation
        # stack to be created.
        continue_job_later(job_id, 'Stack create started')

def check_stack_update_status(job_id, stack):
    """Monitor an already-running CloudFormation update/create

    Succeeds, fails or continues the job depending on the stack status.

    Args:
        job_id: The CodePipeline job ID
        stack: The stack to monitor

    """
    status = get_stack_status(stack)
    if status in ['UPDATE_COMPLETE', 'CREATE_COMPLETE']:
        # If the update/create finished successfully then
        # succeed the job and don't continue.
        put_job_success(job_id, 'Stack update complete')

    elif status in ['UPDATE_IN_PROGRESS', 'UPDATE_ROLLBACK_IN_PROGRESS',
    'UPDATE_ROLLBACK_COMPLETE_CLEANUP_IN_PROGRESS', 'CREATE_IN_PROGRESS',
```

```
    'ROLLBACK_IN_PROGRESS', 'UPDATE_COMPLETE_CLEANUP_IN_PROGRESS']:
        # If the job isn't finished yet then continue it
        continue_job_later(job_id, 'Stack update still in progress')

    else:
        # If the Stack is a state which isn't "in progress" or "complete"
        # then the stack update/create has failed so end the job with
        # a failed result.
        put_job_failure(job_id, 'Update failed: ' + status)

def get_user_params(job_data):
    """Decodes the JSON user parameters and validates the required properties.

    Args:
        job_data: The job data structure containing the UserParameters string which
 should be a valid JSON structure

    Returns:
        The JSON parameters decoded as a dictionary.

    Raises:
        Exception: The JSON can't be decoded or a property is missing.

    """
    try:
        # Get the user parameters which contain the stack, artifact and file
 settings
        user_parameters = job_data['actionConfiguration']['configuration']
['UserParameters']
        decoded_parameters = json.loads(user_parameters)

    except Exception as e:
        # We're expecting the user parameters to be encoded as JSON
        # so we can pass multiple values. If the JSON can't be decoded
        # then fail the job with a helpful message.
        raise Exception('UserParameters could not be decoded as JSON')

    if 'stack' not in decoded_parameters:
        # Validate that the stack is provided, otherwise fail the job
        # with a helpful message.
        raise Exception('Your UserParameters JSON must include the stack name')

    if 'artifact' not in decoded_parameters:
        # Validate that the artifact name is provided, otherwise fail the job
```

```
        # with a helpful message.
        raise Exception('Your UserParameters JSON must include the artifact name')

    if 'file' not in decoded_parameters:
        # Validate that the template file is provided, otherwise fail the job
        # with a helpful message.
        raise Exception('Your UserParameters JSON must include the template file
 name')

    return decoded_parameters

def setup_s3_client(job_data):
    """Creates an S3 client

    Uses the credentials passed in the event by CodePipeline. These
    credentials can be used to access the artifact bucket.

    Args:
        job_data: The job data structure

    Returns:
        An S3 client with the appropriate credentials

    """
    key_id = job_data['artifactCredentials']['accessKeyId']
    key_secret = job_data['artifactCredentials']['secretAccessKey']
    session_token = job_data['artifactCredentials']['sessionToken']

    session = Session(aws_access_key_id=key_id,
        aws_secret_access_key=key_secret,
        aws_session_token=session_token)
    return session.client('s3',
 config=botocore.client.Config(signature_version='s3v4'))

def lambda_handler(event, context):
    """The Lambda function handler

    If a continuing job then checks the CloudFormation stack status
    and updates the job accordingly.

    If a new job then kick of an update or creation of the target
    CloudFormation stack.

    Args:
```

```
        event: The event passed by Lambda
        context: The context passed by Lambda

    """
    try:
        # Extract the Job ID
        job_id = event['CodePipeline.job']['id']

        # Extract the Job Data
        job_data = event['CodePipeline.job']['data']

        # Extract the params
        params = get_user_params(job_data)

        # Get the list of artifacts passed to the function
        artifacts = job_data['inputArtifacts']

        stack = params['stack']
        artifact = params['artifact']
        template_file = params['file']

        if 'continuationToken' in job_data:
            # If we're continuing then the create/update has already been triggered
            # we just need to check if it has finished.
            check_stack_update_status(job_id, stack)
        else:
            # Get the artifact details
            artifact_data = find_artifact(artifacts, artifact)
            # Get S3 client to access artifact with
            s3 = setup_s3_client(job_data)
            # Get the JSON template file out of the artifact
            template = get_template(s3, artifact_data, template_file)
            # Kick off a stack update or create
            start_update_or_create(job_id, stack, template)

    except Exception as e:
        # If any other exceptions which we didn't expect are raised
        # then fail the job and log the exception message.
        print('Function failed due to exception.')
        print(e)
        traceback.print_exc()
        put_job_failure(job_id, 'Function exception: ' + str(e))

    print('Function complete.')
```

```
    return "Complete."
```

6. 將 [處理常式] 保留為預設值，並將角色保留在您先前選取或建立的名稱**CodePipelineLambdaExecRole**。
7. 在 Basic settings (基本設定) 的 Timeout (逾時) 中，以 **20** 取代預設值 3 秒。
8. 選擇 儲存 。
9. 從 CodePipeline 主控台，編輯要新增函數的管道，做為管道中階段動作的函數。針對您要變更的管線階段選擇 「編輯」，然後選擇「新增動作群組」。在 [編輯動作] 頁面的 [動作名稱] 中，輸入動作的名稱。在動作提供者中，選擇 Lambda。

   在「輸入人工因素」下選擇MyTemplate。在中 UserParameters，您必須提供具有三個參數的 JSON 字串：

   - Stack name (堆疊名稱)
   - AWS CloudFormation 範本名稱和該檔案的路徑
   - 輸入成品

   使用大括弧 ({ })，並以逗號分隔參數。例如，要為具有輸入工件的管道創建一個名*MyTestStack*為的堆棧 *MyTemplate*，請在中UserParameters輸入：{*MyTestStack*「堆棧」：「」，「文件」：「模板包/template.json」，「工件」：*MyTemplate*「}。

   > **Note**
   >
   > 即使您已在中指定輸入人工因素 UserParameters，您也必須為「輸入」人工因素中的動作指定此輸入人工因素。

10. 將變更儲存至管道，然後手動發行變更以測試動作和 Lambda 函數。

重試階段中失敗的動作

您可以重試失敗的階段，而不必從頭開始再次執行管線。您可以在階段中重試失敗的動作，或重試階段中的所有動作，從階段中的第一個動作開始。當您重試階段中的失敗動作時，所有仍在進行中的動作會繼續運作，而失敗的動作會再次觸發。當您從階段中的第一個動作重試失敗的階段時，階段不會有任何正在進行中的動作。在可以重試階段之前，它必須讓所有動作都失敗，或是某些動作失敗且某些動作成功。

> **Important**
>
> 重試失敗的階段會從階段中的第一個動作重試階段中的所有動作，重試失敗的動作會重試階段中所有失敗的動作。這會覆寫相同執行中先前成功動作的輸出成品。
> 雖然可能會覆寫人工因素，但仍會保留先前成功動作的執行歷程記錄。

如果您使用主控台來檢視管線，則可重試階段按鈕或重試失敗動作按鈕會出現在可重試的階段上。

如果您使用 AWS CLI，可以使用get-pipeline-state命令來判斷是否有任何動作失敗。

> **Note**
>
> 在下列情況下，您可能無法重試階段：
>
> - 階段中的所有動作都成功，因此階段不會處於失敗狀態。
> - 階段失敗後，整體配管結構已變更。
> - 階段中的另一個重試正在處理中。

主題

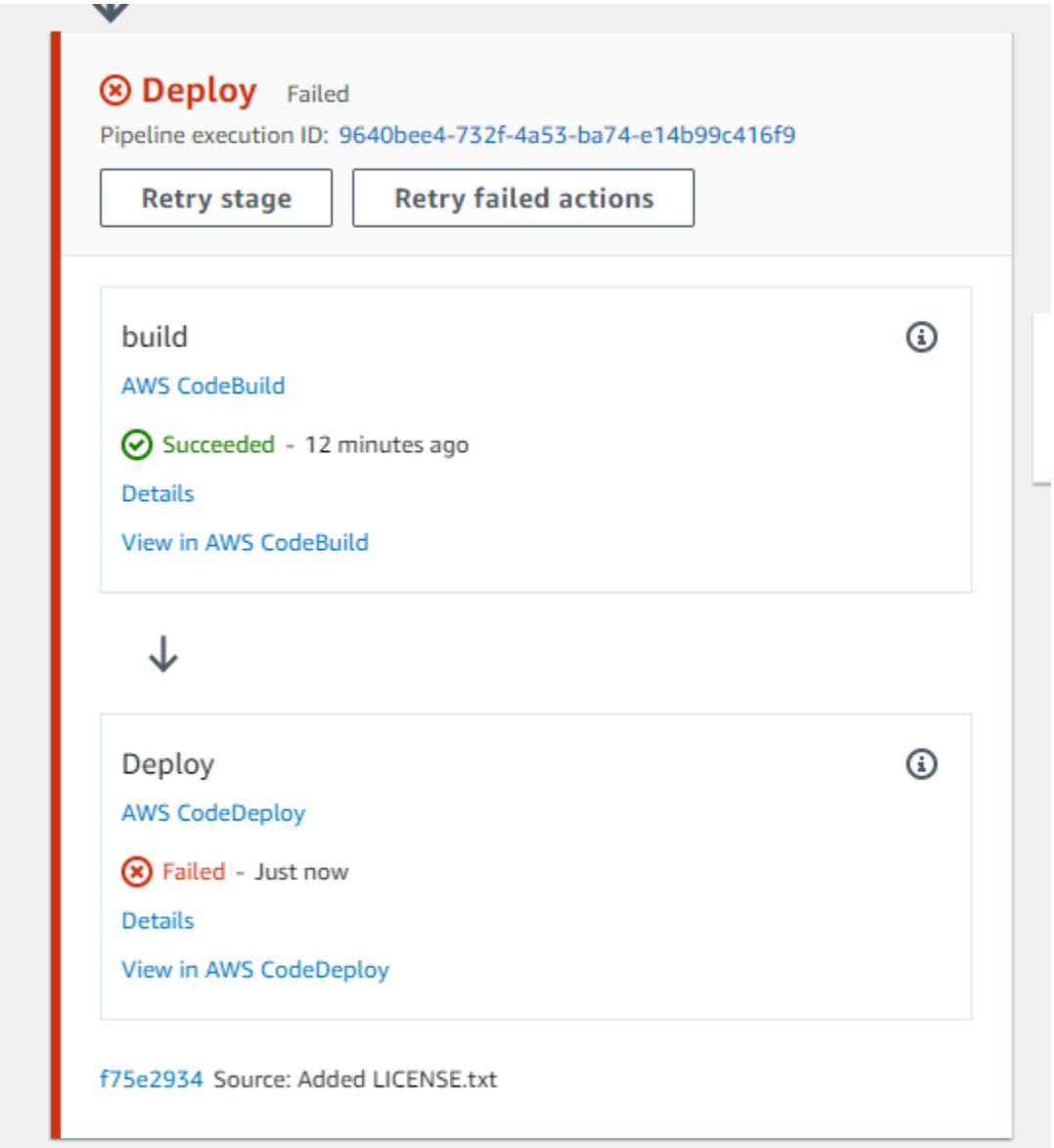
重試失敗的動作 (主控台)

在階段-主控台中重試失敗的階段或失敗的動作

1. 請登入AWS Management Console並開啟 CodePipeline 主控台，網址為 http://console.aws.amazon.com/codesuite/codepipeline/home。

   所有與您的 AWS 帳戶相關的管道名稱都會顯示。
2. 在 Name (名稱) 中，選擇管道的名稱。
3. 找出具有失敗動作的階段，然後選擇下列其中一項：

   - 若要重試階段中的所有動作，請選擇「重試階段」。
   - 若要僅重試階段中失敗的動作，請選擇「重試失敗的動作」。



若階段中所有重試動作皆已成功完成，管道將持續執行。

重試失敗的動作 (CLI)

若要在階段中重試失敗的階段或失敗的動作-CLI

若要使用重AWS CLI試所有動作或所有失敗的動作，請使用下列參數執行retry-stage-execution命令：

```
--pipeline-name <value>
--stage-name <value>
```

```
--pipeline-execution-id <value>
--retry-mode ALL_ACTIONS/FAILED_ACTIONS
```

> **Note**
>
> 您可以使用的值retry-mode是FAILED_ACTIONS和ALL_ACTIONS。

1. 在終端機 (Linux、macOS 或 Unix) 或命令提示字元 (Windows) 上，執行retry-stage-execution命令，如下列範例所示，針對名稱為的管線MyPipeline。

   ```
   aws codepipeline retry-stage-execution --pipeline-name MyPipeline --stage-name
    Deploy --pipeline-execution-id b59babff-5f34-EXAMPLE --retry-mode FAILED_ACTIONS
   ```

   輸出會傳回執行 ID：

   ```
   {
       "pipelineExecutionId": "b59babff-5f34-EXAMPLE"
   }
   ```

2. 您也可以使用 JSON 輸入檔案來執行命令。首先您必須建立 JSON 檔案，此檔案指出管道、包含失敗動作的階段，以及該階段中最新的管道執行。接著使用 `--cli-input-json` 參數來執行 retry-stage-execution 命令。若要擷取 JSON 檔案所需的詳細資訊，使用 get-pipeline-state 命令是最簡單的方法。

   a. 在終端機 (Linux、macOS 或 Unix) 或命令提示字元 (Windows) 上，在管線上執行get-pipeline-state命令。例如，對於名為的管線 MyFirstPipeline，您可以鍵入類似下列內容的內容：

      ```
      aws codepipeline get-pipeline-state --name MyFirstPipeline
      ```

      對於命令的回應包括各階段的管道狀態資訊。在以下範例中，回應表示在預備 (Staging) 階段中失敗的一個或多個動作。

      ```
      {
          "updated": 1427245911.525,
          "created": 1427245911.525,
          "pipelineVersion": 1,
          "pipelineName": "MyFirstPipeline",
      ```

```
    "stageStates": [
        {
            "actionStates": [...],
            "stageName": "Source",
            "latestExecution": {
                "pipelineExecutionId": "9811f7cb-7cf7-SUCCESS",
                "status": "Succeeded"
            }
        },
        {
            "actionStates": [...],
            "stageName": "Staging",
            "latestExecution": {
                "pipelineExecutionId": "3137f7cb-7cf7-EXAMPLE",
                "status": "Failed"
            }
        }
    ]
}
```

b. 在純文字編輯器中，以 JSON 格式來建立您將用於記錄下列內容的檔案：

- 包含失敗動作之管道的名稱
- 包含失敗動作之階段的名稱
- 該階段中最新管道執行的 ID
- 重試模式。

在前面的 MyFirstPipeline 例子中，您的文件看起來像這樣：

```
{
    "pipelineName": "MyFirstPipeline",
    "stageName": "Staging",
    "pipelineExecutionId": "3137f7cb-7cf7-EXAMPLE",
    "retryMode": "FAILED_ACTIONS"
}
```

c. 以類似 **`retry-failed-actions.json`** 的名稱儲存檔案。

d. 呼叫您執行 retry-stage-execution 命令時建立的檔案。例如：

> **Important**
>
> 請確認在檔案名稱之前包含 `file://`。這是此命令必要項目。

```
aws codepipeline retry-stage-execution --cli-input-json file://retry-failed-actions.json
```

e. 若要檢視重試嘗試的結果，請開啟 CodePipeline 主控台並選擇包含失敗動作的管線，或再次使用get-pipeline-state命令。如需詳細資訊，請參閱 在中檢視管線和詳細資訊 CodePipeline。

管理中的核准動作 CodePipeline

在 AWS CodePipeline 中，您可以將核准動作新增到管道中您希望停止管道執行的階段，讓擁有必要 AWS Identity and Access Management 許可的人員可核准或拒絕動作。

若動作獲得核准，管道執行將繼續。如果動作被拒絕 (或者在管線達到動作並停止後的七天內沒有人核准或拒絕動作)，則結果與動作失敗相同，且管線執行不會繼續。

由於這些原因，您可以使用手動核准：

- 您想要在允許修訂進入管道的下一階段前，有人執行程式碼審視或更改管理審視。
- 您想要有人針對應用程式最新版本在發行前執行手動品質保證測試，或者確認建構成品的完整性。
- 您想要有人在新文字或更新文字發佈到公司網站前進行審視。

中手動核准動作的組態選項 CodePipeline

CodePipeline 提供三個組態選項，可讓您用來告知核准人核准動作。

發佈核准通知您可以設定核准動作，以便在管道停止動作時向 Amazon 簡單通知服務主題發佈訊息。Amazon SNS 會將訊息傳送到訂閱該主題的每個端點。您必須使用在同一AWS區域中建立的主題作為將包含核准動作的管道。當您建立主題時，我們建議您以能夠判別其目的之名稱命名，例如像是 `MyFirstPipeline-us-east-2-approval` 的格式。

當您將核准通知發佈到 Amazon SNS 主題時，您可以選擇電子郵件或簡訊收件者、SQS 佇列、HTTP/HTTPS 端點或您使用 Amazon SNS 叫用的AWS Lambda函數等格式。如需 Amazon SNS 主題通知的相關資訊，請參閱下列主題：

- 什麼是 Amazon 簡單通知服務？
- 在 Amazon SNS 中創建一個主題
- 傳送 Amazon SNS 訊息到 Amazon SQS 佇列
- 為佇列訂閱 Amazon SNS 主題
- 傳送 Amazon SNS 訊息至 HTTP/HTTPS 端點
- 使用 Amazon SNS 通知叫用 Lambda 函數

如需為核准動作通知產生的 JSON 資料結構，請參閱 手動核准通知的 JSON 資料格式 CodePipeline。

指定用於審視的 URL 做為核准動作組態的一部分，您可以指定要審視的 URL。URL 可能是您想讓核准者測試的 web 應用程式連結，或是擁有關於核准請求之其他資訊的頁面。該 URL 包含在發佈至 Amazon SNS 主題的通知中。核准者可使用主控台或 CLI 來檢視。

輸入核准者的評論 當您建立核准動作時，也可新增對收到通知的人 (或在主控台或 CLI 回應中檢視動作的人) 顯示的評論。

沒有組態選項 您也可以不要設置這三種選項的任何一種。有些情況下，您可能不需要組態：例如，您可以直接通知某人動作已可供檢閱，或者您只想讓管道停止，直到您自行決定核准動作。

在 CodePipeline 中的核准動作設定與工作流程概觀

下列為設定並使用手動核准的概觀。

1. 您將核准或拒絕核准動作所需的 IAM 許可授與組織中的一或多個 IAM 角色。
2. (選擇性) 如果您使用 Amazon SNS 通知，請確保您在 CodePipeline 操作中使用的服務角色具有存取 Amazon SNS 資源的權限。
3. (選擇性) 如果您使用 Amazon SNS 通知，請建立 Amazon SNS 主題，並在其中新增一或多個訂閱者或端點。
4. 當您使用 AWS CLI 建立管線時，或在使用 CLI 或主控台建立管線之後，您可以將核准動作新增至管線中的階段。

   如果您使用的是通知，請在動作的組態中包含 Amazon SNS 主題的 Amazon 資源名稱 (ARN)。（ ARN 是 Amazon 資源的唯一標識符。 Amazon SNS 主題的 ARN 的結構類似於 *arn: awn:*

SNS: ####-2:80398 ##:. MyApprovalTopic 如需詳細資訊，請參閱中的 Amazon 資源名稱 (ARN) 和AWS 服務命名空間。) Amazon Web Services 一般參考

5. 管道會在到達核准動作時停止。如果動作的組態中包含 Amazon SNS 主題 ARN，則會向 Amazon SNS 主題發佈通知，並將訊息傳送給主題或訂閱端點的任何訂閱者，並在主控台中檢閱核准動作的連結。
6. 核准者會檢查目標 URL 並檢閱註解 (若有的話)。
7. 使用主控台、CLI 或軟體開發套件，核准者可提供摘要註解並提交回應：
   - 核准：繼續執行管道。
   - 拒絕：階段狀態會變更為 "Failed" (失敗)，並且不會繼續執行管道。

   若七天內沒有提交任何回應，則會將動作標記為 "Failed" (失敗)。

將核准許可授予 IAM 使用者 CodePipeline

組織中的 IAM 使用者必須先獲得存取管道和更新核准動作狀態的權限，才能核准或拒絕核准動作。您可以將`AWSCodePipelineApproverAccess`受管政策附加到 IAM 使用者、角色或群組，以授與存取帳戶中所有管道和核准動作的權限；或者，您也可以指定 IAM 使用者、角色或群組可存取的個別資源，以授予有限的權限。

> **Note**
>
> 本主題中說明的許可會授予非常有限的存取。若要讓使用者、角色或群組執行核准或拒絕核准動作以外的作業，您可以連接其他受管政策。如需有關可用於的受管理原則的資訊 CodePipeline，請參閱AWS 受管理的政策 AWS CodePipeline。

授予所有管道及核准動作的核准許可

對於需要在中執行核准動作的使用者 CodePipeline，請使用受`AWSCodePipelineApproverAccess`管理的策略。

若要提供存取權，請新增權限至您的使用者、群組或角色：

- AWS IAM Identity Center 中的使用者和群組：

  建立權限合集。請遵循 AWS IAM Identity Center 使用者指南 的 建立權限合集 中的指示。
- 透過身分提供者在 IAM 中管理的使用者：

建立聯合身分的角色。請遵循 IAM 使用者指南 的 為第三方身分提供者 (聯合) 建立角色 中的指示。

- IAM 使用者：
  - 建立您的使用者可擔任的角色。請遵循 IAM 使用者指南 的 為 IAM 使用者建立角色 中的指示。
  - (不建議) 將政策直接附加至使用者，或將使用者新增至使用者群組。請遵循 IAM 使用者指南 的 新增權限至使用者 (主控台) 中的指示。

指定特定管道及核准動作的核准許可

對於需要在中執行核准動作的使用者 CodePipeline，請使用下列自訂原則。在以下策略中，指定使用者可以存取的個別資源。例如，下列政策授予使用者僅核准或拒絕美東部 (俄亥俄) 區域 (us-east-2) `MyFirstPipeline` 管道中指`MyApprovalAction`定的動作的權限：

> **Note**
>
> 僅當 `codepipeline:ListPipelines` IAM 使用者需要存取 CodePipeline 儀表板以檢視此管道清單時，才需要此權限。若不需要主控台存取，您可以忽略 `codepipeline:ListPipelines`。

若要使用 JSON 政策編輯器來建立政策

1. 登入 AWS Management Console，並開啟位於 https://console.aws.amazon.com/iam/ 的 IAM 主控台。
2. 在左側的導覽窗格中，選擇 Policies (政策)。

   如果這是您第一次選擇 Policies (政策)，將會顯示 Welcome to Managed Policies (歡迎使用受管政策) 頁面。選擇 Get Started (開始使用)。
3. 在頁面頂端，選擇 Create policy (建立政策)。
4. 在政策編輯器中，選擇 JSON 選項。
5. 輸入下列 JSON 政策文件：

```
{
   "Version": "2012-10-17",
   "Statement": [
       {
           "Effect": "Allow",
```

```
            "Action": [
                "codepipeline:ListPipelines"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "codepipeline:GetPipeline",
                "codepipeline:GetPipelineState",
                "codepipeline:GetPipelineExecution"
            ],
            "Resource": "arn:aws:codepipeline:us-
east-2:80398EXAMPLE:MyFirstPipeline"
        },
        {
            "Effect": "Allow",
            "Action": [
                "codepipeline:PutApprovalResult"
            ],
            "Resource": "arn:aws:codepipeline:us-
east-2:80398EXAMPLE:MyFirstPipeline/MyApprovalStage/MyApprovalAction"
        }
    ]
}
```

6. 選擇下一步。

> **Note**
>
> 您可以隨時切換視覺化與 JSON 編輯器選項。不過，如果您進行變更或在視覺化編輯器中選擇下一步，IAM 就可能會調整您的政策結構，以便針對視覺化編輯器進行最佳化。如需詳細資訊，請參閱 IAM 使用者指南中的調整政策結構。

7. 在檢視與建立頁面上，為您正在建立的政策輸入政策名稱與描述 (選用)。檢視此政策中定義的許可，來查看您的政策所授予的許可。

8. 選擇 Create policy (建立政策) 儲存您的新政策。

將 Amazon SNS 許可授與 CodePipeline服務角色

如果您計劃在核准動作需要審核時使用 Amazon SNS 向主題發佈通知，則您在 CodePipeline 操作中使用的服務角色必須獲得存取 Amazon SNS 資源的權限。您可以使用 IAM 主控台將此權限新增至您的服務角色。

在下列原則中，指定使用 SNS 發佈的原則。對於以下策略，您可以將其命名SNSPublish。將其附加到您的服務角色，以使用下列原則。

> **Important**
>
> 確認您已利用在 開始使用 CodePipeline 中使用的相同帳戶資訊登入 AWS Management Console。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "sns:Publish",
            "Resource": "*"
        }
    ]
}
```

若要使用 JSON 政策編輯器來建立政策

1. 登入 AWS Management Console，並開啟位於 https://console.aws.amazon.com/iam/ 的 IAM 主控台。
2. 在左側的導覽窗格中，選擇 Policies (政策)。

   如果這是您第一次選擇 Policies (政策)，將會顯示 Welcome to Managed Policies (歡迎使用受管政策) 頁面。選擇 Get Started (開始使用)。
3. 在頁面頂端，選擇 Create policy (建立政策)。
4. 在政策編輯器中，選擇 JSON 選項。
5. 輸入或貼上 JSON 政策文件。如需有關 IAM 政策語言的詳細資訊，請參閱 IAM JSON 政策參考。
6. 解決政策驗證期間產生的任何安全性警告、錯誤或一般性警告，然後選擇 Next (下一步)。

> **Note**
>
> 您可以隨時切換視覺化與 JSON 編輯器選項。不過，如果您進行變更或在視覺化編輯器中選擇下一步，IAM 就可能會調整您的政策結構，以便針對視覺化編輯器進行最佳化。如需詳細資訊，請參閱 IAM 使用者指南中的調整政策結構。

7. (選用) 在 AWS Management Console 中建立或編輯政策時，您可以產生可在 AWS CloudFormation 範本中使用的 JSON 或 YAML 政策範本。

   若要這麼做，請在 [原則編輯器] 中選擇 [動作]，然後選擇 [產生 CloudFormation範本]。若要進一步了解 AWS CloudFormation，請參閱《AWS CloudFormation 使用者指南》中的 AWS Identity and Access Management 資源類型參考。
8. 將許可新增至政策後，請選擇下一步。
9. 在檢視與建立頁面上，為您在建立的政策輸入政策名稱與描述 (選用)。檢視此政策中定義的許可，來查看您的政策所授予的許可。
10. (選用) 藉由連接標籤作為鍵值組，將中繼資料新增至政策。如需有關在 IAM 中使用標籤的詳細資訊，請參閱《IAM 使用者指南》中的標記 IAM 資源。
11. 選擇 Create policy (建立政策) 儲存您的新政策。

將手動核准動作新增至 CodePipeline 中的管道

您可以在您希望 CodePipeline 管道停止的時間點將核准動作新增至管道中的階段，以便某人可以手動核准或拒絕該動作。

> **Note**
>
> 核准動作無法新增至「來源」階段。來源階段僅能包含來源動作。

如果您想要在核准動作準備好可供審閱時使用 Amazon SNS 傳送通知，您必須先完成下列先決條件：

- 授與您的 CodePipeline 服務角色存取 Amazon SNS 資源的權限。如需相關資訊，請參閱 將 Amazon SNS 許可授與 CodePipeline服務角色。
- 授予組織中一或多個 IAM 身分的權限，以更新核准動作的狀態。如需相關資訊，請參閱 將核准許可授予 IAM 使用者 CodePipeline。

在此範例中，您會建立新的核准階段，並將手動核准動作新增至階段。您也可以將手動核准動作新增至包含其他動作的現有階段。

將手動核准動作新增至 CodePipeline 管道 (主控台)

您可以使用 CodePipeline 主控台將核准動作新增至現有 CodePipeline管道。如果要在建立新管道時新增核准動作，則必須使用 AWS CLI。

1. 開啟主 CodePipeline 控台，網址為 https://console.aws.amazon.com/codepipeline/。
2. 在 Name (名稱) 中，選擇管道。
3. 在管道詳細資訊頁面上，選擇 Edit (編輯)。
4. 若您希望將核准動作新增至新的階段，請在管道中您希望新增核准請求的位置選擇 + Add stage (+ 新增階段)，然後輸入階段的名稱。在 Add stage (新增階段) 頁面的 Stage name (階段名稱) 中，輸入新的階段名稱。例如，新增階段並命名為 `Manual_Approval`。

   若您希望將核准動作新增至現有的階段，請選擇 Edit stage (編輯階段)。
5. 在您要新增核准動作的階段中，選擇 + Add action group (+ 新增動作群組)。
6. 在 Edit action (編輯動作) 頁面上，執行下列作業：
   1. 在 Action name (動作名稱) 中，輸入識別該動作的名稱。
   2. 在 Action provider (動作供應商) 的 Approval (核准) 下，選擇 Manual approval (手動核准)。
   3. (選擇性) 在 SNS topic ARN (SNS 主題 ARN) 中，選擇您用來傳送核准動作通知的主題名稱。
   4. (選擇性) 在 URL for review (檢閱的 URL) 中，輸入您希望核准者檢查的頁面或應用程式 URL。核准者可透過管道主控台檢視中包含的連結存取此 URL。
   5. (選擇性) 在 Comments (註解) 中，輸入任何您希望與檢閱者共享的其他資訊。
   6. 選擇儲存。

新增手動核准動作至 CodePipeline 管道 (CLI)

您可以使用 CLI 將核准動作新增至現有的管道 (或是在您建立管道時)。您可以透過在您建立或編輯的階段中包含核准類型為手動核准的核准動作，來執行此操作。

如需建立及編輯管道的詳細資訊，請參閱在中建立管線 CodePipeline和在中編輯配管 CodePipeline。

若要將階段新增至僅包含核准動作的管道，建議您在建立或更新管道時，包含與下列範例相似的內容。

> **Note**
>
> configuration 區段為選擇性區塊。此僅為一部分，而非整個結構或檔案。如需詳細資訊，請參閱CodePipeline 配管結構參照。

```
{
    "name": "MyApprovalStage",
    "actions": [
        {
            "name": "MyApprovalAction",
            "actionTypeId": {
                "category": "Approval",
                "owner": "AWS",
                "version": "1",
                "provider": "Manual"
            },
            "inputArtifacts": [],
            "outputArtifacts": [],
            "configuration": {
                "NotificationArn": "arn:aws:sns:us-
east-2:80398EXAMPLE:MyApprovalTopic",
                "ExternalEntityLink": "http://example.com",
                "CustomData": "The latest changes include feedback from Bob."},
            "runOrder": 1
        }
    ]
}
```

若核准動作位於具有其他動作的階段，則包含階段的 JSON 檔案區段可能看起來會與下列內容相似。

> **Note**
>
> configuration 區段為選擇性區塊。此僅為一部分，而非整個結構或檔案。如需詳細資訊，請參閱CodePipeline 配管結構參照。

```
,
{
    "name": "Production",
    "actions": [
```

```
        {
            "inputArtifacts": [],
            "name": "MyApprovalAction",
            "actionTypeId": {
                "category": "Approval",
                "owner": "AWS",
                "version": "1",
                "provider": "Manual"
            },
            "outputArtifacts": [],
            "configuration": {
                "NotificationArn": "arn:aws:sns:us-
east-2:80398EXAMPLE:MyApprovalTopic",
                "ExternalEntityLink": "http://example.com",
                "CustomData": "The latest changes include feedback from Bob."
            },
            "runOrder": 1
        },
        {
            "inputArtifacts": [
                {
                    "name": "MyApp"
                }
            ],
            "name": "MyDeploymentAction",
            "actionTypeId": {
                "category": "Deploy",
                "owner": "AWS",
                "version": "1",
                "provider": "CodeDeploy"
            },
            "outputArtifacts": [],
            "configuration": {
                "ApplicationName": "MyDemoApplication",
                "DeploymentGroupName": "MyProductionFleet"
            },
            "runOrder": 2
        }
    ]
}
```

在 CodePipeline 中核准或拒絕核准動作

當管道包含核准動作時，管道會在新增動作的位置停止執行。除非有人手動核准動作，否則管道不會繼續。若核准者拒絕動作，或是在管道因為核准動作停止之後的七天內沒有收到任何核准回應，則管道狀態會變更為 "Failed" (失敗)。

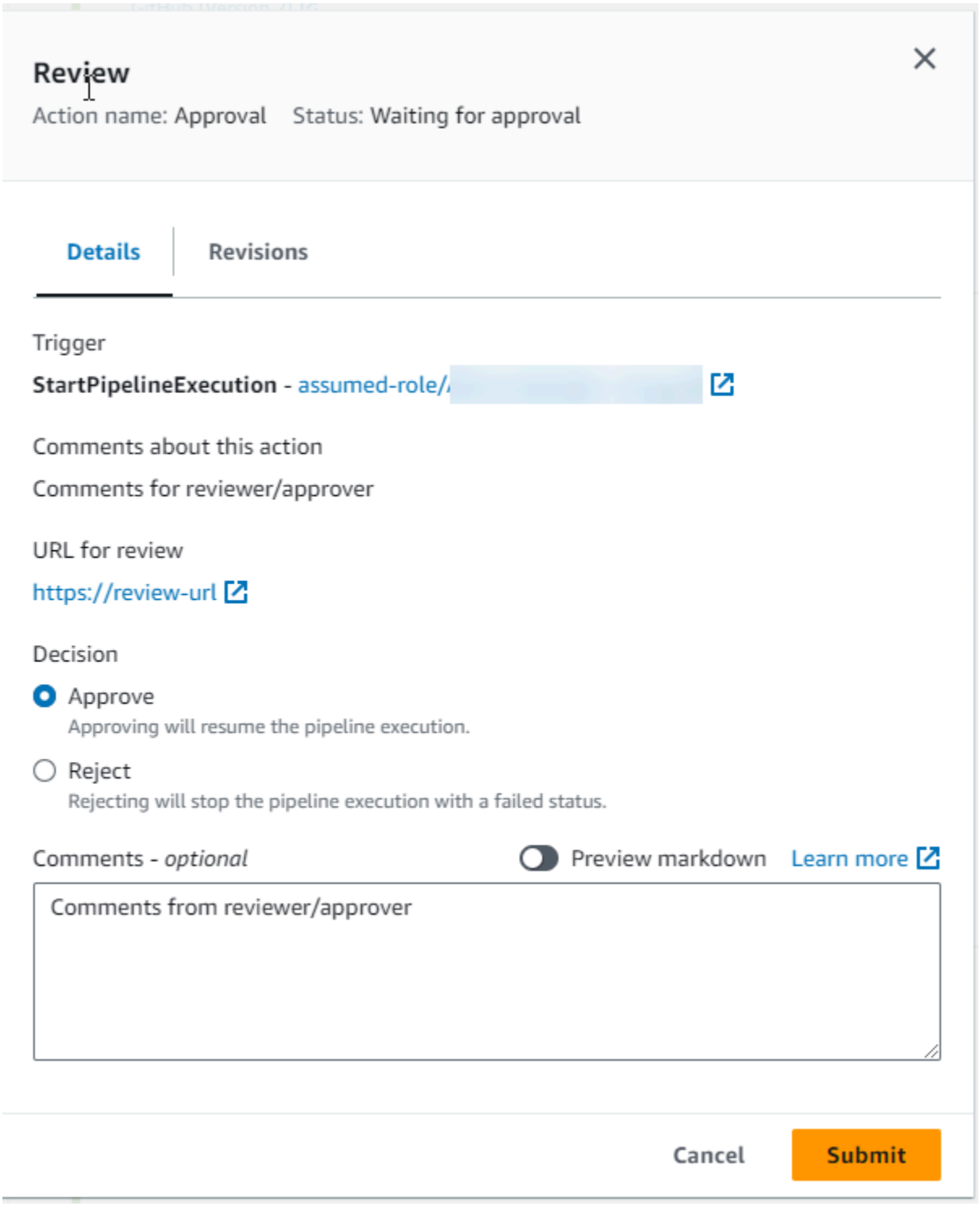
如果將核准動作新增至管道的人員已設定通知，您可能會收到一封電子郵件，其中包含管道資訊和狀態以供核准。

核准或拒絕核准動作 (主控台)

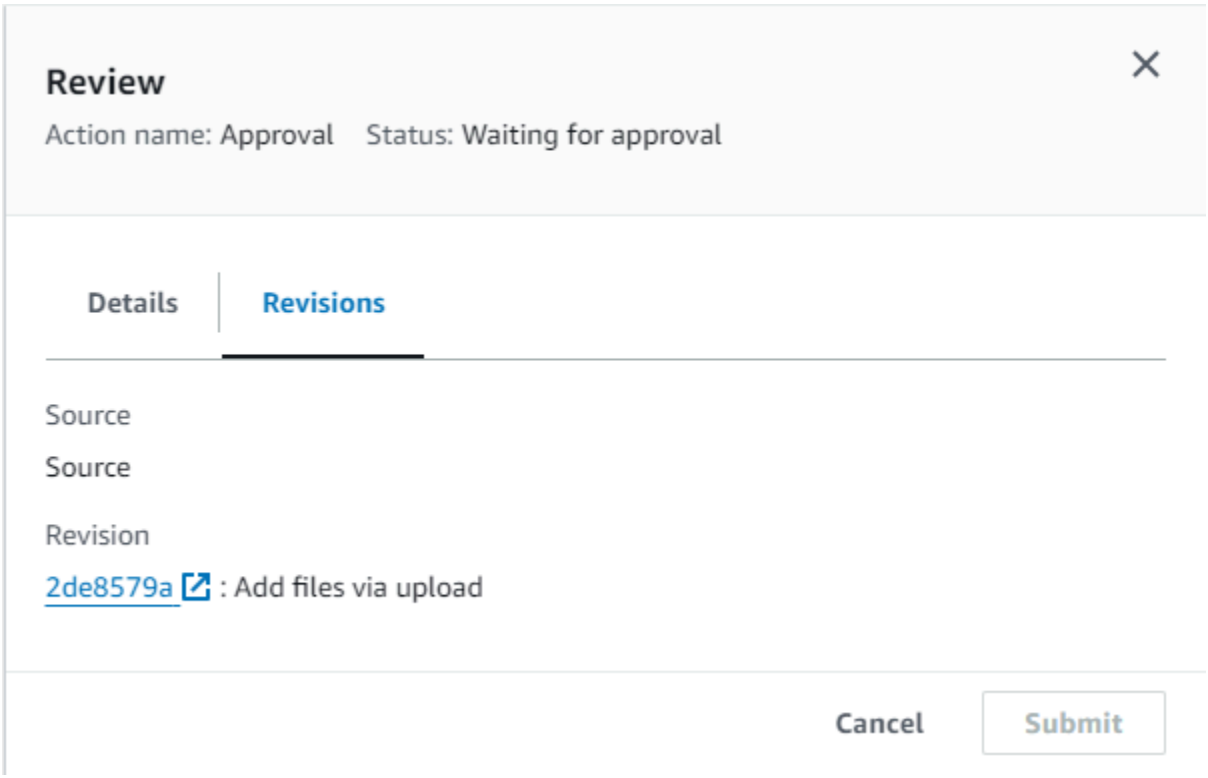
若您收到包含核准動作直接連結的通知，請選擇 Approve or reject (核准或拒絕) 連結、登入主控台，然後繼續此處的步驟 7。否則，請依照下列所有步驟進行。

1. 開啟主 CodePipeline 控台，網址為 https://console.aws.amazon.com/codepipeline/。
2. 在 All Pipelines (所有管道) 頁面上，選擇管道名稱。
3. 找到具有核准動作的階段。選擇 檢閱。

   將顯示「檢閱」對話方塊。[詳細資料] 索引標籤會顯示檢閱內容和註解。



「版本修訂」標籤會顯示執行的來源修訂。



4. 在 [詳細資料] 索引標籤上，檢視註解和 URL (如果有的話)。該訊息也會顯示需要您檢閱的內容 URL (若其中包含的話)。
5. 如果已提供 URL，請在動作中選擇要檢閱的 URL 連結以開啟目標網頁，然後檢閱內容。
6. 在「複查」視窗中，輸入複查附註，例如核准或拒絕作業的原因，然後選擇「核准」或「拒絕」。
7. 選擇提交。

核准或拒絕核准請求 (CLI)

若要使用 CLI 回應核准動作，您必須先使用 get-pipeline-state 命令擷取核准動作最後一次執行的關聯字符。

1. 在終端機 (Linux、macOS 或 Unix) 或命令提示字元 (Windows) 上，在包含核准動作的管道上執行get-pipeline-state命令。例如，針對名為 *MyFirstPipeline* 的管道輸入下列內容：

```
aws codepipeline get-pipeline-state --name MyFirstPipeline
```

2. 在命令的回應中，找出 `token` 值，其位於核准動作 `actionStates` 區段的 `latestExecution` 中，如此處所示：

```
{
```

```
    "created": 1467929497.204,
    "pipelineName": "MyFirstPipeline",
    "pipelineVersion": 1,
    "stageStates": [
        {
            "actionStates": [
                {
                    "actionName": "MyApprovalAction",
                    "currentRevision": {
                        "created": 1467929497.204,
                        "revisionChangeId": "CEM7d6Tp7zfelUSLCPPwo234xEXAMPLE",
                        "revisionId": "HYGp7zmwbCPPwo23xCMdTeqIlEXAMPLE"
                    },
                    "latestExecution": {
                        "lastUpdatedBy": "identity",
                        "summary": "The new design needs to be reviewed before
 release.",
                        "token": "1a2b3c4d-573f-4ea7-a67E-XAMPLETOKEN"
                    }
                }
//More content might appear here
}
```

3. 在純文字編輯器中，以 JSON 格式來建立您將用於新增下列內容的檔案：

   - 包含核准動作的管道名稱。
   - 包含核准動作的階段名稱。
   - 核准動作的名稱。
   - 您在先前步驟中收集到的字符值。
   - 您針對動作的回應 (核准或拒絕)。回應必須以大寫表示。
   - 您的摘要註解。

   對於前述的 *MyFirstPipeline* 範例，您的檔案應類似這樣：

```
{
  "pipelineName": "MyFirstPipeline",
  "stageName": "MyApprovalStage",
  "actionName": "MyApprovalAction",
  "token": "1a2b3c4d-573f-4ea7-a67E-XAMPLETOKEN",
  "result": {
    "status": "Approved",
```

```
    "summary": "The new design looks good. Ready to release to customers."
  }
}
```

4. 以類似 **approvalstage-approved.json** 的名稱儲存檔案。
5. 執行put-approval-result命令，指定核准 JSON 檔案的名稱，如下所示：

> **Important**
>
> 請確認在檔案名稱之前包含 `file://`。這是此命令必要項目。

```
aws codepipeline put-approval-result --cli-input-json file://approvalstage-approved.json
```

手動核准通知的 JSON 資料格式 CodePipeline

對於使用 Amazon SNS 通知的核准動作，會在管道停止時建立動作的 JSON 資料，並將其發佈到 Amazon SNS。您可以使用 JSON 輸出將訊息傳送到 Amazon SQS 佇列或叫用中AWS Lambda的函數。

> **Note**
>
> 本指南不會談論如何使用 JSON 設定通知。如需相關資訊，請參閱 Amazon SNS 開發人員指南中的將 Amazon SNS 訊息傳送至 Amazon SQS 佇列和使用 Amazon SNS 通知叫用 Lambda 函數。

下列範例顯示使用 CodePipeline 核准的可用 JSON 輸出結構。

```
{
    "region": "us-east-2",
    "consoleLink": "https://console.aws.amazon.com/codepipeline/home?region=us-east-2#/view/MyFirstPipeline",
    "approval": {
        "pipelineName": "MyFirstPipeline",
        "stageName": "MyApprovalStage",
        "actionName": "MyApprovalAction",
```

```
        "token": "1a2b3c4d-573f-4ea7-a67E-XAMPLETOKEN",
        "expires": "2016-07-07T20:22Z",
        "externalEntityLink": "http://example.com",
        "approvalReviewLink": "https://console.aws.amazon.com/codepipeline/
home?region=us-east-2#/view/MyFirstPipeline/MyApprovalStage/MyApprovalAction/
approve/1a2b3c4d-573f-4ea7-a67E-XAMPLETOKEN",
        "customData": "Review the latest changes and approve or reject within seven
 days."
    }
}
```

在 CodePipeline 中新增跨區域動作

AWS CodePipeline 包含數個動作，可協助您為自動化發行程序設定建置、測試和部署資源。您可以將動作新增至管道位於與管道不同的AWS區域中的管道。如果AWS 服務是動作的提供者，且此動作類型/提供者類型與管線位於不同的「AWS區域」時，這是「跨區域」動作。

> **Note**
>
> 支援跨區AWS域動作，且只能在支援的區域中建立。CodePipeline如需支援的清單AWSCodePipeline，請參閱AWS CodePipeline 中的配額。

您可以使用主控台、AWS CLI 或 AWS CloudFormation，在管道中新增跨區域動作。

> **Note**
>
> 中的某些動作類型CodePipeline可能只適用於特定AWS地區。另請注意，可能有一些AWS區域可以使用動作類型，但該動作類型的特定AWS提供者不可用。

當您建立或編輯管道時，您必須在管道區域中擁有一個成品儲存貯體，然後對於每個您計劃執行動作的區域，都必須擁有一個成品儲存貯體。如需 `ArtifactStores` 參數的詳細資訊，請參閱CodePipeline配管結構參照。

> **Note**
>
> CodePipeline執行跨區域動作時，會處理從一個「AWS區域」複製到其他區域的人工因素。

如果您使用主控台來建立管道或跨區域動作，CodePipeline 會在您有在其中具有動作的區域中設定預設的成品儲存貯體。當您使用 AWS CLI、AWS CloudFormation 或開發套件建立管道或跨區域動作時，您會為每個您在其中具有動作的區域提供成品儲存貯體。

> **Note**
>
> 您必須在與「跨區域」動作相同的「AWS區域」中建立成品儲存貯體和加密金鑰，並在與管道相同的帳戶中建立。

您無法針對以下動作類型建立跨區域動作：

- 來源動作
- 第三方動作
- 自訂動作

> **Note**
>
> 在中使用跨區域 Lambda 叫用動作時CodePipeline，使用PutJobSuccessResult和的 lambda 執行狀態PutJobFailureResult應傳送至存在 Lambda 函數的AWS區域，而不是傳送至CodePipeline存在 Lambda 函數的區域。

當管道將跨區域動作包含為階段的一部分時，CodePipeline 只會將跨區域動作的輸入成品從管道區域複寫到動作區域。

> **Note**
>
> 管道區域和CloudWatch事件變更偵測資源所在的區域保持不變。管道託管所在的區域不會變更。

在管道中管理跨區域動作 (主控台)

您可以使用 CodePipeline 主控台，將跨區域動作新增至現有管道。若要使用建立管道精靈來建立具有跨區域動作的新管道，請參閱建立管道 (主控台)。

在主控台中，您可以選擇動作提供者和 Region (區域) 欄位 (其中列出您在該區域中為該提供者建立的資源)，在管道階段中建立跨區域動作。當您新增跨區域動作時，CodePipeline 會使用該動作的區域中不同的成品儲存貯體。如需跨區域成品儲存貯體的詳細資訊，請參閱 CodePipeline 配管結構參照。

新增跨區域動作至管道階段 (主控台)

使用主控台將跨區域動作新增至管道。

> **Note**
>
> 如果儲存變更時管道正在執行中，該項執行便不會完成。

新增跨區域動作

1. 在 http://console.aws.amazon.com/codesuite/codepipeline/home 登入主控台。
2. 選取您的管道，然後選擇 Edit (編輯)。
3. 如果您要新增新的階段，請在圖表底部選擇 + Add stage (新增階段)，或是如果您希望新增動作到現有的階段中，請選擇 Edit stage (編輯階段)。
4. 在 Edit: <Stage> (編輯：<階段>) 上，選擇 + Add action group (新增動作群組) 以新增序列動作。或者，選擇 + Add action (新增動作) 以新增平行動作。
5. 在 Edit action (編輯動作) 頁面：
   a. 在 Action name (動作名稱) 中，輸入跨區域動作的名稱。
   b. 在 Action provider (動作供應商)，選擇動作供應商。
   c. 在「區域」中，選擇您已建立或計劃為動作建立資源的「AWS區域」。當選定區域時，會列出該區域可用的資源以供選擇。[Re gion] 欄位會指定針對此動作類型和提供者類型建立AWS資源的位置。此欄位只會針對動作提供者為的動作顯示AWS 服務。「區域」(Region) 欄位預設AWS 區域為與您的管道相同。
   d. 在 Input artifacts (輸入成品) 中，選擇前一階段的適當輸入。例如，如果上一個階段是來源階段，請選擇SourceArtifact。
   e. 為您設定的動作供應商完成所有必要的欄位。
   f. 在 Output artifacts (輸出成品) 中，選擇下一階段的適當輸出。例如，如果下一個階段是部署階段，請選擇BuildArtifact。
   g. 選擇 儲存 。
6. 在 Edit: <Stage> (編輯：<階段>) 上，選擇 Done (完成)。

7. 選擇 儲存 。

編輯管道階段中的跨區域動作 (主控台)

使用主控台來編輯管道中現有的跨區域動作。

> **Note**
> 如果儲存變更時管道正在執行中，該項執行便不會完成。

如何編輯跨區域動作

1. 前往 https://console.aws.amazon.com/codesuite/codepipeline/home. 登入主控台
2. 選取您的管道，然後選擇 Edit (編輯)。
3. 選擇 Edit stage (編輯階段)。
4. 在 Edit: <Stage> (編輯：<階段>) 上，選擇圖示以編輯現有動作。
5. 在 Edit action (編輯動作) 頁面上，適當地對欄位進行變更。
6. 在 Edit: <Stage> (編輯：<階段>) 上，選擇 Done (完成)。
7. 選擇 儲存 。

從管道階段刪除跨區域動作 (主控台)

使用主控台從管道刪除現有的跨區域動作。

> **Note**
> 如果儲存變更時管道正在執行中，該項執行便不會完成。

刪除跨區域動作

1. 在 http://console.aws.amazon.com/codesuite/codepipeline/home 登入主控台。
2. 選取您的管道，然後選擇 Edit (編輯)。
3. 選擇 Edit stage (編輯階段)。

4. 在 Edit: <Stage> (編輯：<階段>) 上，選擇圖示以刪除現有動作。
5. 在 Edit: <Stage> (編輯：<階段>) 上，選擇 Done (完成)。
6. 選擇 儲存 。

將跨區域動作新增至管道 (CLI)

您可以使用 AWS CLI，將跨區域動作新增至現有管道。

若要使用 AWS CLI 在管道階段中建立跨區域動作，請新增組態動作以及選用的 `region` 欄位。您亦須已在動作的區域中建立成品儲存貯體。您不用提供單一區域管道的 `artifactStore` 參數，而是利用 `artifactStores` 參數，包含每個區域的成品儲存貯體清單。

> **Note**
>
> 在這個逐步解說及其範例中，*RegionA* 是建立管道的區域。它可以存取用於存放管道成品的 *RegionA* Amazon S3 儲存貯體以及所使用的服務角色CodePipeline。*RegionB* 是建立 CodeDeploy 應用程式、部署群組，以及 CodeDeploy 所使用之服務角色的區域。

先決條件

您必須已建立下列項目：

- *RegionA* 中的管道。
- *RegionB* 中的 Amazon S3 神器存儲桶。
- *RegionB ##################CodeDeploy#########*。

將跨區域動作新增至管道 (CLI)

使用 AWS CLI 將跨區域動作新增至管道。

新增跨區域動作

1. 對於 *RegionA* 中的管道，執行 get-pipeline 命令，以將管道結構複製到 JSON 檔案。例如，針對名為 `MyFirstPipeline` 的管道，執行下列命令：

```
aws codepipeline get-pipeline --name MyFirstPipeline >pipeline.json
```

此命令不會傳回任何內容，但您建立的檔案應該會顯示在您執行命令的目錄中。

2. 新增 region 欄位來新增階段與跨區域動作，其中包括動作的區域和資源。下列 JSON 範本會新增部署階段與跨區域部署動作，其中提供者是 CodeDeploy，位於新的區域 us-east-1 中。

```
{
                "name": "Deploy",
                "actions": [
                    {
                        "inputArtifacts": [
                            {
                                "name": "SourceArtifact"
                            }
                        ],
                        "name": "Deploy",
                        "region": "RegionB",
                        "actionTypeId": {
                            "category": "Deploy",
                            "owner": "AWS",
                            "version": "1",
                            "provider": "CodeDeploy"
                        },
                        "outputArtifacts": [],
                        "configuration": {
                            "ApplicationName": "name",
                            "DeploymentGroupName": "name"
                        },
                        "runOrder": 1
                    }
```

3. 在管道結構中，移除 artifactStore 欄位，並為新的跨區域動作新增 artifactStores 對應。對映必須包含您執行動作之每個「AWS區域」的項目。對於映射中的每個項目，資源必須位於個別AWS區域中。在以下範例中，ID-A 是 *RegionA* 的加密金鑰 ID，而 ID-B 是 *RegionB* 的加密金鑰 ID。

```
"artifactStores":{
   "RegionA":{
      "encryptionKey":{
         "id":"ID-A",
         "type":"KMS"
      },
      "location":"Location1",
```

```
      "type":"S3"
   },
   "RegionB":{
      "encryptionKey":{
         "id":"ID-B",
         "type":"KMS"
      },
      "location":"Location2",
      "type":"S3"
   }
}
```

以下 JSON 範例將 us-west-2 儲存貯體顯示為 `my-storage-bucket`，並新增命名 `my-storage-bucket-us-east-1` 的 us-east-1 儲存貯體。

```
        "artifactStores": {
            "us-west-2": {
                "type": "S3",
                "location": "my-storage-bucket"
            },
            "us-east-1": {
                "type": "S3",
                "location": "my-storage-bucket-us-east-1"
            }
        },
```

4. 如果您使用的是使用 get-pipeline 命令擷取的管道結構，請從 JSON 檔案中移除 `metadata` 行。否則，update-pipeline 命令無法使用它。移除 `"metadata": { }` 行，以及 `"created"`、`"pipelineARN"` 和 `"updated"` 欄位。

   例如，從結構中移除下列幾行：

```
"metadata": {
  "pipelineArn": "arn:aws:codepipeline:region:account-ID:pipeline-name",
  "created": "date",
  "updated": "date"
  }
```

   儲存檔案。

5. 若要套用您的變更，請執行 update-pipeline 命令、指定管道 JSON 檔案：

> **Important**
>
> 請確認在檔案名稱之前包含 `file://`。這是此命令必要項目。

```
aws codepipeline update-pipeline --cli-input-json file://pipeline.json
```

此命令會傳回所編輯管道的整個結構。輸出類似如下。

```
{
    "pipeline": {
        "version": 4,
        "roleArn": "ARN",
        "stages": [
            {
                "name": "Source",
                "actions": [
                    {
                        "inputArtifacts": [],
                        "name": "Source",
                        "actionTypeId": {
                            "category": "Source",
                            "owner": "AWS",
                            "version": "1",
                            "provider": "CodeCommit"
                        },
                        "outputArtifacts": [
                            {
                                "name": "SourceArtifact"
                            }
                        ],
                        "configuration": {
                            "PollForSourceChanges": "false",
                            "BranchName": "main",
                            "RepositoryName": "MyTestRepo"
                        },
                        "runOrder": 1
                    }
                ]
            },
            {
```

```
                "name": "Deploy",
                "actions": [
                    {
                        "inputArtifacts": [
                            {
                                "name": "SourceArtifact"
                            }
                        ],
                        "name": "Deploy",
                        "region": "us-east-1",
                        "actionTypeId": {
                            "category": "Deploy",
                            "owner": "AWS",
                            "version": "1",
                            "provider": "CodeDeploy"
                        },
                        "outputArtifacts": [],
                        "configuration": {
                            "ApplicationName": "name",
                            "DeploymentGroupName": "name"
                        },
                        "runOrder": 1
                    }
                ]
            }
        ],
        "name": "AnyCompanyPipeline",
        "artifactStores": {
            "us-west-2": {
                "type": "S3",
                "location": "my-storage-bucket"
            },
            "us-east-1": {
                "type": "S3",
                "location": "my-storage-bucket-us-east-1"
            }
        }
    }
}
```

> **Note**
>
> update-pipeline 命令將終止管道。若在您執行 update-pipeline 命令時有修訂正在透過管道執行，該執行將停止。您必須手動啟動管道，以透過更新的管道執行該修訂。使用 **start-pipeline-execution** 命令來手動啟動您的管道。

6. 更新管線後，「跨區域」動作會顯示在主控台中。

將跨區域動作新增至管道 (AWS CloudFormation)

您可以使用 AWS CloudFormation，將跨區域動作新增至現有管道。

如何使用 AWS CloudFormation 新增跨區域動作

1. 將 `Region` 參數新增到範本中的 `ActionDeclaration` 資源，如以下範例所示：

```
ActionDeclaration:
              Type: Object
              Properties:
                ActionTypeId:
                  Type: ActionTypeId
                  Required: true
                Configuration:
                  Type: Map
                InputArtifacts:
                  Type: Array
                  ItemType:
```

```
                Type: InputArtifact
              Name:
                Type: String
                Required: true
              OutputArtifacts:
                Type: Array
                ItemType:
                  Type: OutputArtifact
              RoleArn:
                Type: String
              RunOrder:
                Type: Integer
              Region:
                Type: String
```

2. 在 Mappings 下，新增區域地圖，如這個範例所示，其中名為 SecondRegionMap 的映射會對映金鑰 RegionA 和 RegionB 的值。在 Pipeline 資源下，於 artifactStore 欄位下，為新的跨區域動作新增 artifactStores 對應，如下所示：

```
Mappings:
  SecondRegionMap:
    RegionA:
      SecondRegion: "RegionB"
    RegionB:
      SecondRegion: "RegionA"

...

          Properties:
            ArtifactStores:
              -
                Region: RegionB
                ArtifactStore:
                  Type: "S3"
                  Location: test-cross-region-artifact-store-bucket-RegionB
              -
                Region: RegionA
                ArtifactStore:
                  Type: "S3"
                  Location: test-cross-region-artifact-store-bucket-RegionA
```

下列 YAML 範例會將 *RegionA* 儲存貯體顯示為 us-west-2，並新增 *RegionB* 儲存貯體 eu-central-1：

```
Mappings:
  SecondRegionMap:
    us-west-2:
      SecondRegion: "eu-central-1"
    eu-central-1:
      SecondRegion: "us-west-2"

...

          Properties:
            ArtifactStores:
              -
                Region: eu-central-1
                ArtifactStore:
                  Type: "S3"
                  Location: test-cross-region-artifact-store-bucket-eu-central-1
              -
                Region: us-west-2
                ArtifactStore:
                  Type: "S3"
                  Location: test-cross-region-artifact-store-bucket-us-west-2
```

3. 將更新的範本儲存到本機電腦，然後開啟 AWS CloudFormation 主控台。
4. 選擇您的堆疊，然後選擇 Create Change Set for Current Stack (建立目前堆疊的變更集)。
5. 上傳範本，然後檢視 AWS CloudFormation 中所列的變更。這些是會針對堆疊進行的變更。您應該會在清單中看到新資源。
6. 選擇 Execute (執行)。

使用變數

CodePipeline 產生變數中的某些動作。若要使用變數：

- 請將命名空間指派給動作，使該動作產生的變數可供下游動作組態使用。
- 請設定下游動作來取用該動作所產生的變數。

您可以檢視每個動作執行的詳細資料，以查看動作在執行階段產生的每個輸出變數的值。

若要查看 step-by-step 使用變數的範例：

- 如需具有 Lambda 動作的教學課程，該動作使用上游動作 (CodeCommit) 中的變數並產生輸出變數，請參閱教學課程：使用變數搭配 Lambda 呼叫動作。
- 如需具有參考上游AWS CloudFormation動作之堆疊輸出變數之 CloudFormation 動作的自學課程，請參閱〈〉教學：建立透過 AWS CloudFormation 部署動作使用變數的管道。
- 如需含有訊息文字的手動核准動作範例，其中參考解析為 CodeCommit 提交 ID 和提交訊息的輸出變數，請參閱範例：在手動核准中使用變數。
- 如需具有可解析為 GitHub 分支名稱之環境變數的範例 CodeBuild 動作，請參閱範例：搭配 CodeBuild 環境 BranchName變數使用變數。
- CodeBuild 動作會以變數形式產生所有已匯出為組建一部分的環境變數。如需詳細資訊，請參閱 CodeBuild 動作輸出變數。如需可在中使用的環境變數清單 CodeBuild，請參閱《使AWS CodeBuild用指南》中的建置環境中的環境變數。

主題

設定變數的動作

當您將動作新增至管道時，您可以指派命名空間給此動作，並設定此動作使用先前動作的變數。

使用變數設定動作 (主控台)

此範例會建立包含 CodeCommit 來源動作和 CodeBuild 建置動作的管線。 CodeBuild 動作會設定為使用動 CodeCommit 作所產生的變數。

如果未指定命名空間，則變數無法供動作組態中參考。當您使用主控台建立管道時，系統會自動產生每個動作的命名空間。

建立具有變數的管道

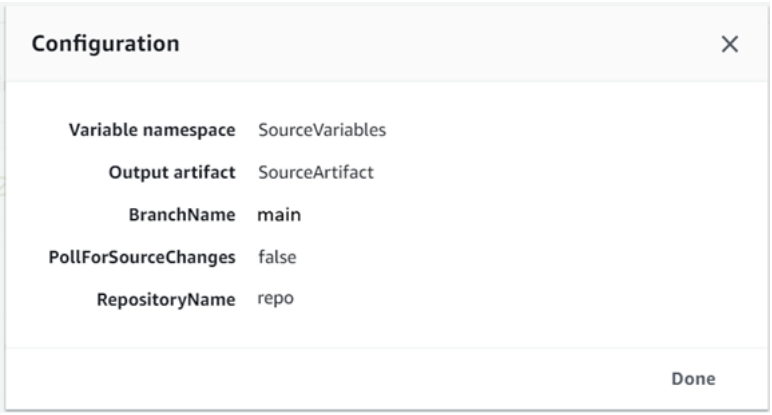
1. 請登入AWS Management Console並開啟 CodePipeline 主控台，網址為 http://console.aws.amazon.com/codesuite/codepipeline/home。

2. 選擇 Create pipeline (建立管道)。輸入管道的名稱，然後選擇 Next (下一步)。
3. 在來源中的提供者中，選擇CodeCommit。選擇來源動作的 CodeCommit 存放庫和分支，然後選擇 [下一步]。
4. 在組建中的提供者中，選擇CodeBuild。選擇現有的 CodeBuild 組建專案名稱或選擇 [建立專案]。在 [建立組建專案] 上，建立組建專案，然後選擇 [返回至] CodePipeline。

   在 Environment variables (環境變數) 下，選擇 Add environment variables (新增環境變數)。例如，輸入帶有變數語法的執行 ID，`#{codepipeline.PipelineExecutionId}`並使用變數語法輸入提交 ID`#{SourceVariables.CommitId}`。

   > **Note**
   >
   > 您可以在精靈的任何動作組態欄位中輸入變數語法。

5. 選擇 Create (建立)。
6. 管道建立之後，您可以檢視精靈所建立的命名空間。在管線上，選擇您要檢視其命名空間之階段的圖示。在此範例中，將會顯示來源動作自動產生的命名空間 `SourceVariables`。



編輯現有動作的命名空間

1. 請登入AWS Management Console並開啟 CodePipeline 主控台，網址為 http://console.aws.amazon.com/codesuite/codepipeline/home。
2. 選擇您要編輯的管道，然後選擇 Edit (編輯)。針對來源階段，選擇 Edit stage (編輯階段)。新增動 CodeCommit 作。
3. 在 Edit action (編輯動作) 上，檢視 Variable namespace (變數命名空間) 欄位。如果現有的動作是先前建立，或不是使用精靈來建立，您必須新增命名空間。在 Variable namespace (變數命名空間) 中，輸入命名空間名稱，然後選擇 Save (儲存)。

檢視輸出變數

1. 請登入AWS Management Console並開啟 CodePipeline 主控台，網址為 http://console.aws.amazon.com/codesuite/codepipeline/home。
2. 建立管道並成功執行之後，您可以在 Action execution details (動作執行詳細資訊) 頁面上檢視變數。如需相關資訊，請參閱 檢視變數 (主控台)。

設定變數的動作 (CLI)

當您使用 create-pipeline 命令建立管道，或使用 update-pipeline 命令編輯管道時，您可以在動作的組態中參考/使用變數。

如果未指定命名空間，則無法在任何下游動作組態中參考該動作產生的變數。

使用命名空間來設定動作

1. 遵循在中建立管線 CodePipeline中的步驟，使用 CLI 建立管道。啟動輸入檔案以提供 `--cli-input-json` 參數給 create-pipeline 命令。在管道結構中，新增 `namespace` 參數並指定名稱，例如 `SourceVariables`。

```
. . .
{
          "inputArtifacts": [],
          "name": "Source",
          "region": "us-west-2",
          "namespace": "SourceVariables",
          "actionTypeId": {
            "category": "Source",
            "owner": "AWS",
            "version": "1",
            "provider": "CodeCommit"
          },
          "outputArtifacts": [

. . .
```

2. 以類似 **`MyPipeline.json`** 的名稱儲存檔案。
3. 在終端機 (Linux、macOS 或 Unix) 或命令提示 (Windows)，執行create-pipeline命令。

   呼叫您執行 create-pipeline 命令時建立的檔案。例如：

```
aws codepipeline create-pipeline --cli-input-json file://MyPipeline.json
```

設定下游動作來取用變數

1. 編輯輸入檔案以提供 --cli-input-json 參數給 update-pipeline 命令。在下游動作中，將變數新增至該動作的組態。變數由名稱空間和索引鍵組成 (以句點分隔)。例如，若要為管道執行 ID 和來源遞交 ID 新增變數，請指定命名空間 codepipeline 給變數 #{codepipeline.PipelineExecutionId}。指定命名空間 SourceVariables 給變數 #{SourceVariables.CommitId}。

```
{
    "name": "Build",
    "actions": [
        {
            "outputArtifacts": [
                {
                    "name": "BuildArtifacts"
                }
            ],
            "name": "Build",
            "configuration": {
                "EnvironmentVariables": "[{\"name\":\"Execution_ID\",\"value
\":\"#{codepipeline.PipelineExecutionId}\",\"type\":\"PLAINTEXT\"},{\"name\":
\"Commit_ID\",\"value\":\"#{SourceVariables.CommitId}\",\"type\":\"PLAINTEXT\"}]",
                "ProjectName": "env-var-test"
            },
            "inputArtifacts": [
                {
                    "name": "SourceArtifact"
                }
            ],
            "region": "us-west-2",
            "actionTypeId": {
                "provider": "CodeBuild",
                "category": "Build",
                "version": "1",
                "owner": "AWS"
            },
            "runOrder": 1
        }
```

```
    ]
},
```

2. 以類似 **MyPipeline.json** 的名稱儲存檔案。
3. 在終端機 (Linux、macOS 或 Unix) 或命令提示 (Windows)，執行create-pipeline命令。

   呼叫您執行 create-pipeline 命令時建立的檔案。例如：

```
aws codepipeline create-pipeline --cli-input-json file://MyPipeline.json
```

檢視輸出變數

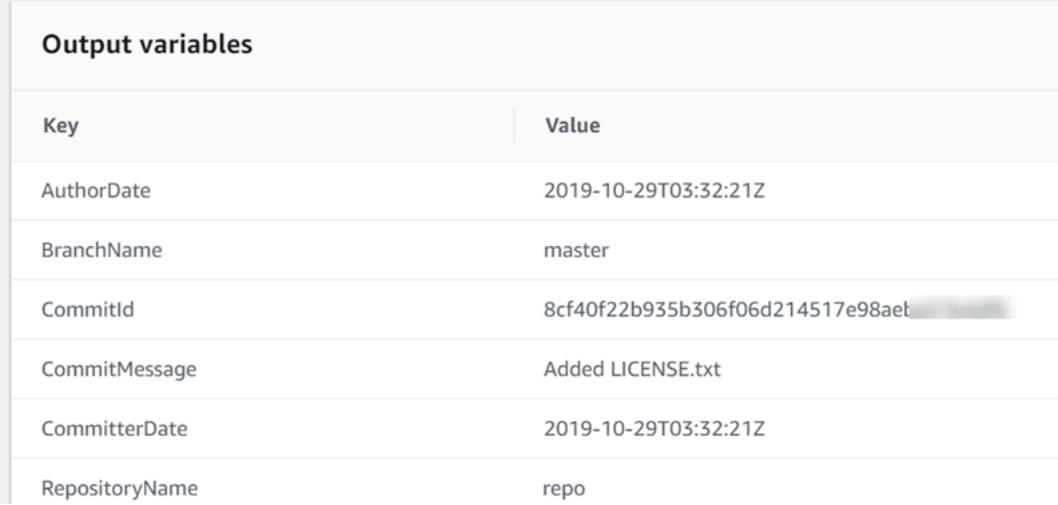
您可以檢視動作執行詳細資訊，以檢視該動作的變數 (每個執行所特有)。

檢視變數 (主控台)

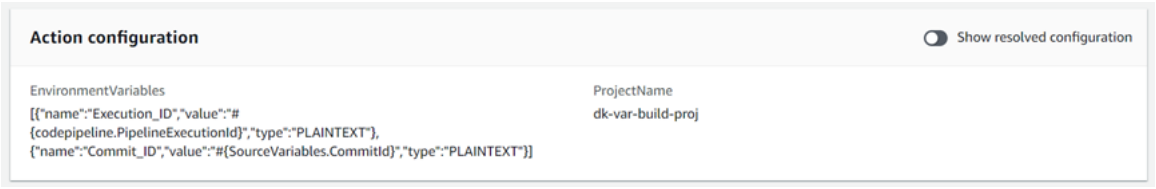
您可以使用主控台來檢視動作的變數。

1. 請登入AWS Management Console並開啟 CodePipeline 主控台，網址為 http://console.aws.amazon.com/codesuite/codepipeline/home。

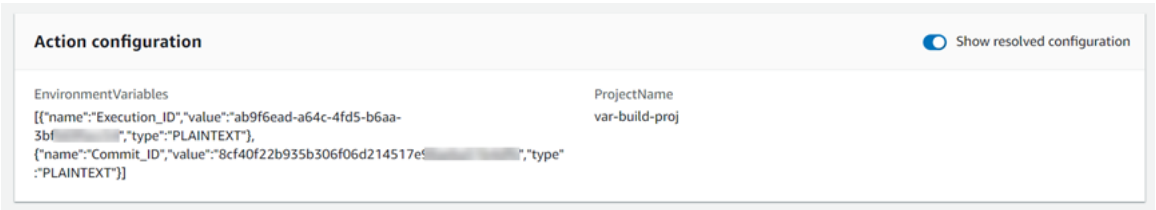
   所有與您的 AWS 帳戶相關的管道名稱都會顯示。
2. 在 Name (名稱) 中，選擇管道的名稱。
3. 選擇 View history (檢視歷程記錄)。
4. 管道成功執行後，您可以檢視來源動作所產生的變數。選擇 View history (檢視歷程記錄)。在管線執行的動作清單中選擇「來源」，以檢視動作的動作執行詳細資訊。 CodeCommit 在動作詳細資訊畫面上，檢視 Output variables (輸出變數) 下的變數。



5. 管道成功執行後，您可以檢視建置動作所取用的變數。選擇 View history (檢視歷程記錄)。在管線執行的動作清單中，選擇「建置」以檢視動作的動作執行詳細資訊。 CodeBuild 在動作詳細資訊頁面上，檢視 Action configuration (動作組態) 下的變數。將會顯示自動產生的命名空間。



根據預設，Action configuration (動作組態) 會顯示變數語法。您可以選擇 Show resolved configuration (顯示解析的組態)，以切換清單來顯示動作執行期間產生的值。



檢視變數 (CLI)

您可以使用 list-action-executions 命令來檢視動作的變數。

1. 使用下列命令：

```
aws codepipeline list-action-executions
```

輸出會顯示 `outputVariables` 參數，如下所示。

```
"outputVariables": {
                    "BranchName": "main",
                    "CommitMessage": "Updated files for test",
                    "AuthorDate": "2019-11-08T22:24:34Z",
                    "CommitId": "d99b0083cc10EXAMPLE",
                    "CommitterDate": "2019-11-08T22:24:34Z",
                    "RepositoryName": "variables-repo"
                },
```

2. 使用下列命令：

```
aws codepipeline get-pipeline --name <pipeline-name>
```

在動作的動作組態中 CodeBuild ，您可以檢視變數：

```
{
    "name": "Build",
    "actions": [
        {
            "outputArtifacts": [
                {
                    "name": "BuildArtifact"
                }
            ],
            "name": "Build",
            "configuration": {
                "EnvironmentVariables": "[{\"name\":\"Execution_ID\",\"value
\":\"#{codepipeline.PipelineExecutionId}\",\"type\":\"PLAINTEXT\"},{\"name\":
\"Commit_ID\",\"value\":\"#{SourceVariables.CommitId}\",\"type\":\"PLAINTEXT\"}]",
                "ProjectName": "env-var-test"
            },
            "inputArtifacts": [
                {
                    "name": "SourceArtifact"
                }
            ],
            "region": "us-west-2",
            "actionTypeId": {
                "provider": "CodeBuild",
                "category": "Build",
```

```
                "version": "1",
                "owner": "AWS"
            },
            "runOrder": 1
        }
    ]
},
```

範例：在手動核准中使用變數

當您指定動作的命名空間，而該動作會產生輸出變數時，您可以新增手動核准，在核准訊息中顯示變數。此範例示範如何將變數語法新增至手動核准訊息。

1. 請登入AWS Management Console並開啟 CodePipeline 主控台，網址為 http://console.aws.amazon.com/codesuite/codepipeline/home。

   所有與您的 AWS 帳戶相關的管道名稱都會顯示。選擇您要新增核准的管道。

2. 若要編輯管道，請選擇 Edit (編輯)。在來源動作之後新增手動核准。在 Action name (動作名稱) 中，輸入核准動作的名稱。
3. 在 Action provider (動作提供者) 中，選擇 Manual approval (手動核准)。
4. 在要檢閱的 URL 中，將變數語法新增`CommitId`至您的 CodeCommit URL。請確定您使用指派給來源動作的命名空間。例如，具有預設命名空間 `SourceVariables` 的 CodeCommit 動作，變數語法為 `#{SourceVariables.CommitId}`。

   在註解中`CommitMessage`，輸入提交訊息：

   ```
   Please approve this change. Commit message: #{SourceVariables.CommitMessage}
   ```

5. 管道成功執行後，您可以檢視核准訊息中的變數值。

範例：搭配 CodeBuild 環境 BranchName變數使用變數

將 CodeBuild 動作新增至管線時，可以使用 CodeBuild 環境變數來參照上游來源動作的`BranchName`輸出變數。使用中動作的輸出變數 CodePipeline，您可以建立自己的 CodeBuild環境變數，以便在建置指令中使用。

此範例說明如何將來 GitHub 源動作的輸出變數語法新增至 CodeBuild 環境變數。此範例中的輸出變數語法代表的 GitHub 來源動作輸出變數`BranchName`。動作成功執行之後，變數會解析以顯示 GitHub 分支名稱。

1. 請登入AWS Management Console並開啟 CodePipeline 主控台，網址為 http://console.aws.amazon.com/codesuite/codepipeline/home。

   所有與您的 AWS 帳戶相關的管道名稱都會顯示。選擇您要新增核准的管道。
2. 若要編輯管道，請選擇 Edit (編輯)。在包含 CodeBuild 動作的舞台上，選擇 「編輯階段」。
3. 選擇圖示以編輯 CodeBuild 動作。
4. 在「編輯」動作頁面的「環境變數」下，輸入下列內容：

   - 在 Name (((名稱) 中，輸入環境變數的名稱。
   - 在值中，輸入管線輸出變數的變數語法，其中包括指派給來源動作的命名空間。例如，具有預設命名空間之 GitHub動作的輸出變數語法`SourceVariables`為`#{SourceVariables.BranchName}`。
   - 在「類型」中，選擇「純文字」。
5. 管線成功執行之後，您可以看到解析的輸出變數是環境變數中的值。選擇下列其中一項：

   - CodePipeline 控制台：選擇您的管道，然後選擇歷史記錄。選擇最近的管線執行。
     - 在「時間軸」下，選擇「來源」的選取器。這是產生 GitHub 輸出變數的來源動作。選擇檢視執行詳細資訊。在「輸出變數」下，檢視此動作所產生的輸出變數清單。
     - 在「時間軸」下，選擇「組建」的選取器。這是構建操作，用於指定構建項目的 CodeBuild 環境變量。選擇檢視執行詳細資訊。在動作組態下，檢視您的 CodeBuild環境變數。選擇顯示已解析的組態。您的環境變數值是來自 GitHub 來源動作的已解析`BranchName`輸出變數。在此範例中，解析值為`main`。

       如需詳細資訊，請參閱 檢視變數 (主控台)。
   - CodeBuild 控制台：選擇您的構建項目並選擇構建運行的鏈接。在環境變量下，解析的輸出變量是 CodeBuild 環境變量的值。在此範例中，環境變數 Name 為，`BranchName`而 Value 是 GitHub 來源動作中已解析的`BranchName`輸出變數。在此範例中，解析值為`main`。


Build logs
Phase details
Reports
Environment variables
Build details
Resource utilization
Name
Value
Type
BranchName
main
PLAINTEXT

在 CodePipeline 中使用階段轉換

轉換為可停用或啟用的管道階段之間的連結。預設為皆啟用。當您重新啟用已停用的轉換，最新修訂會在管道中所有剩餘階段內執行，除非已超過 30 天。管道執行不會繼續已停用超過 30 天的轉換，除非偵測到新的更改或者您以手動重新執行管道。

您可以使用 AWS CodePipeline 主控台或 AWS CLI 來停用或啟用管道內階段之間的轉換。

> **Note**
>
> 您可以使用核准動作來暫停管道的執行，直到經過手動核准可繼續進行。如需詳細資訊，請參閱 管理中的核准動作 CodePipeline。

主題

停用或啟用轉換 (主控台)

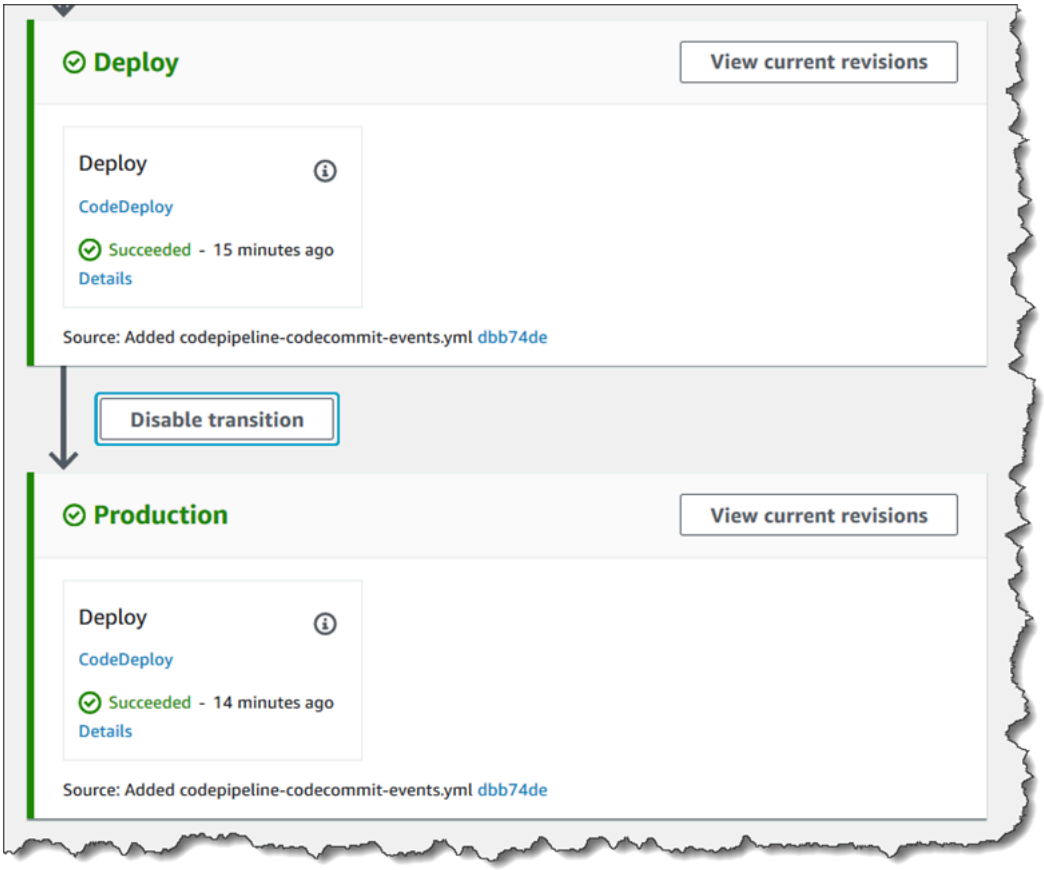
若要停用或啟用管道中的轉換

1. 前往登入AWS Management Console並開啟 CodePipeline 主控台http://console.aws.amazon.com/codesuite/codepipeline/home。

   所有與您的 AWS 帳戶相關的管道名稱都會顯示。

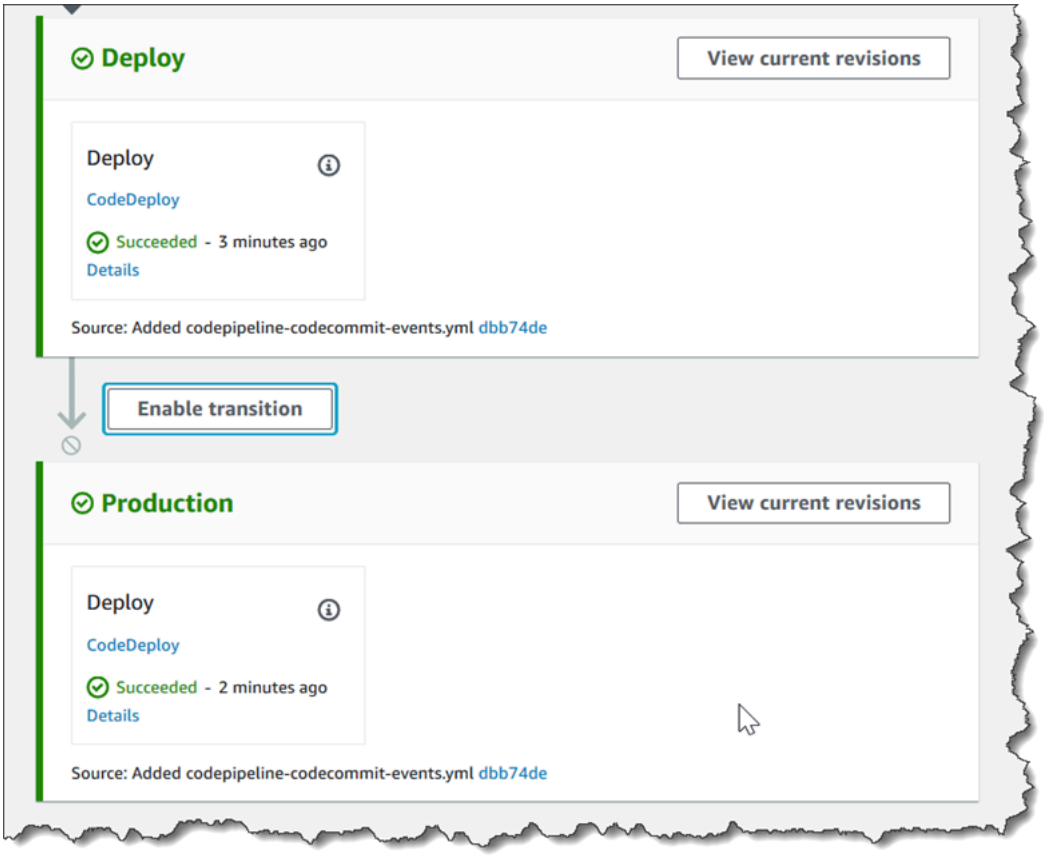
2. 在 Name (名稱) 中，選擇您想要啟用或停用轉換的管道之名稱。這會開啟管道的詳細檢視，包含管道各階段之間的轉換。

3. 尋找您想要執行的最後階段後方的箭號，然後選擇它旁邊的按鈕。例如，在下列範例管道中，若您想要讓 Staging (暫存) 階段中的動作可執行，而非讓名為 Production (生產) 的階段內的動作執行，您需要選擇兩個階段之間的 Disable transition (停用轉換) 按鈕：



4. 在 Disable transition (停用轉換) 對話方塊中，輸入停用轉換的原因，然後選擇 Disable (停用)。

   該按鈕會改變，以顯示在箭號前方的階段以及箭號後方的階段之間停用的轉換。任何在停用階段之後、已在階段中執行的修訂會繼續在管道中進行，但是在經歷停用的轉換後，任何之後發生的修訂都不會繼續。



5. 選擇箭號旁的 Enable transition (啟用轉換) 按鈕。在 Enable transition (啟用轉換) 對話方塊中，選擇 Enable (啟用)。管道會立即啟用兩個階段之間的轉換。若在轉換停用後，有任何在稍早階段中執行的修訂，管道將在之前停用的轉換之後，開始在階段之間執行最新修訂版。管道會在管道內的所有剩餘階段間執行修訂版。

> **Note**
>
> 在您啟用轉換後，可能會需要幾秒時間，變更才會顯示在 CodePipeline 主控台中。

停用或啟用轉換 (CLI)

若要使用 AWS CLI 來停用階段之間的轉換，請執行 disable-stage-transition 命令。若要啟用已停用的轉換，請執行 enable-stage-transition 命令。

停用轉換

1. 開啟終端機 (Linux、macOS 或 Unix) 或命令提示字元 (Windows)，然後使用AWS CLI執行殘疾階段過渡命令、指定管道名稱、您想要停用轉換的階段名稱、轉換類型、以及您要停用轉換到該階段

的原因。與使用主控台不同，您也必須指定是否將停用轉換到階段 (輸入) 或者停用在所有動作完成後從該階段向外的轉換 (輸出)。

例如，要禁用轉換到名為*##*管道中名為*MyFirstPipeline*，您可能會執行與下列類似的命令：

```
aws codepipeline disable-stage-transition --pipeline-name MyFirstPipeline --stage-
name Staging --transition-type Inbound --reason "My Reason"
```

此命令不會傳回任何結果。

2. 若要確認已停用的轉換，可在 CodePipeline 主控台中檢視管道或者執行get-pipeline-state命令。如需更多詳細資訊，請參閱 檢視管線 (主控台) 及 檢視管道詳細資訊與歷程記錄 (CLI)。

啟用轉換

1. 開啟終端機 (Linux、macOS 或 Unix) 或命令提示字元 (Windows)，然後使用AWS CLI執行啟用階段過渡命令、指定管道名稱、您想要啟用轉換的階段名稱以及轉換類型。

   例如，要啟用轉換到名為*##*管道中名為*MyFirstPipeline*，您可能會執行與下列類似的命令：

   ```
   aws codepipeline enable-stage-transition --pipeline-name MyFirstPipeline --stage-
   name Staging  --transition-type Inbound
   ```

   此命令不會傳回任何結果。

2. 若要確認已停用的轉換，可在 CodePipeline 主控台中檢視管道或者執行get-pipeline-state命令。如需詳細資訊，請參閱 檢視管線 (主控台) 和 檢視管道詳細資訊與歷程記錄 (CLI)。

監控管道

監控是維護 AWS CodePipeline 可靠性、可用性和效能的重要環節。您應該收集 AWS 解決方案各方面的監控資料，以便在發生多點失敗的情況下，更輕鬆地偵錯。在開始監控之前，應該先建立監控計畫，以回答下列問題：

- 監控目標是什麼？
- 監控哪些資源？
- 監控這些資源的頻率為何？
- 您可以使用哪些監控工具？
- 誰將執行監控任務？
- 發生問題時應該通知誰？

您可以使用下列工具來監控 CodePipeline 管道和其資源：

- EventBridge事件匯流排事件 — 您可以監視中的CodePipeline事件EventBridge，以偵測管道、階段或動作執行狀態的變更。EventBridge將資料路由到目標，例如 AWS Lambda Amazon 簡單通知服務。EventBridge事件與亞馬遜活動中出現的CloudWatch事件相同。
- 開發人員工具主控台中管道事CodePipeline件的通知 — 您可以透過在主控台中設定的通知監控事件，然後建立 Amazon 簡單通知服務主題和訂閱。如需詳細資訊，請參閱開發人員工具主控台使用者指南中的通知是什麼。
- AWS CloudTrail— 用於擷CloudTrail取您帳戶中或代表您AWS帳戶發出CodePipeline的 API 呼叫，並將日誌檔傳送到 Amazon S3 儲存貯體。您可以選擇在交付新的日誌檔時CloudWatch發佈 Amazon SNS 通知，以便快速採取行動。
- 主控台和 CLI — 您可以使用CodePipeline主控台和 CLI 來檢視有關管線狀態或特定管線執行的詳細資料。

主題

- 監控 CodePipeline 事件
- 事件預留位置儲存貯體參考
- 使用 AWS CloudTrail 記錄 CodePipeline API 呼叫

監控 CodePipeline 事件

您可以監控中的 CodePipeline 事件 EventBridge，從您自己的應用程式、 software-as-a-service (SaaS) 應用程式和 AWS 服務. EventBridge 將資料路由到目標，例如 AWS Lambda Amazon 簡單通知服務。這些事件與 Amazon Events 中出現的 CloudWatch 事件相同，可提供描述 AWS 資源變更的近乎即時的系統事件串流。有關更多信息，請參閱什麼是 Amazon EventBridge？ 在 Amazon 用 EventBridge 戶指南。

> **Note**
>
> Amazon EventBridge 是管理您的活動的首選方式。Amazon CloudWatch 活動和 EventBridge 基礎服務和 API 相同，但 EventBridge 提供了更多功能。您在 CloudWatch 活動中所做的變更，或 EventBridge 將會出現在每個主控台中。

事件由規則組成。您可以選擇下列項目來設定規則：

- 事件模式。每個規則都以事件模式表示，其中包含要監視的事件來源和類型，以及事件目標。若要監視事件，請使用您監視的服務建立規則作為事件來源，例如 CodePipeline。例如，您可以建立具有事件模式的規則，該規則用 CodePipeline 作事件來源，以在管線、階段或動作的狀態發生變更時觸發規則。
- 目標。新的規則收到選取的服務做為事件目標。您可能想要設定目標服務來傳送通知、擷取狀態資訊、採取更正動作、起始事件或採取其他動作。新增目標時，您還必須授與權限，以 EventBridge 允許它呼叫選取的目標服務。

每種類型的執行狀態變更事件都會發出具有特定訊息內容的通知，其中：

- 初始`version`項目會顯示事件的版本號碼。
- `detail` 管道下的 `version` 項目會顯示管道結構版本號碼。
- `detail` 管道下的 `execution-id` 項目會顯示導致狀態變更之管道執行的執行 ID。請參閱 GetPipelineExecution API 參考資料中的AWS CodePipeline API 呼叫。
- `pipeline-execution-attempt`此項目會顯示特定執行 ID 的嘗試次數或重試次數。

CodePipeline EventBridge 每當您變更資源的狀態時，都會報告事 AWS 帳戶 件。事件會保證發出，以下列資源為 at-least-once 基礎：

- 管道執行
- 階段執行
- 動作執行

事件由 EventBridge 上述事件模式和結構描述發出。對於已處理的事件，例如您透過在 Developer Tools 主控台中設定的通知接收的事件，事件訊息包含具有某些變化的事件模式欄位。例如，`detail-type`欄位會轉換為`detailType`。如需詳細資訊，請參閱 Amazon PutEvents API 參考中的 EventBridge API 呼叫。

下列範例顯示的事件 CodePipeline。在可能的情況下，每個範例都會顯示已發射事件的結構描述以及已處理事件的結構描述。

主題

詳細資訊類型

當您設定要監視的事件時，您可以選擇事件的詳細資料類型。

您可以設定要在下列項目的狀態變更時傳送的通知：

- 指定的管道或您的所有管道。使用 `"detail-type": "CodePipeline Pipeline Execution State Change"`，即可控制此項目。
- 指定的階段或您的所有階段，位於指定的管道或您的所有管道內。使用 `"detail-type": "CodePipeline Stage Execution State Change"`，即可控制此項目。
- 指定的動作或所有動作，位在指定管道或您所有管道的指定階段或所有階段內。使用 `"detail-type": "CodePipeline Action Execution State Change"`，即可控制此項目。

> **Note**
>
> 所發出的事件 EventBridge 包含`detail-type`參數，該參數會在處理事件`detailType`時轉換為。

詳圖類型	州	描述
CodePipeline 管線執行狀態變更	CANCELED	已取消管道執行，因為已更新管道結構。
	失敗	管道執行未成功完成。
	RESUMED (繼續)	已重試失敗的管道執行，以回應 `RetryStageExecution` API 呼叫。
	STARTED (已啟動)	管道執行目前正在執行。
	已停止	停止程序已完成，且管道執行已停止。
	停止中	由於要求停止並等待或停止並捨棄管道執行，因此管道執行正在停止中。
	SUCCEEDED (成功)	管道執行已成功完成。
	SUPERSEDED (已取代)	雖然此管道執行等待下一個階段完成，但較新的管道執行已改為透過管道前進並繼續。
CodePipeline 階段執行狀態變更	CANCELED	已取消階段，因為已更新管道結構。
	失敗	階段未成功完成。
	RESUMED (繼續)	已重試失敗的階段，以回應 `RetryStageExecution` API 呼叫。
	STARTED (已啟動)	階段目前正在執行。
	已停止	停止程序已完成，且階段執行已停止。

詳圖類型	州	描述
	停止中	由於要求停止並等待或停止並捨棄管道執行，因此階段執行正在停止中。
	SUCCEEDED (成功)	階段已成功完成。
CodePipeline 動作執行狀態變更	放棄	由於要求停止並捨棄管道執行，因而捨棄動作。
	CANCELED	已取消動作，因為已更新管道結構。
	失敗	對於核准動作，FAILED (失敗) 狀態表示檢閱者拒絕動作，或因動作組態不正確而失敗。
	STARTED (已啟動)	動作目前正在執行。
	SUCCEEDED (成功)	動作已成功完成。

管線層級事件

當管線執行的狀態發生變更時，就會發出管線層級事件。

主題

- 管道啟動事件
- 管道停止事件
- 管線成功事件
- 管道成功（使用 Git 標籤示例）
- 管線失敗事件
- 管道失敗（使用 Git 標籤的示例）

管道啟動事件

當管線執行開始時，它會發出一個事件，該事件會傳送包含下列內容的通知。此範例適用於在「us-east-1區域」"myPipeline" 中命名的配管。此id欄位代表事件 ID，而account欄位則代表建立管線的帳戶 ID。

Emitted event

```
{
    "version": "0",
    "id": "01234567-EXAMPLE",
    "detail-type": "CodePipeline Pipeline Execution State Change",
    "source": "aws.codepipeline",
    "account": "123456789012",
    "time": "2020-01-24T22:03:07Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:codepipeline:us-east-1:123456789012:myPipeline"
    ],
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:49:39.208Z",
        "execution-trigger": {
            "trigger-type": "StartPipelineExecution",
            "trigger-detail": "arn:aws:sts::123456789012:assumed-role/Admin/my-user"
        },
        "state": "STARTED",
        "version": 1,
        "pipeline-execution-attempt": 1
    }
}
```

Processed event

```
{
    "account": "123456789012",
    "detailType": "CodePipeline Pipeline Execution State Change",
    "region": "us-east-1",
    "source": "aws.codepipeline",
    "time": "2021-06-24T00:44:50Z",
    "notificationRuleArn": "arn:aws:codestar-notifications:us-
east-1:123456789012:notificationrule/a69c62c21EXAMPLE",
```

```
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:49:39.208Z",
        "execution-trigger": {
            "trigger-type": "StartPipelineExecution",
            "trigger-detail": "arn:aws:sts::123456789012:assumed-role/Admin/my-user"
        },
        "state": "STARTED",
        "version": 1,
        "pipeline-execution-attempt": 1
    },
    "resources": [
        "arn:aws:codepipeline:us-east-1:123456789012:myPipeline"
    ],
    "additionalAttributes": {}
}
```

管道停止事件

當管線執行停止時，它會發出一個事件，該事件會傳送包含下列內容的通知。此範例適用於在「us-west-2區域」myPipeline 中命名的配管。

```
{
    "version": "0",
    "id": "01234567-EXAMPLE",
    "detail-type": "CodePipeline Pipeline Execution State Change",
    "source": "aws.codepipeline",
    "account": "123456789012",
    "time": "2020-01-24T22:02:20Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:49:39.208Z",
        "state": "STOPPING",
        "version": 3,
        "pipeline-execution-attempt": 1
        "stop-execution-comments": "Stopping the pipeline for an update"
```

```
    }
}
```

管線成功事件

當管道執行成功時，它會發出一個事件，該事件會傳送包含下列內容的通知。此範例適用於在「us-east-1區域」myPipeline 中命名的配管。

Emitted event

```
{
    "version": "0",
    "id": "01234567-EXAMPLE",
    "detail-type": "CodePipeline Pipeline Execution State Change",
    "source": "aws.codepipeline",
    "account": "123456789012",
    "time": "2020-01-24T22:03:44Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:codepipeline:us-east-1:123456789012:myPipeline"
    ],
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:49:39.208Z",
        "state": "SUCCEEDED",
        "version": 3,
        "pipeline-execution-attempt": 1
    }
}
```

Processed event

```
{
    "account": "123456789012",
    "detailType": "CodePipeline Pipeline Execution State Change",
    "region": "us-east-1",
    "source": "aws.codepipeline",
    "time": "2021-06-30T22:13:51Z",
    "notificationRuleArn": "arn:aws:codestar-notifications:us-
west-2:123456789012:notificationrule/a69c62c21EXAMPLE",
    "detail": {
        "pipeline": "myPipeline",
```

```
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:49:39.208Z",
        "state": "SUCCEEDED",
        "version": 1,
        "pipeline-execution-attempt": 1
    },
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "additionalAttributes": {}
}
```

管道成功（使用 Git 標籤示例）

當管線執行具有已重試並成功的階段時，它會發出一個事件，該事件會傳送包含下列內容的通知。此範例適用於myPipeline在針對 Git 標籤設定的eu-central-1execution-trigger區域中命名的管線。

> **Note**
>
> 該execution-trigger字段將具有tag-name或branch-name，具體取決於觸發管道的事件類型。

```
{
    "version": "0",
    "id": "b128b002-09fd-4574-4eba-27152726c777",
    "detail-type": "CodePipeline Pipeline Execution State Change",
    "source": "aws.codepipeline",
    "account": "123456789012",
    "time": "2023-10-26T13:50:53Z",
    "region": "eu-central-1",
    "resources": [
        "arn:aws:codepipeline:eu-central-1:123456789012:BuildFromTag"
    ],
    "detail": {
        "pipeline": "BuildFromTag",
        "execution-id": "e17b5773-cc0d-4db2-9ad7-594c73888de8",
        "start-time": "2023-10-26T13:49:39.208Z",
        "execution-trigger": {
            "author-display-name": "Mary Major",
```

```
            "full-repository-name": "mmajor/sample-project",
            "provider-type": "GitLab",
            "author-email": "email_address",
            "commit-message": "Update file README.md",
            "author-date": "2023-08-16T21:08:08Z",
            "tag-name": "gitlab-v4.2.1",
            "commit-id": "commit_ID",
            "connection-arn": "arn:aws:codestar-connections:eu-
central-1:123456789012:connection/0f5b706a-1a1d-46c5-86b6-f177321bcfb2",
            "author-id": "Mary Major"
        },
        "state": "SUCCEEDED",
        "version": 32.0,
        "pipeline-execution-attempt": 1.0
    }
}
```

管線失敗事件

當管線執行失敗時，它會發出一個事件，該事件會傳送包含下列內容的通知。此範例適用於在「us-west-2區域」"myPipeline" 中命名的配管。

Emitted event

```
{
    "version": "0",
    "id": "01234567-EXAMPLE",
    "detail-type": "CodePipeline Pipeline Execution State Change",
    "source": "aws.codepipeline",
    "account": "123456789012",
    "time": "2020-01-31T18:55:43Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:49:39.208Z",
        "state": "FAILED",
        "version": 4,
        "pipeline-execution-attempt": 1
    }
```

```
}
```

Processed event

```
{
    "account": "123456789012",
    "detailType": "CodePipeline Pipeline Execution State Change",
    "region": "us-west-2",
    "source": "aws.codepipeline",
    "time": "2021-06-24T00:46:16Z",
    "notificationRuleArn": "arn:aws:codestar-notifications:us-
west-2:123456789012:notificationrule/a69c62c21EXAMPLE",
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:49:39.208Z",
        "state": "FAILED",
        "version": 1,
        "pipeline-execution-attempt": 1
    },
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "additionalAttributes": {
        "failedActionCount": 1,
        "failedActions": [
            {
                "action": "Deploy",
                "additionalInformation": "Deployment <ID> failed"
            }
        ],
        "failedStage": "Deploy"
    }
```

管道失敗（使用 Git 標籤的示例）

除非它在來源階段失敗，否則對於使用觸發器配置的管道，它會發出一個事件，該事件會傳送包含下列內容的通知。此範例適用於`myPipeline`在針對 Git 標籤設定的`eu-central-1execution-trigger`區域中命名的管線。

> **Note**
>
> 該execution-trigger字段將具有tag-name或branch-name，具體取決於觸發管道的事件類型。

Emitted event

```
{
    "version": "0",
    "id": "01234567-EXAMPLE",
    "detail-type": "CodePipeline Pipeline Execution State Change",
    "source": "aws.codepipeline",
    "account": "123456789012",
    "time": "2020-01-31T18:55:43Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:49:39.208Z",
        "execution-trigger": {
            "author-display-name": "Mary Major",
            "full-repository-name": "mmajor/sample-project",
            "provider-type": "GitLab",
            "author-email": "email_address",
            "commit-message": "Update file README.md",
            "author-date": "2023-08-16T21:08:08Z",
            "tag-name": "gitlab-v4.2.1",
            "commit-id": "commit_ID",
            "connection-arn": "arn:aws:codestar-connections:eu-
central-1:123456789012:connection/0f5b706a-1a1d-46c5-86b6-f177321bcfb2",
            "author-id": "Mary Major"
        },
        "state": "FAILED",
        "version": 4,
        "pipeline-execution-attempt": 1
    }
}
```

Processed event


```
{
    "account": "123456789012",
    "detailType": "CodePipeline Pipeline Execution State Change",
    "region": "us-west-2",
    "source": "aws.codepipeline",
    "time": "2021-06-24T00:46:16Z",
    "notificationRuleArn": "arn:aws:codestar-notifications:us-
west-2:123456789012:notificationrule/a69c62c21EXAMPLE",
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:49:39.208Z",
        "execution-trigger": {
            "author-display-name": "Mary Major",
            "full-repository-name": "mmajor/sample-project",
            "provider-type": "GitLab",
            "author-email": "email_address",
            "commit-message": "Update file README.md",
            "author-date": "2023-08-16T21:08:08Z",
            "tag-name": "gitlab-v4.2.1",
            "commit-id": "commit_ID",
            "connection-arn": "arn:aws:codestar-connections:eu-
central-1:123456789012:connection/0f5b706a-1a1d-46c5-86b6-f177321bcfb2",
            "author-id": "Mary Major"
        },
        "state": "FAILED",
        "version": 1,
        "pipeline-execution-attempt": 1
    },
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "additionalAttributes": {
        "failedActionCount": 1,
        "failedActions": [
            {
                "action": "Deploy",
                "additionalInformation": "Deployment <ID> failed"
            }
        ],
        "failedStage": "Deploy"
```

```
    }
```

階段級事件

階段執行發生狀態變更時，就會發出階段層級事件。

主題

- 舞台開始活動
- 階段停止事件
- 階段已停止事件
- 階段重試事件後恢復階段

舞台開始活動

當階段執行開始時，它會發出一個事件，該事件會傳送包含下列內容的通知。此範例適用於「區域」(us-east-1Region) "myPipeline" 中指定的管線，適用於階段Prod。

Emitted event

```
{
    "version": "0",
    "id": 01234567-EXAMPLE,
    "detail-type": "CodePipeline Stage Execution State Change",
    "source": "aws.codepipeline",
    "account": 123456789012,
    "time": "2020-01-24T22:03:07Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:codepipeline:us-east-1:123456789012:myPipeline"
    ],
    "detail": {
        "pipeline": "myPipeline",
        "version": "1",
        "execution-id": 12345678-1234-5678-abcd-12345678abcd,
        "start-time": "2023-10-26T13:49:39.208Z",
        "stage": "Prod",
        "state": "STARTED",
        "pipeline-execution-attempt": 1
    }
```

```
}
```

Processed event

```
{
    "account": "123456789012",
    "detailType": "CodePipeline Stage Execution State Change",
    "region": "us-east-1",
    "source": "aws.codepipeline",
    "time": "2021-06-24T00:45:40Z",
    "notificationRuleArn": "arn:aws:codestar-notifications:us-
west-2:123456789012:notificationrule/a69c62c21EXAMPLE",
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:49:39.208Z",
        "stage": "Source",
        "state": "STARTED",
        "version": 1,
        "pipeline-execution-attempt": 0
    },
    "resources": [
        "arn:aws:codepipeline:us-east-1:123456789012:myPipeline"
    ],
    "additionalAttributes": {
        "sourceActions": [
            {
                "sourceActionName": "Source",
                "sourceActionProvider": "CodeCommit",
                "sourceActionVariables": {
                    "BranchName": "main",
                    "CommitId": "<ID>",
                    "RepositoryName": "my-repo"
                }
            }
        ]
    }
}
```

階段停止事件

當階段執行停止時，它會發出一個事件，該事件會傳送包含下列內容的通知。此範例適用於「區域」(us-west-2Region) myPipeline 中指定的管線，適用於階段Deploy。

```
{
    "version": "0",
    "id": "01234567-EXAMPLE",
    "detail-type": "CodePipeline Stage Execution State Change",
    "source": "aws.codepipeline",
    "account": "123456789012",
    "time": "2020-01-24T22:02:20Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:49:39.208Z",
        "stage": "Deploy",
        "state": "STOPPING",
        "version": 3,
        "pipeline-execution-attempt": 1
    }
}
```

階段已停止事件

當階段執行停止時，它會發出一個事件，該事件會傳送包含下列內容的通知。此範例適用於「區域」(us-west-2Region) myPipeline 中指定的管線，適用於階段Deploy。

```
{
    "version": "0",
    "id": "01234567-EXAMPLE",
    "detail-type": "CodePipeline Stage Execution State Change",
    "source": "aws.codepipeline",
    "account": "123456789012",
    "time": "2020-01-31T18:21:39Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
```

```
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:49:39.208Z",
        "stage": "Deploy",
        "state": "STOPPED",
        "version": 3,
        "pipeline-execution-attempt": 1
    }
}
```

階段重試事件後恢復階段

當階段執行恢復並具有已重試的階段時，它會發出一個事件，傳送包含下列內容的通知。

重試階段後，會顯示`stage-last-retry-attempt-time`欄位，如範例所示。如果執行重試，則該欄位會在所有階段事件上顯示。

> **Note**
>
> 重試階段之後，該`stage-last-retry-attempt-time`欄位將出現在所有後續階段事件中。

```
{
    "version": "0",
    "id": "38656bcd-a798-5f92-c738-02a71be484e1",
    "detail-type": "CodePipeline Stage Execution State Change",
    "source": "aws.codepipeline",
    "account": "123456789012",
    "time": "2023-10-26T14:14:56Z",
    "region": "eu-central-1",
    "resources": [
        "arn:aws:codepipeline:eu-central-1:123456789012:BuildFromTag"
    ],
    "detail": {
        "pipeline": "BuildFromTag",
        "execution-id": "05dafb6a-5a56-4951-a858-968795364846",
        "stage-last-retry-attempt-time": "2023-10-26T14:14:56.305Z",
        "stage": "Build",
        "state": "RESUMED",
        "version": 32.0,
        "pipeline-execution-attempt": 2.0
```

```
    }
}
```

動作層級事件

當動作執行的狀態變更時，就會發出動作層級的事件。

主題

- 動作已開始事件
- 動作成功事件
- 動作失敗事件
- 動作已放棄事件

動作已開始事件

當動作執行開始時，它會發出一個事件，該事件會傳送包含下列內容的通知。此範例適用於「us-east-1區域」myPipeline 中命名的管道，用於部署動作myAction。

Emitted event

```
{
    "version": "0",
    "id": 01234567-EXAMPLE,
    "detail-type": "CodePipeline Action Execution State Change",
    "source": "aws.codepipeline",
    "account": 123456789012,
    "time": "2020-01-24T22:03:07Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:codepipeline:us-east-1:123456789012:myPipeline"
    ],
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": 12345678-1234-5678-abcd-12345678abcd,
        "start-time": "2023-10-26T13:51:09.981Z",
        "stage": "Prod",
        "action-execution-id": "47f821c5-a902-44b2-ae61-b878d31ecd21",
        "action": "myAction",
        "state": "STARTED",
        "type": {
```

```
            "owner": "AWS",
            "category": "Deploy",
            "provider": "CodeDeploy",
            "version": 1
        },
        "pipeline-execution-attempt": 1
        "input-artifacts": [
            {
                "name": "SourceArtifact",
                "s3location": {
                    "bucket": "codepipeline-us-east-1-BUCKETEXAMPLE",
                    "key": "myPipeline/SourceArti/KEYEXAMPLE"
                }
            }
        ]
    }
}
```

Processed event

```
{
    "account": "123456789012",
    "detailType": "CodePipeline Action Execution State Change",
    "region": "us-west-2",
    "source": "aws.codepipeline",
    "time": "2021-06-24T00:45:44Z",
    "notificationRuleArn": "arn:aws:codestar-notifications:us-
west-2:123456789012:notificationrule/a69c62c21EXAMPLE",
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:51:09.981Z",
        "stage": "Deploy",
        "action-execution-id": "47f821c5-a902-44b2-ae61-b878d31ecd21",
        "action": "Deploy",
        "input-artifacts": [
            {
                "name": "SourceArtifact",
                "s3location": {
                    "bucket": "codepipeline-us-east-1-EXAMPLE",
                    "key": "myPipeline/SourceArti/EXAMPLE"
                }
            }
```

```
        ],
        "state": "STARTED",
        "region": "us-east-1",
        "type": {
            "owner": "AWS",
            "provider": "CodeDeploy",
            "category": "Deploy",
            "version": "1"
        },
        "version": 1,
        "pipeline-execution-attempt": 1
    },
    "resources": [
        "arn:aws:codepipeline:us-east-1:123456789012:myPipeline"
    ],
    "additionalAttributes": {}
}
```

動作成功事件

當動作執行成功時，它會發出一個事件，該事件會傳送包含下列內容的通知。此範例適用於「us-west-2區域」"myPipeline" 中名為的管線，適用於來源動作"Source"。此事件類型有兩個不同的region欄位。事件region欄位會指定管線事件的「區域」。區regiondetail段下方的欄位會指定動作的 [區域]。

Emitted event

```
{
    "version": "0",
    "id": "01234567-EXAMPLE",
    "detail-type": "CodePipeline Action Execution State Change",
    "source": "aws.codepipeline",
    "account": "123456789012",
    "time": "2020-01-24T22:03:11Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:51:09.981Z",
```

```
        "stage": "Source",
        "execution-result": {
            "external-execution-url": "https://us-west-2.console.aws.amazon.com/
codecommit/home#/repository/my-repo/commit/8cf40f2EXAMPLE",
            "external-execution-summary": "Added LICENSE.txt",
            "external-execution-id": "8cf40fEXAMPLE"
        },
        "output-artifacts": [
            {
                "name": "SourceArtifact",
                "s3location": {
                    "bucket": "codepipeline-us-west-2-BUCKETEXAMPLE",
                    "key": "myPipeline/SourceArti/KEYEXAMPLE"
                }
            }
        ],
        "action-execution-id": "47f821c5-a902-44b2-ae61-b878d31ecd21",
        "action": "Source",
        "state": "SUCCEEDED",
        "region": "us-west-2",
        "type": {
            "owner": "AWS",
            "provider": "CodeCommit",
            "category": "Source",
            "version": "1"
        },
        "version": 3,
        "pipeline-execution-attempt": 1
    }
}
```

Processed event

```
{
    "account": "123456789012",
    "detailType": "CodePipeline Action Execution State Change",
    "region": "us-west-2",
    "source": "aws.codepipeline",
    "time": "2021-06-24T00:45:44Z",
    "notificationRuleArn": "arn:aws:codestar-notifications:us-
west-2:ACCOUNT:notificationrule/a69c62c21EXAMPLE",
    "detail": {
        "pipeline": "myPipeline",
```

```
        "execution-id": "arn:aws:codepipeline:us-west-2:123456789012:myPipeline",
        "start-time": "2023-10-26T13:51:09.981Z",
        "stage": "Source",
        "execution-result": {
            "external-execution-url": "https://us-west-2.console.aws.amazon.com/
codecommit/home#/repository/my-repo/commit/8cf40f2EXAMPLE",
            "external-execution-summary": "Edited index.html",
            "external-execution-id": "36ab3ab7EXAMPLE"
        },
        "output-artifacts": [
            {
                "name": "SourceArtifact",
                "s3location": {
                    "bucket": "codepipeline-us-west-2-EXAMPLE",
                    "key": "myPipeline/SourceArti/EXAMPLE"
                }
            }
        ],
        "action-execution-id": "47f821c5-a902-44b2-ae61-b878d31ecd21",
        "action": "Source",
        "state": "SUCCEEDED",
        "region": "us-west-2",
        "type": {
            "owner": "AWS",
            "provider": "CodeCommit",
            "category": "Source",
            "version": "1"
        },
        "version": 1,
        "pipeline-execution-attempt": 1
    },
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "additionalAttributes": {}
}
```

動作失敗事件

當動作執行失敗時，它會發出一個事件，該事件會傳送包含下列內容的通知。此範例適用於在「us-west-2區域」"myPipeline" 中命名的管線，用於動作"Deploy"。

Emitted event

```
{
    "version": "0",
    "id": "01234567-EXAMPLE",
    "detail-type": "CodePipeline Action Execution State Change",
    "source": "aws.codepipeline",
    "account": "123456789012",
    "time": "2020-01-31T18:55:43Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:51:09.981Z",
        "stage": "Deploy",
        "execution-result": {
            "external-execution-url": "https://us-west-2.console.aws.amazon.com/
codedeploy/home?#/deployments/<ID>",
            "external-execution-summary": "Deployment <ID> failed",
            "external-execution-id": "<ID>",
            "error-code": "JobFailed"
        },
        "action-execution-id": "47f821c5-a902-44b2-ae61-b878d31ecd21",
        "action": "Deploy",
        "state": "FAILED",
        "region": "us-west-2",
        "type": {
            "owner": "AWS",
            "provider": "CodeDeploy",
            "category": "Deploy",
            "version": "1"
        },
        "version": 4,
        "pipeline-execution-attempt": 1
    }
}
```

Processed event

```
{
```

```
    "account": "123456789012",
    "detailType": "CodePipeline Action Execution State Change",
    "region": "us-west-2",
    "source": "aws.codepipeline",
    "time": "2021-06-24T00:46:16Z",
    "notificationRuleArn": "arn:aws:codestar-notifications:us-
west-2:123456789012:notificationrule/a69c62c21EXAMPLE",
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "stage": "Deploy",
        "execution-result": {
            "external-execution-url": "https://console.aws.amazon.com/codedeploy/
home?region=us-west-2#/deployments/<ID>",
            "external-execution-summary": "Deployment <ID> failed",
            "external-execution-id": "<ID>",
            "error-code": "JobFailed"
        },
        "action-execution-id": "47f821c5-a902-44b2-ae61-b878d31ecd21",
        "action": "Deploy",
        "state": "FAILED",
        "region": "us-west-2",
        "type": {
            "owner": "AWS",
            "provider": "CodeDeploy",
            "category": "Deploy",
            "version": "1"
        },
        "version": 13,
        "pipeline-execution-attempt": 1
    },
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "additionalAttributes": {
        "additionalInformation": "Deployment <ID> failed"
    }
}
```

動作已放棄事件

放棄動作執行時，它會發出一個事件，該事件會傳送包含下列內容的通知。此範例適用於在「us-west-2區域」"myPipeline" 中命名的管線，用於動作"Deploy"。

```
{
    "version": "0",
    "id": "01234567-EXAMPLE",
    "detail-type": "CodePipeline Action Execution State Change",
    "source": "aws.codepipeline",
    "account": "123456789012",
    "time": "2020-01-31T18:21:39Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "stage": "Deploy",
        "action-execution-id": "47f821c5-a902-44b2-ae61-b878d31ecd21",
        "action": "Deploy",
        "state": "ABANDONED",
        "region": "us-west-2",
        "type": {
            "owner": "AWS",
            "provider": "CodeDeploy",
            "category": "Deploy",
            "version": "1"
        },
        "version": 3,
        "pipeline-execution-attempt": 1
    }
}
```

建立傳送管線事件通知的規則

規則會監視某些事件，然後將它們路由到您選擇的 AWS 目標。您可以建立規則，以便在發生另一個 AWS 動 AWS 作時自動執行動作，或是根據設定的排程定期執行動 AWS 作的規則。

主題

- 管線狀態變更時傳送通知 (主控台)

- 管線狀態變更時傳送通知 (CLI)

管線狀態變更時傳送通知 (主控台)

這些步驟顯示如何使用 EventBridge 主控台建立規則，以便在中傳送變更通知 CodePipeline。

若要使用 Amazon S3 來源建立以管道為目標的 EventBridge 規則

1. 在以下位置打開 Amazon EventBridge 控制台 https://console.aws.amazon.com/events/。
2. 在導覽窗格中，選擇規則。保持選取預設匯流排，或選擇活動匯流排。選擇建立規則。
3. 在名稱中，輸入規則的名稱。
4. 在「規則類型」下，選擇「具有事件模式的規則」。選擇下一步。
5. 在事件模式下，選擇AWS 服務。
6. 從 Event Type (事件類型) 下拉式清單中，選擇通知的狀態變更層級。
   - 對於套用至管線層級事件的規則，請選擇「CodePipeline管線執行狀態變更」。
   - 對於套用至階段層級事件的規則，請選擇「CodePipeline階段執行狀態變更」。
   - 針對套用至動作層級事件的規則，請選擇「CodePipeline動作執行狀態變更」。
7. 指定規則套用至的狀態變更：
   - 針對套用至所有狀態變更的規則，選擇 Any state (任何狀態)。
   - 針對僅套用到部分狀態變更的規則，選擇 Specific state(s) (特定狀態)，然後從清單中選擇一或多個狀態值。
8. 對於比選取器允許更詳細的事件模式，您也可以使用 [事件模式] 視窗中的 [編輯模式] 選項，以 JSON 格式指定事件模式。

   > **Note**
   >
   > 如果未指定，則會針對所有管道/階段/動作和狀態建立事件模式。

   如需更詳細的事件模式，您可以將下列範例事件模式複製並貼到「事件模式」視窗中。

   - Example

     使用此範例事件模式，擷取所有管道的失敗部署和建置動作。

```
{
"source": [
    "aws.codepipeline"
  ],
  "detail-type": [
    "CodePipeline Action Execution State Change"
  ],
  "detail": {
    "state": [
      "FAILED"
    ],
    "type": {
      "category": ["Deploy", "Build"]
    }
  }
}
```

- Example

  使用此範例事件模式，擷取所有管道的所有已拒絕或失敗核准動作。

```
{
 "source": [
    "aws.codepipeline"
  ],
  "detail-type": [
    "CodePipeline Action Execution State Change"
  ],
  "detail": {
    "state": [
      "FAILED"
    ],
    "type": {
      "category": ["Approval"]
    }
  }
}
```

- Example

  使用此範例事件模式，擷取所指定管道的所有事件。

```
{
"source": [
    "aws.codepipeline"
  ],
  "detail-type": [
    "CodePipeline Pipeline Execution State Change",
    "CodePipeline Action Execution State Change",
    "CodePipeline Stage Execution State Change"
  ],
  "detail": {
    "pipeline": ["myPipeline", "my2ndPipeline"]
  }
}
```

9. 選擇下一步。
10. 在目標類型中，選擇AWS 服務。
11. 在 [選取目標] 中，選擇CodePipeline。在管線 ARN 中，輸入管線 ARN，讓此規則啟動的管線。

> **Note**
>
> 若要取得管道 ARN，請執行 get-pipeline 命令。管道 ARN 會出現在輸出中。其建構格式如下：
>
> *AR: aws: #####:##:##:####*
>
> 範例管道 ARN：
>
> ARN：AW：代碼流水線：美國東部-2：80398 示例：MyFirstPipeline

12. 若要建立或指定 IAM 服務角色，以授與呼叫 EventBridge 規則關聯之目標的 EventBridge 權限 (在本例中，目標為 CodePipeline)：
    - 選擇 [為此特定資源建立新角色] 以建立服務角色，以提供啟動管線執行的 EventBridge 權限的服務角色。
    - 選擇 [使用現有角色] 以輸入授與啟動管線執行之 EventBridge 權限的服務角色。
13. 選擇下一步。
14. 在 [標籤] 頁面上，選擇 [下一步]。
15. 在 [檢閱並建立] 頁面上，檢閱規則組態。如果您對此規則感到滿意，請選擇 Create rule (建立規則)。

管線狀態變更時傳送通知 (CLI)

這些步驟顯示如何使用 CLI 建立 CloudWatch 事件規則，以便在中傳送變更通知 CodePipeline。

若要使用 AWS CLI 建立規則，請呼叫指put-rule令，並指定：

- 可唯一識別您所建立規則的名稱。在與 AWS 帳戶 CodePipeline 相關聯的所有管道中，此名稱必須是唯一的。
- 規則所使用來源和詳細資訊欄位的事件模式。如需詳細資訊，請參閱 Amazon EventBridge 和事件模式。

建立 EventBridge 規則 CodePipeline 做為事件來源的步驟

1. 呼叫 put-rule 命令，以建立規則並指定事件模式 (如需有效狀態，請參閱上述各表)。

   下列範例命令用--event-pattern來建立一個名為的規則，“MyPipelineStateChanges”該規則會在名為「MyPipeline」的管線執行失敗時發出 CloudWatch 事件。

   ```
   aws events put-rule --name "MyPipelineStateChanges" --event-pattern "{\"source\":
   [\"aws.codepipeline\"],\"detail-type\":[\"CodePipeline Pipeline Execution State
    Change\"],\"detail\":{\"pipeline\":[\"myPipeline\"],\"state\":[\"FAILED\"]}}"
   ```

2. 呼叫命put-targets令並包含下列參數：

   - `--rule` 參數與您使用 put-rule 所建立的 `rule_name` 搭配使用。
   - 此`--targets`參數會與目標清單中`Id`的目標清單和 Amazon SNS 主題ARN的清單搭配使用。

   以下命令範例指定名為 `MyPipelineStateChanges` 的規則，該目標 `Id` 是由數字 1 組成，指出在規則的目標清單中，這是目標 1。範例命令也會指定 Amazon SNS 主題的範例ARN。

   ```
   aws events put-targets --rule MyPipelineStateChanges --targets
    Id=1,Arn=arn:aws:sns:us-west-2:11111EXAMPLE:MyNotificationTopic
   ```

3. 新增使用指定目標服務呼叫通知的權限。 EventBridge 如需詳細資訊，請參閱在 Amazon EventBridge 使用以資源為基礎的政策。

事件預留位置儲存貯體參考

本節僅供參考。如需建立包含事件偵測資源之管道的相關資訊，請參閱 來源動作和變更偵測方法。

Amazon S3 提供的來源動作，並在來源儲存貯體或儲存庫中進行變更時， CodeCommit 使用以事件為基礎的變更偵測資源觸發管道。這些資源是設定為回應管線來源中的事件 (例如對 CodeCommit 存放庫的程式碼變更) 的事件規則。 CloudWatch 當您將 CloudWatch 事件用於 Amazon S3 來源時，您必須開啟 CloudTrail 以記錄事件。 CloudTrail 需要一個 S3 存儲桶，它可以在其中發送摘要。您可以從自訂值區存取 E CloudWatch vents 資源的記錄檔，但無法從預留位置值區存取資料。

- 如果您使用 CLI 或設AWS CloudFormation定 E CloudWatch vents 資源，則可以在設定管線時指定的儲存貯體中找到 CloudTrail 檔案。
- 如果您使用主控台透過 S3 來源設定管道，則主控台會在為您建立 E CloudWatch vents 資源時使用 CloudTrail 預留位置儲存貯體。 CloudTrail 摘要會儲存在建立管線的預留位置值區中。AWS 區域

如果您想使用預留位置儲存貯體以外的儲存貯體，可以變更組態。

> **Note**
>
> 寫入 CloudTrail 預留位置值區的資料會在一天後自動過期，且不會保留。

如需有關尋找和管理 CloudTrail 記錄檔的詳細資訊，請參閱取得和檢視 CloudTrail 記錄檔。

主題

事件預留位置儲存貯體名稱 (依區域)

此表格列出 S3 預留位置儲存貯體的名稱，其中包含可追蹤具有 Amazon S3 來源動作之管道變更偵測事件的日誌檔。

區域名稱	預留位置儲存貯體名稱	區域識別碼
美國東部 (俄亥俄)	codepipeline-cloudtrail-placeholder-bucket-美國東部 -2	us-east-2
美國東部 (維吉尼亞北部)	codepipeline-cloudtrail-placeholder-bucket-美國東部 -1	us-east-1

區域名稱	預留位置儲存貯體名稱	區域識別碼
美國西部 (加利佛尼亞北部)	codepipeline-cloudtrail-placeholder-bucket-美國西部 -1	us-west-1
美國西部 (奧勒岡)	codepipeline-cloudtrail-placeholder-bucket-美國西部 -2	us-west-2
加拿大 (中部)	codepipeline-cloudtrail-placeholder-bucket-CA-中央 -1	ca-central-1
歐洲 (法蘭克福)	codepipeline-cloudtrail-placeholder-bucket-歐盟中央 -1	eu-central-1
歐洲 (愛爾蘭)	codepipeline-cloudtrail-placeholder-bucket-歐洲西部 -1	eu-west-1
歐洲 (倫敦)	codepipeline-cloudtrail-placeholder-bucket-歐洲西部 -2	eu-west-2
Europe (Paris)	codepipeline-cloudtrail-placeholder-bucket-歐洲西部 -3	eu-west-3
歐洲 (斯德哥爾摩)	codepipeline-cloudtrail-placeholder-bucket-歐洲北部 -1	eu-north-1
亞太區域 (香港)	codepipeline-cloudtrail-placeholder-bucket-AP-東 -1	ap-east-1
亞太區域 (海德拉巴)	codepipeline-cloudtrail-placeholder-bucket-南方	ap-south-2
亞太區域 (雅加達)	codepipeline-cloudtrail-placeholder-bucket-AP-東南部 -3	ap-southeast-3
亞太區域 (墨爾本)	codepipeline-cloudtrail-placeholder-bucket-AP-東南部 -4	ap-southeast-4
亞太區域 (孟買)	codepipeline-cloudtrail-placeholder-bucket-南方 -1	ap-south-1

區域名稱	預留位置儲存貯體名稱	區域識別碼
亞太區域 (大阪)	codepipeline-cloudtrail-placeholder-bucket-AP-東北部 -3-產品	ap-northeast-3
亞太區域 (東京)	codepipeline-cloudtrail-placeholder-bucket-AP-東北 -1	ap-northeast-1
亞太區域 (首爾)	codepipeline-cloudtrail-placeholder-bucket-AP-東北 -2	ap-northeast-2
亞太區域 (新加坡)	codepipeline-cloudtrail-placeholder-bucket-AP-東南部 -1	ap-southeast-1
亞太區域 (雪梨)	codepipeline-cloudtrail-placeholder-bucket-AP-東南部 -2	ap-southeast-2
亞太區域 (東京)	codepipeline-cloudtrail-placeholder-bucket-AP-東北 -1	ap-northeast-1
加拿大 (中部)	codepipeline-cloudtrail-placeholder-bucket-CA-中央 -1	ca-central-1
歐洲 (法蘭克福)	codepipeline-cloudtrail-placeholder-bucket-歐盟中央 -1	eu-central-1
歐洲 (愛爾蘭)	codepipeline-cloudtrail-placeholder-bucket-歐洲西部 -1	eu-west-1
歐洲 (倫敦)	codepipeline-cloudtrail-placeholder-bucket-歐洲西部 -2	eu-west-2
歐洲 (米蘭)	codepipeline-cloudtrail-placeholder-bucket-歐洲南方 -1	eu-south-1
Europe (Paris)	codepipeline-cloudtrail-placeholder-bucket-歐洲西部 -3	eu-west-3

區域名稱	預留位置儲存貯體名稱	區域識別碼
歐洲 (西班牙)	codepipeline-cloudtrail-placeholder-bucket-歐洲南方 -2	eu-south-2
歐洲 (斯德哥爾摩)	codepipeline-cloudtrail-placeholder-bucket-歐洲北部 -1	eu-north-1
歐洲 (蘇黎世) *	codepipeline-cloudtrail-placeholder-bucket-歐盟中央 -2	eu-central-2
以色列 (特拉維夫)	codepipeline-cloudtrail-placeholder-bucket-中央 -1	il-central-1
中東 (巴林) *	codepipeline-cloudtrail-placeholder-bucket-我-南 -1	me-south-1
中東 (阿拉伯聯合大公國)	codepipeline-cloudtrail-placeholder-bucket-我中央 -1	me-central-1
南美洲 (聖保羅)	codepipeline-cloudtrail-placeholder-bucket-SA-東 -1	sa-east-1

使用 AWS CloudTrail 記錄 CodePipeline API 呼叫

AWS CodePipeline整合AWS CloudTrail，該服務提供由使用者、角色或AWS 服務在中所採取的動作的服CodePipeline務。CloudTrail 會擷取 CodePipeline 的所有 API 呼叫當作事件。擷取的呼叫包括從 CodePipeline 主控台進行的呼叫，以及針對 CodePipeline API 操作的程式碼呼叫。如果您建立追蹤，就可以將事件持續交付至 Amazon S3 儲存貯體，請將CloudTrail事件持續交付至 Amazon S3 儲存貯體 CodePipeline。如果您不設定追蹤記錄，仍然可以透過 CloudTrail 主控台中的 Event history (事件歷史記錄) 檢視最新的事件。使用由 CloudTrail 收集的資訊，您就可以判斷送至 CodePipeline 的請求、提出請求的 IP 地址、誰提出請求、提出請求的時間，以及其他詳細資訊。

若要進一步了解 CloudTrail，請參閱 AWS CloudTrail 使用者指南。

CloudTrail 中的 CodePipeline 資訊

當您建立帳戶時，系統會在您的 AWS 帳戶 中啟用 CloudTrail。當 CodePipeline 中發生活動時，該活動會記錄在 CloudTrail 事件中，其他 AWS 服務 事件則記錄於 Event history (事件歷史記錄)。您可以

檢視、搜尋和下載 AWS 帳戶的最新事件。如需詳細資訊，請參閱使用 CloudTrail 事件歷程記錄檢視事件。

若要持續記錄您的中的事件 AWS 帳戶CodePipeline，請建立追蹤蹤。線索能讓您CloudTrail將日誌檔案交付至 Amazon S3 儲存貯體。根據預設，當您在主控台建立線索時，線索會套用到所有 AWS 區域。該追蹤會記錄來自 AWS 分割區中所有區域的事件，並將日誌檔案交付到您指定的 Amazon S3 儲存貯體。此外，您可以設定其他 AWS 服務，以進一步分析和處理 CloudTrail 日誌中所收集的事件資料。如需詳細資訊，請參閱下列內容：

- 建立追蹤的概觀
- CloudTrail 支援的服務和整合
- 設定 CloudTrail 的 Amazon SNS 通知
- 接收多個區域的 CloudTrail 日誌檔案及接收多個帳戶的 CloudTrail 日誌檔案

CloudTrail 會記錄所有 CodePipeline 動作，列在 CodePipeline API 參考中。例如，對 `CreatePipeline`、`GetPipelineExecution` 和 `UpdatePipeline` 動作發出的呼叫會在 CloudTrail 日誌檔案中產生項目。

每一筆事件或日誌項目都會包含產生請求者的資訊。身分資訊可協助您判斷下列事項：

- 該請求是否使用根或 AWS Identity and Access Management (IAM) 登入資料提出。
- 提出該請求時，是否使用了特定角色或聯合身分使用者的暫時安全憑證。
- 該請求是否由另一項 AWS 服務 服務提出。

如需詳細資訊，請參閱 CloudTrail 使用者身分元素。

了解 CodePipeline 日誌檔案項目

追蹤是一種組態，能讓事件以日誌檔案的形式交付到您指定的 Amazon S3 儲存貯體。CloudTrail 日誌檔案包含一個或多個日誌項目。一個事件為任何來源提出的單一請求，並包含請求動作、請求的日期和時間、請求參數等資訊。CloudTrail 日誌檔案並非依公有 API 呼叫追蹤記錄的堆疊排序，因此不會以任何特定順序出現。

下列範例顯示更新管線事件的CloudTrail記錄項目，其中名為的管線MyFirstPipeline已由名為的使用者編輯 JaneDoe-CodePipeline 帳戶識別碼為 80398EXAMP例。使用者已將管道的來源階段名稱從 `Source` 變更為 `MySourceStage`。因為 CloudTrail 日誌中的 `requestParameters` 和 `responseElements` 元素包含編輯過管道的整個結構，所以下列範例中已將這些元素縮

寫。Emphasis 已新增至發生變更之管道的 requestParameters 部分、管道的舊版本號碼，以及 responseElements 部分 (顯示遞加 1 的版本號碼)。編輯過部分會標上省略符號 (...)，說明實際日誌項目中出現更多資料的位置。

```
{
 "eventVersion":"1.03",
  "userIdentity": {
   "type":"IAMUser",
   "principalId":"AKIAI44QH8DHBEXAMPLE",
   "arn":"arn:aws:iam::80398EXAMPLE:user/JaneDoe-CodePipeline",
   "accountId":"80398EXAMPLE",
   "accessKeyId":"AKIAIOSFODNN7EXAMPLE",
   "userName":"JaneDoe-CodePipeline",
   "sessionContext": {
   "attributes":{
    "mfaAuthenticated":"false",
    "creationDate":"2015-06-17T14:44:03Z"
    }
   },
 "invokedBy":"signin.amazonaws.com"},
 "eventTime":"2015-06-17T19:12:20Z",
 "eventSource":"codepipeline.amazonaws.com",
 "eventName":"UpdatePipeline",
 "awsRegion":"us-east-2",
 "sourceIPAddress":"192.0.2.64",
 "userAgent":"signin.amazonaws.com",
 "requestParameters":{
   "pipeline":{
   "version":1,
   "roleArn":"arn:aws:iam::80398EXAMPLE:role/CodePipeline_Service_Role",
   "name":"MyFirstPipeline",
   "stages":[
     {
     "actions":[
      {
          "name":"MySourceStage",
       "actionType":{
       "owner":"AWS",
       "version":"1",
       "category":"Source",
       "provider":"S3"
     },
    "inputArtifacts":[],
```

```
    "outputArtifacts":[
     {"name":"MyApp"}
     ],
    "runOrder":1,
    "configuration":{
     "S3Bucket":"awscodepipeline-demobucket-example-date",
     "S3ObjectKey":"sampleapp_linux.zip"
      }
      }
     ],
       "name":"Source"
     },
     (...)
                },
 "responseElements":{
   "pipeline":{
     "version":2,
     (...)
           },
     "requestID":"2c4af5c9-7ce8-EXAMPLE",
     "eventID":""c53dbd42-This-Is-An-Example"",
     "eventType":"AwsApiCall",
     "recipientAccountId":"80398EXAMPLE"
     }
      ]
}
```

疑難排 CodePipeline

以下資訊可能有助於診斷 AWS CodePipeline 內的常見問題。

主題

- 當達到檔案限制時，具有使用依檔案路徑觸發程序篩選的連線的管線可能無法啟動
- 需要不同問題的協助嗎？

管道錯誤：使用 AWS Elastic Beanstalk 設定的管道傳回錯誤訊息：「部署失敗。提供的角色沒有足夠的權限：服務：AmazonElasticLoadBalancing」

問題：的服務角色 CodePipeline 沒有足夠的權限AWS Elastic Beanstalk，包括但不限於 Elastic Load Balancing 中的某些作業。的服務角色 CodePipeline 已於 2015 年 8 月 6 日更新，以解決此問題。在此日期前建立其服務角色的客戶，必須修改服務角色的政策陳述式以新增必要許可。

可能的修正：最簡單的解決方案是編輯服務角色的政策陳述式，詳述於將許可新增至 CodePipeline 服務角色中。

套用已編輯的原則後，請遵循中手動啟動管道的步驟手動重新執行任何使用 Elastic Beanstalk 的管線。

根據您的安全性需求，也可以使用其他方式修改許可。

部署錯誤：如果缺少 "DescribeEvents" 權限，配置了部AWS Elastic Beanstalk署動作的管道會掛起而不是失敗

問題：的服務角色 CodePipeline 必須包含任何使用的管線的`"elasticbeanstalk:DescribeEvents"`動作AWS Elastic Beanstalk。若沒有此許可，AWS Elastic Beanstalk 部署動作會在沒有失敗或指出任何錯誤的情況下停止回應。如果您的服務角色遺漏此動作，則 CodePipeline 沒有代表您執行管線部署階段AWS Elastic Beanstalk的權限。

可能的修正：檢閱您的 CodePipeline 服務角色。如果 `"elasticbeanstalk:DescribeEvents"` 動作遺失，請使用將許可新增至 CodePipeline 服務角色 中的步驟並使用 Edit Policy (編輯政策) 功能將它加入 IAM 主控台。

套用已編輯的原則後，請遵循中手動啟動管道的步驟手動重新執行任何使用 Elastic Beanstalk 的管線。

管道錯誤：源動作返回權限不足消息：「無法訪問 CodeCommit 存儲庫**repository-name**。請確認管道 IAM 角色擁有存取該程式庫所需的足夠許可。」

問題：的服務角色 CodePipeline 沒有足夠的權限，可能是在 2016 年 4 月 18 日新增使用 CodeCommit儲存庫的支援之前建立的。 CodeCommit 在此日期前建立其服務角色的客戶，必須修改服務角色的政策陳述式以新增必要許可。

可能的修正：將必要的權限新增 CodeCommit 至 CodePipeline 服務角色的原則。如需詳細資訊，請參閱 將許可新增至 CodePipeline 服務角色。

管道錯誤：Jenkins 建置或測試動作在一段長時間的執行後，因為缺乏登入資料或許可而失敗。

問題：如果 Jenkins 伺服器安裝在 Amazon EC2 執行個體上，則可能未使用具有所需許可的執行個體角色建立執行個體角色。 CodePipeline如果您在 Jenkins 伺服器、現場部署執行個體或沒有必要 IAM 角色建立的 Amazon EC2 執行個體上使用 IAM 使用者，則 IAM 使用者可能沒有必要的許可，或者 Jenkins 伺服器無法透過伺服器上設定的設定檔存取這些登入資料。

可能的修正：確定 Amazon EC2 執行個體角色或 IAM 使用者已設定`AWSCodePipelineCustomActionAccess`受管政策或具有等效許可。如需詳細資訊，請參閱 AWS 受管理的政策 AWS CodePipeline。

如果您使用 IAM 使用者，請確定執行個體上設定的設定AWS檔使用設定為正確許可的 IAM 使用者。您可能必須提供為 Jenkins 之間進行集成而配置的 IAM 用戶憑據，並 CodePipeline 直接進入 Jenkins UI。這不是建議的最佳實務。若您必須執行此作業，請確認 Jenkins 伺服器為安全並使用 HTTPS 而非 HTTP。

管道錯誤：在一個區域中使用在另AWS一個AWS區域中建立的儲存貯體建立的管道，會傳回代碼為 "InternalError」的 "JobFailed"

問題：如果管道和儲存貯體在不同AWS區域建立，則下載存放在 Amazon S3 儲存貯體中的成品將會失敗。

可能的修正：確保存放成品的 Amazon S3 儲存貯體與您建立的管道位於相同的AWS區域。

部署錯誤：包含 WAR 檔案的 ZIP 檔案已成功部署到 AWS Elastic Beanstalk，但應用程式 URL 報告「404 找不到」錯誤。

問題：WAR 檔案已成功部署到 AWS Elastic Beanstalk 環境，但應用程式 URL 卻傳回「404 找不到」錯誤。

可能的修復方法：AWS Elastic Beanstalk 可以解除封裝 ZIP 檔案，但無法解除封裝位於 ZIP 檔案中的 WAR 檔案。請改為指定包含要部署內容的資料夾，而非您 `buildspec.yml` 檔案中的 WAR 檔案。例如：

```
version: 0.2

phases:
  post_build:
    commands:
      - mvn package
      - mv target/my-web-app ./
artifacts:
  files:
    - my-web-app/**/*
 discard-paths: yes
```

如需範例，請參閱 CodeBuild 的 AWS Elastic Beanstalk 範例。

管道成品資料夾名稱似乎被截斷了

問題：當您在中檢視管線人工因素名稱時 CodePipeline，名稱會被截斷。這可能使名稱看起來都很相似或似乎不再包含整個管道名稱。

說明： CodePipeline 截斷成品名稱，以確保在為任務工作者 CodePipeline 產生臨時登入資料時，完整的 Amazon S3 路徑不會超過政策大小限制。

即使成品名稱似乎被截斷，也會以不受名稱截斷的成品影響的方式對 CodePipeline 應至成品值區。該管道可以正常運作。這不是資料夾或成品的問題。管道名稱有 100 個字元的限制。雖然成品資料夾名稱看起來可能變短了，對管道來說仍然是唯一的。

新增連線至 Bitbucket GitHub、 GitHub 企業伺服器或 GitLab .com 的 CodeBuild GitClone 權限

當您在來源動作和動作中使用AWS CodeStar連線時，有兩種方式可將輸入加工品傳遞至組建：CodeBuild

- 預設值：來源動作會產生 zip 檔案，其中包含 CodeBuild 下載的程式碼。
- Git 複製：來源程式碼可以直接下載到建置環境。

  Git 複製模式可讓您將原始程式碼當成工作中 Git 儲存庫來互動。若要使用此模式，您必須授與 CodeBuild 環境使用連線的權限。

若要將權限新增至 CodeBuild 服務角色原則，您需要建立附加至 CodeBuild 服務角色的客戶管理原則。下列步驟建立政策，其中，`action` 欄位中指定 `UseConnection` 許可，而 `Resource` 欄位中指定連線 ARN。

使用主控台新增 UseConnection 權限

1. 若要尋找管道的連線 ARN，請開啟管道，在來源動作上按一下 (i) 圖示。您可以將連線 ARN 新增至您的 CodeBuild 服務角色原則。

   ARN 連接的示例是：

   ```
   arn:aws:codestar-connections:eu-central-1:123456789123:connection/
   sample-1908-4932-9ecc-2ddacee15095
   ```

2. 若要尋找您的 CodeBuild 服務角色，請開啟管線中使用的建置專案，然後瀏覽至 [建置詳細資料] 索引標籤。
3. 選擇 Service role (服務角色) 連結。這會開啟 IAM 主控台，讓您新增政策以授予存取您的連線。
4. 在 IAM 主控台，選擇 Attach policies (連接政策)，然後選擇 Create policy (建立政策)。

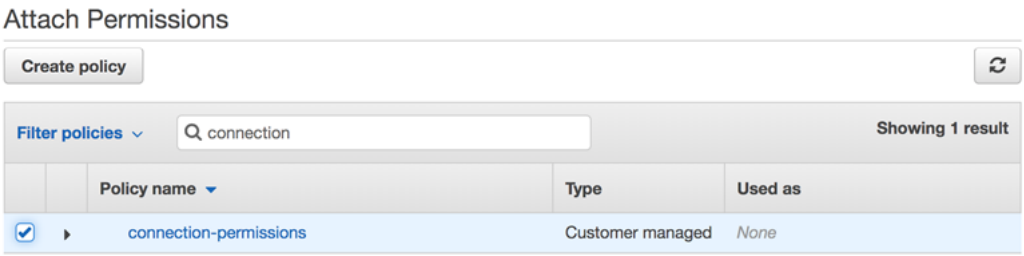
   使用下列政策範本範例。在 `Resource` 欄位中新增連線 ARN，如下列範例所示：

   ```
   {
       "Version": "2012-10-17",
       "Statement": [
           {
               "Effect": "Allow",
               "Action": "codestar-connections:UseConnection",
   ```

```
            "Resource": "insert connection ARN here"
        }
    ]
}
```

在 JSON 索引標籤上，貼上您的政策。

5. 選擇檢閱政策。輸入政策的名稱 (例如 **connection-permissions**)，然後選擇 Create policy (建立政策)。
6. 返回您連接許可的頁面，重新整理政策清單，然後選取您剛建立的政策。選擇連接政策。



新增 CodeCommit來源動作的 CodeBuild GitClone 權限

當您的管道具有 CodeCommit 源動作時，有兩種方法可以將輸入成品傳遞給構建：

- 預設 — 來源動作會產生包含 CodeBuild 下載程式碼的 zip 檔案。
- 完整複製 — 原始程式碼可直接下載至建置環境。

「完整複製」選項可讓您與原始程式碼作為有效的 Git 儲存庫進行互動。若要使用此模式，您必須新增要從存放庫中提取 CodeBuild環境的權限。

若要將權限新增至 CodeBuild 服務角色原則，您需要建立附加至 CodeBuild 服務角色的客戶管理原則。下列步驟會建立在`action`欄位中指定`codecommit:GitPull`權限的原則。

使用主控台新增 GitPull 權限

1. 若要尋找您的 CodeBuild 服務角色，請開啟管線中使用的建置專案，然後瀏覽至 [建置詳細資料] 索引標籤。
2. 選擇 Service role (服務角色) 連結。這會開啟 IAM 主控台，您可以在其中新增授與存放庫存取權的新政策。
3. 在 IAM 主控台，選擇 Attach policies (連接政策)，然後選擇 Create policy (建立政策)。

4. 在 JSON 索引標籤上，貼上下列範例原則。

```
{
    "Action": [
        "codecommit:GitPull"
    ],
    "Resource": "*",
    "Effect": "Allow"
},
```

5. 選擇檢閱政策。輸入政策的名稱 (例如 **codecommit-gitpull**)，然後選擇 Create policy (建立政策)。
6. 返回您連接許可的頁面，重新整理政策清單，然後選取您剛建立的政策。選擇連接政策。

<source artifact name>管線錯誤：具有 CodeDeployTo ECS 動作的部署會傳回錯誤訊息：「嘗試從下列位置讀取作業定義人工因素檔案時發生異常：」

問題：

任務定義檔案是透過 CodeDeploy (動作) 將動作 CodePipeline 部署至 Amazon ECS 的必要成品。CodeDeployToECSCodeDeployToECS部署動作中的成品 ZIP 大小上限為 3 MB。找不到檔案或成品大小超過 3 MB 時，會傳回下列錯誤訊息：

嘗試從 <來源成品名稱> 讀取工作定義成品檔案時發生例外狀況

可能的修正：請確定工作定義檔案已包含為人工因素。如果該文件已經存在，請確保壓縮的大小小小於 3 MB。

GitHub 版本 1 源動作：存儲庫列表顯示不同的存儲庫

問題：

在 CodePipeline 主控台中成功授權第 1 GitHub 版動作之後，您可以從 GitHub 存放庫清單中選擇。如果清單中不包含您預期看到的儲存庫，您可以疑難排解用於授權的帳戶。

可能的修正： CodePipeline 主控台中提供的儲存庫清單是根據授權帳戶所屬的 GitHub 組織而定。確認您用來授權的帳戶 GitHub 是與建立存放庫的 GitHub 組織相關聯的帳戶。

GitHub 版本 2 源操作：無法完成存儲庫的連接

問題：

由於 GitHub 存放庫的連線會使用 AWS Connector GitHub，因此您需要組織擁有者權限或存放庫的管理員權限，才能建立連線。

可能的修正：如需 GitHub 儲存庫權限層級的相關資訊，請參閱 https://docs.github.com/en/ free-pro-team @latest /github/-/setting-up-and-managing-組織。organizations-and-teams permission-levels-for-an

Amazon S3 錯誤： CodePipeline 服務角色<ARN>正在拒絕 S3 存取 S3 儲存貯體 < BucketName >

問題：

進行中的 CodeCommit 動作 CodePipeline 會檢查管線加工品值區是否存在。如果動作沒有檢查權限，Amazon S3 會發生錯`AccessDenied`誤，並在中顯示以下錯誤訊息 CodePipeline：

CodePipeline *`#####arn: aw: iam:: AccountID: ##/####/## ID##### S3 ## S3 #### "#BucketName`*

動作的 CloudTrail 記錄檔也會記錄`AccessDenied`錯誤。

可能的修正：請執行下列動作：

- 針對附加至 CodePipeline 服務角色的原則，新增`s3:ListBucket`至原則中的動作清單。如需有關檢視服務角色原則的指示，請參閱檢視管線 ARN 和服務角色 ARN (主控台)。編輯服務角色的政策聲明，如中所述將許可新增至 CodePipeline 服務角色。
- 對於管道附加至 Amazon S3 成品儲存貯體的資源型政策 (也稱為成品儲存貯體政策)，請新增聲明以允許您的 CodePipeline 服務角色使用`s3:ListBucket`權限。

  將您的政策新增至成品值區

  1. 按照中的步驟在檢視管線 ARN 和服務角色 ARN (主控台)管道設定頁面上選擇您的成品儲存貯體，然後在 Amazon S3 主控台中檢視它。
  2. 選擇 Permissions (許可)。
  3. 在 Bucket policy (儲存貯體政策) 下方，選擇 Edit (編輯)。

4. 在 [原則] 文字欄位中，輸入新的儲存貯體政策，或編輯現有政策，如下列範例所示。值區政策是 JSON 檔案，因此您必須輸入有效的 JSON。

   下列範例顯示成品值區的值區原則陳述式，其中服務角色的範例角色識別碼為 *AROA* EXAMPLEID。

```
{
    "Effect": "Allow",
    "Principal": "*",
    "Action": "s3:ListBucket",
    "Resource": "arn:aws:s3:::BucketName",
    "Condition": {
        "StringLike": {
            "aws:userid": "AROAEXAMPLEID:*"
        }
    }
}
```

   下列範例顯示新增權限後的相同儲存貯體政策陳述式。

```
{
    "Version": "2012-10-17",
    "Id": "SSEAndSSLPolicy",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "s3:ListBucket",
            "Resource": "arn:aws:s3:::codepipeline-us-east-2-1234567890",
            "Condition": {
                "StringLike": {
                    "aws:userid": "AROAEXAMPLEID:*"
                }
            }
        },
        {
            "Sid": "DenyUnEncryptedObjectUploads",
            "Effect": "Deny",
            "Principal": "*",
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::codepipeline-us-east-2-1234567890/*",
            "Condition": {
```

```
                "StringNotEquals": {
                    "s3:x-amz-server-side-encryption": "aws:kms"
                }
            }
        },
        {
            "Sid": "DenyInsecureConnections",
            "Effect": "Deny",
            "Principal": "*",
            "Action": "s3:*",
            "Resource": "arn:aws:s3:::codepipeline-us-east-2-1234567890/*",
            "Condition": {
                "Bool": {
                    "aws:SecureTransport": false
                }
            }
        }
    ]
}
```

如需詳細資訊，請參閱https://aws.amazon.com/blogs/security/writing-iam-policies-how-to-grant-access-to-an-amazon-s3-bucket/中的步驟。

5. 選擇儲存。

套用已編輯的策略之後，請按照中手動啟動管道的步驟手動重新執行管道。

具有 Amazon S3、Amazon ECR 或 CodeCommit來源的管道不再自動啟動

問題：

變更使用事件規則 (EventBridge或 CloudWatch 「事件」) 進行變更偵測的動作的組態設定後，主控台可能無法偵測到來源觸發器識別碼相似且具有相同初始字元的變更。由於新的事件規則不是由控制台建立的，因此管線不會再自動啟動。

參數名稱末尾的一個小更改示例是 CodeCommit 將您的 CodeCommit 分支名稱更改`MyTestBranch-1`為`MyTestBranch-2`。由於變更位於分支名稱的末尾，因此來源動作的事件規則可能不會更新或建立新來源設定的規則。

這適用於使用 CWE 事件進行變更偵測的來源動作，如下所示：

來源動作	參數/觸發器識別碼 (主控台)
Amazon ECR	儲存庫名稱 圖像標籤
Amazon S3	儲存貯體 S3 物件金鑰
CodeCommit	儲存庫名稱 分行名稱

可能的修正：

執行以下任意一項：

- 變更 CodeCommit /S3/ECR 組態設定，以便對參數值的起始部分進行變更。

  範例：將您的分支名稱變更`release-branch`為`2nd-release-branch`. 避免在名稱的末尾進行更改，例如`release-branch-2`。
- 變更每個配管的 CodeCommit /S3/ECR 組態設定。

  範例：將您的分支名稱變更`myRepo/myBranch`為`myDeployRepo/myDeployBranch`. 避免在名稱的末尾進行更改，例如`myRepo/myBranch2`。
- 而不是控制台，使用 CLI 或AWS CloudFormation創建和更新您的變更檢測事件規則。如需針對 S3 來源動作建立事件規則的指示，請參閱Amazon S3 來源動作 EventBridge 與 AWS CloudTrail。如需針對 Amazon ECR 動作建立事件規則的指示，請參閱 Amazon ECR 來源動作和 EventBridge 資源。如需為動作建立事件規則的指 CodeCommit 示，請參閱 CodeCommit 來源動作和 EventBridge。

在主控台中編輯動作設定之後，請接受由主控台建立的更新變更偵測資源。

連線時發生連線錯誤 GitHub：「發生問題，請確定您的瀏覽器已啟用 Cookie」或「組織擁有者必須安裝 GitHub 應用程式」

問題：

若要在中建立 GitHub 來源動作的連線 CodePipeline，您必須是 GitHub組織擁有者。對於不在組織下的儲存庫，您必須是儲存庫擁有者。由組織擁有者以外的其他人員建立連線時，會針對組織擁有者建立請求，並顯示下列其中一個錯誤：

發生問題，請確保您的瀏覽器已啟用 cookie

或

組織擁有者必須安裝 GitHub 應用程式

可能的修正：對於 GitHub組織中的儲存庫，組織擁有者必須建立與 GitHub 存放庫的連線。對於不在組織下的儲存庫，您必須是儲存庫擁有者。

區域中無法使用`CloudFormationStackSet`或`CloudFormationStackInstances`作時發生錯誤

問題：在某個區域 CodePipeline 中存在並不意味著該區域中的所有操作都可以使用。例如，當管線在`CloudFormationStackSet`動作不可用的區域 (例如非洲 (開普敦) 執行動作時，會顯示下列錯誤：

```
InvalidActionDeclarationException: ActionType (Category: 'Deploy',
Provider: 'CloudFormationStackSet', Owner: 'AWS, Version: '1') in action
'Deploy' is not available in region 'AF_SOUTH_1'
```

可能的修正：請參考下列注意事項，以在可用動作的「區域」中使用動作。

> **Note**
>
> 該`CloudFormationStackSet`和`CloudFormationStackInstances`行動不適用於亞太區域 (香港)、歐洲 (蘇黎世)、歐洲 (米蘭)、非洲 (開普敦) 及中東 (巴林) 等地區。若要參考其他可用動作，請參閱產品與服務整合 CodePipeline。

如需這些動作的詳細資訊，請參閱 AWS CloudFormation StackSets。

當達到執行限制時，執行模式變更為 QUEED 或平行模式的管線會失敗

問題：QUEED 模式下管線的並行執行數目上限為 50 個執行。達到此限制時，管線會失敗，而不顯示狀態訊息。

可能的修正：編輯執行模式的管線定義時，請將編輯與其他編輯動作分開。

如需有關 QUEED 或平行執行模式的詳細資訊，請參閱CodePipeline 概念 。

如果在變更為「佇列」或「已取代」模式時進行編輯，則「平行」模式下的管線定義已過期

問題：對於 parallel 模式下的配管，將配管執行模式編輯為 QUEED 或已取代時，並行模式的配管定義將不會更新。更新平行模式時更新的管線定義不會在「已取代」或「佇列」模式中使用

可能的修正：對於 parallel 模式下的管線，將配管執行模式編輯為 QUEED 或已取代時，請避免同時更新配管定義。

如需有關 QUEED 或平行執行模式的詳細資訊，請參閱CodePipeline 概念 。

從平行模式變更的管線會顯示先前的執行模式

問題：對於平行模式下的配管，將配管執行模式編輯為「佇列」或「已取代」時，配管狀態不會將更新狀態顯示為「平行」。如果管線從「平行」變更為「佇列」或「已取代」，則處於「已取代」或「佇列」模式下的管線狀態將會是這些模式中的最後一個已知狀態。如果管道之前從未在該模式下運行，則狀態將為空。

可能的修正：對於 parallel 模式的管線，將配管執行模式編輯為 QUEED 或已取代時，請注意，執行模式顯示不會顯示「平行」狀態。

如需有關 QUEED 或平行執行模式的詳細資訊，請參閱CodePipeline 概念 。

具有使用依檔案路徑觸發程序篩選的連線的管線可能無法在分支建立時啟動

說明：針對含有使用連線之來源動作的管線 (例如 BitBucket 來源動作)，您可以使用 Git 設定來設定觸發程序，讓您依檔案路徑篩選以啟動管道。在某些情況下，對於具有在檔案路徑上篩選之觸發程序的管線，當第一次建立具有檔案路徑篩選器的分支時，管線可能不會啟動，因為這不允許CodeConnections連線解析已變更的檔案。當觸發器的 Git 配置設置為對文件路徑進行過濾時，管道將不會在源存儲庫中創建具有過濾器的分支時啟動，如需有關過濾文件路徑的更多信息，請參閱篩選程式碼推送或提取要求的觸發程序。

結果：例如，在 CodePipeline 分支「B」上具有檔案路徑篩選器的管線不會在建立分支「B」時觸發。如果沒有檔案路徑篩選器，管線仍會啟動。

當達到檔案限制時，具有使用依檔案路徑觸發程序篩選的連線的管線可能無法啟動

說明：針對含有使用連線之來源動作的管線 (例如 BitBucket 來源動作)，您可以使用 Git 設定來設定觸發程序，讓您依檔案路徑篩選以啟動管道。 CodePipeline最多可擷取前 100 個檔案；因此，當將觸發程序的 Git 組態設定為篩選檔案路徑時，如果有超過 100 個檔案，管線可能無法啟動。如需有關篩選檔案路徑的詳細資訊，請參閱篩選程式碼推送或提取要求的觸發程序。

結果：例如，如果差異包含 150 個檔案，請查 CodePipeline 看前 100 個檔案 (無特定順序)，以根據指定的檔案路徑篩選器進行檢查。如果與檔案路徑篩選器相符的檔案不在擷取的 100 個檔案中 CodePipeline，則不會叫用管線

需要不同問題的協助嗎？

嘗試其他資源：

- 聯絡 AWS 支援。
- 在 CodePipeline 論壇發問。
- 請求提高配額。如需詳細資訊，請參閱 AWS CodePipeline 中的配額。

> **Note**
>
> 最多可能需要兩週時間來處理提高配額的請求。

中的安全性 AWS CodePipeline

雲安全 AWS 是最高的優先級。身為 AWS 客戶，您可以從資料中心和網路架構中獲益，該架構能夠滿足對安全性最敏感的組織的需求。

安全是 AWS 與您之間共同承擔的責任。共同責任模型 將此描述為雲端的安全和雲端內的安全：

- 雲端的安全性 — AWS 負責保護 AWS 服務 中執行的基礎架構 AWS 雲端。 AWS 還為您提供可以安全使用的服務。在 AWS 合規計畫中，第三方稽核員會定期測試並驗證我們的安全功效。若要瞭解適用的法規遵循方案 AWS CodePipeline，請參閱AWS 服務 合規方案的範圍。
- 雲端中的安全性 — 您的責任取決於您使用的資料。 AWS 服務 您也必須對其他因素負責，包括資料的機密性、您公司的要求和適用法律和法規。

本文件可協助您瞭解如何在使用時套用共同責任模型 CodePipeline。下列主題說明如何設定 CodePipeline 以符合安全性與合規性目標。您還將學習如何使用其 AWS 服務 他幫助您監控和保護 CodePipeline 資源的其他方法。

主題

- 資料保護 AWS CodePipeline
- 適用於 AWS CodePipeline的 Identity and Access Management
- 登錄和監控 CodePipeline
- 符合性驗證 AWS CodePipeline
- 韌性 AWS CodePipeline
- 基礎結構安全 AWS CodePipeline
- 安全最佳實務

資料保護 AWS CodePipeline

AWS 共用責任模型適用於中的資料保護 AWS CodePipeline。如此模型中所述， AWS 負責保護執行所有 AWS 雲端. 您負責維護在此基礎設施上託管內容的控制權。您還必須負責您所使用 AWS 服務 的安全組態和管理任務。如需資料隱私權的詳細資訊，請參閱資料隱私權常見問答集。如需歐洲資料保護的相關資訊，請參閱 AWS 安全性部落格上的 AWS 共同責任模型和 GDPR 部落格文章。

基於資料保護目的，我們建議您使用 AWS IAM Identity Center 或 AWS Identity and Access Management (IAM) 保護 AWS 帳戶 登入資料並設定個別使用者。如此一來，每個使用者都只會獲得授與完成其任務所必須的許可。我們也建議您採用下列方式保護資料：

- 每個帳戶均要使用多重要素驗證 (MFA)。
- 使用 SSL/TLS 與 AWS 資源進行通訊。我們需要 TLS 1.2 並建議使用 TLS 1.3。
- 使用設定 API 和使用者活動記錄 AWS CloudTrail。
- 使用 AWS 加密解決方案以及其中的所有默認安全控制 AWS 服務。
- 使用進階的受管安全服務 (例如 Amazon Macie)，協助探索和保護儲存在 Amazon S3 的敏感資料。
- 如果您在透過命令列介面或 API 存取時需要經 AWS 過 FIPS 140-2 驗證的加密模組，請使用 FIPS 端點。如需 FIPS 和 FIPS 端點的相關資訊，請參閱聯邦資訊處理標準 (FIPS) 140-2 概觀。

我們強烈建議您絕對不要將客戶的電子郵件地址等機密或敏感資訊，放在標籤或自由格式的文字欄位中，例如名稱 欄位。這包括當您使用主控台、API CodePipeline 或 AWS SDK 時 AWS 服務 使用或其他使用時。 AWS CLI您在標籤或自由格式文字欄位中輸入的任何資料都可能用於計費或診斷日誌。如果您提供外部伺服器的 URL，我們強烈建議請勿在驗證您對該伺服器請求的 URL 中包含憑證資訊。

以下安全最佳實務也可用來解決 CodePipeline 中的資料保護問題：

- 為 Amazon S3 中存放的成品設定伺服器端加密 CodePipeline
- 用 AWS Secrets Manager 於追蹤資料庫密碼或第三方 API 金鑰

網際網路流量隱私權

Amazon VPC 可用於 AWS 服務 在您定義的虛擬網路 (虛擬私有雲) 中啟動 AWS 資源。 CodePipeline 支援 Amazon VPC 端點 AWS PrivateLink，這項 AWS 技術可促進 AWS 服務 使用具有私有 IP 地址的 elastic network interface 之間的私有通訊。這意味著您可以通 CodePipeline 過 VPC 中的私有端點直接連接，從而將所有流量保留在 VPC 和網絡中 AWS 。過去，VPC 內執行的應用程式需要存取網際網路，才能連接至 CodePipeline。使用 VPC，您可以控制網路設定，例如：

- IP 位址範圍
- 子網路，
- 路線表，以及
- 網路閘道。

若要將 VPC 連接到 CodePipeline，您需要為的定義介面 VPC 端點。 CodePipeline這種類型的端點使您可以將 VPC 連接到 AWS 服務。端點提供可靠、可擴充的連線能力， CodePipeline 無需網際網路閘道、網路位址轉譯 (NAT) 執行個體或 VPN 連線。如需設定 VPC 的相關資訊，請參閱 VPC 使用者指南。

靜態加密

中的資料 CodePipeline 在靜態時使用 AWS KMS keys. 程式碼成品會儲存在客戶擁有的 S3 儲存貯體中，並使用 AWS 受管金鑰 或客戶受管金鑰加密。如需詳細資訊，請參閱 為 Amazon S3 中存放的成品設定伺服器端加密 CodePipeline。

傳輸中加密

所有 service-to-service 通訊在傳輸過程中都使用 SSL/TLS 進行加密。

加密金鑰管理

如果您選擇加密程式碼加工品的預設選項，請 CodePipeline 使用 AWS 受管金鑰. 您無法變更或刪除此項 AWS 受管金鑰。如果您使用客戶受管金鑰 AWS KMS 來加密或解密 S3 儲存貯體中的成品，您可以視需要變更或輪換此客戶受管金鑰。

> **Important**
>
> CodePipeline 僅支援對稱 KMS 金鑰。請勿使用非對稱 KMS 金鑰來加密 S3 儲存貯體中的資料。

為 Amazon S3 中存放的成品設定伺服器端加密 CodePipeline

有兩種方法可以為 Amazon S3 成品設定伺服器端加密：

- CodePipeline 使用「建立管道」精靈建立管道 AWS 受管金鑰 時，會建立 S3 成品儲存貯體並預設值。會與物件資料 AWS 受管金鑰 一起加密，並由 AWS.
- 您可以建立和管理自己的客戶管理金鑰。

> **Important**
>
> CodePipeline 僅支援對稱 KMS 金鑰。請勿使用非對稱 KMS 金鑰來加密 S3 儲存貯體中的資料。

如果您使用預設 S3 金鑰，則無法變更或刪除此金鑰 AWS 受管金鑰。如果您使用客戶受管金鑰 AWS KMS 來加密或解密 S3 儲存貯體中的成品，您可以視需要變更或輪換此客戶受管金鑰。

如果儲存貯體中所存放的所有物件都需要伺服器端加密，則 Amazon S3 支援您可使用的儲存貯體政策。例如，如果要求不包含要求含 SSE-KMS 之伺服器端加密的 `s3:PutObject` 標頭，則下列儲存貯體政策會拒絕向所有人上傳物件 (`x-amz-server-side-encryption`) 的許可。

```
{
    "Version": "2012-10-17",
    "Id": "SSEAndSSLPolicy",
    "Statement": [
        {
            "Sid": "DenyUnEncryptedObjectUploads",
            "Effect": "Deny",
            "Principal": "*",
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::codepipeline-us-west-2-89050EXAMPLE/*",
            "Condition": {
                "StringNotEquals": {
                    "s3:x-amz-server-side-encryption": "aws:kms"
                }
            }
        },
        {
            "Sid": "DenyInsecureConnections",
            "Effect": "Deny",
            "Principal": "*",
            "Action": "s3:*",
            "Resource": "arn:aws:s3:::codepipeline-us-west-2-89050EXAMPLE/*",
            "Condition": {
                "Bool": {
                    "aws:SecureTransport": "false"
                }
            }
        }
    ]
```

```
}
```

如需伺服器端加密的詳細資訊 AWS KMS，請參閱使用伺服器端加密保護資料和使用儲存於 AWS Key Management Service (SSE-KMS) 的 KMS 金鑰使用伺服器端加密保護資料。

如需詳細資訊 AWS KMS，請參閱AWS Key Management Service 開發人員指南。

主題

- 檢視您的 AWS 受管金鑰
- 使用 AWS CloudFormation 或設定 S3 儲存貯體的伺服器端加密 AWS CLI

檢視您的 AWS 受管金鑰

當您使用 Create Pipeline (建立管道) 精靈建立第一個管道時，在您建立管道的相同區域中，將會為您建立 S3 儲存貯體。儲存貯體用於存放管道成品。當管道執行時，S3 儲存貯體中會放入和擷取成品。預設情況下 AWS 受管金鑰 ， CodePipeline AWS KMS 使用伺服器端加密，搭配使用 Amazon S3 (aws/s3金鑰)。這 AWS 受管金鑰 是創建並存儲在您的 AWS 帳戶中。從 S3 儲存貯體擷取人工因素時， CodePipeline 會使用相同的 SSE-KMS 程序來解密成品。

若要檢視您的相關資訊 AWS 受管金鑰

1. 登入 AWS Management Console 並開啟 AWS KMS 主控台。
2. 如果出現歡迎頁面，請選擇 [立即開始使用]。
3. 在服務導覽窗格中，選擇AWS 受管理的金鑰。
4. 選擇管道的「區域」。例如，如果管線是在中建立的us-east-2，請確定篩選器已設定為美國東部 (俄亥俄州)。

   如需有關可用之區域和端點的詳細資訊 CodePipeline，請參閱AWS CodePipeline 端點和配額。
5. 在列表中，選擇帶有用於管道的別名的密鑰（默認情況下，aws/s3）。隨即顯示金鑰的基本資訊。

使用 AWS CloudFormation 或設定 S3 儲存貯體的伺服器端加密 AWS CLI

使用 AWS CloudFormation 或建立管線 AWS CLI 時，必須手動設定伺服器端加密。使用上面的範例儲存貯體政策，然後建立自己的客戶受管金鑰。您也可以使用您自己的金鑰，而不是 AWS 受管金鑰. 選擇您自己的金鑰的一些原因包括：

- 您希望依照排程輪換金鑰，以符合您組織的商業或安全需求。
- 您希望建立管道，使用與另一個 AWS 帳戶建立關聯的資源。這便需要使用客戶受管金鑰。如需詳細資訊，請參閱 在 CodePipeline 中建立使用來自另一個 AWS 帳戶資源的管道。

密碼編譯最佳實務不鼓勵大量重複使用加密金鑰。根據最佳實務，請定期輪換您的金鑰。若要為您的 AWS KMS 金鑰建立新的加密資料，您可以建立客戶管理的金鑰，然後變更應用程式或別名以使用新的客戶管理金鑰。或者，您可以為現有客戶管理的金鑰啟用自動金鑰輪換功能。

若要旋轉您的客戶管理金鑰，請參閱旋轉金鑰。

> **Important**
>
> CodePipeline 僅支援對稱 KMS 金鑰。請勿使用非對稱 KMS 金鑰來加密 S3 儲存貯體中的資料。

用 AWS Secrets Manager 於追蹤資料庫密碼或第三方 API 金鑰

我們建議您在整個生命週期中使用 AWS Secrets Manager 輪換、管理和擷取資料庫登入資料、API 金鑰和其他機密。Secrets Manager 可讓您將程式碼中的硬式編碼認證 (包括密碼) 取代為 Secrets Manager 的 API 呼叫，以程式設計的方式擷取密碼。如需詳細資訊，請參閱什麼是 AWS Secrets Manager？ 在 AWS Secrets Manager 用戶指南。

對於在範本中傳遞密碼參數 (例如 OAuth 認證) 的管道，您應該在 AWS CloudFormation 範本中包含動態參考，以存取您儲存在 Secrets Manager 中的密碼。如需參考 ID 模式和範例，請參閱AWS CloudFormation 使用者指南中的 Secret Secrets Manager 密碼。如需在管線中 GitHub Webhook 的範本程式碼片段中使用動態參考的範例，請參閱 Webhook 資源組態。

另請參閱

下列相關資源可在您管理秘密時提供協助。

- Secrets Manager 可以自動輪替資料庫登入資料，例如對 Amazon RDS 機密進行輪替。如需詳細資訊，請參閱 AWS Secrets Manager 使用者指南中的輪換您的機AWS Secrets Manager。
- 如要檢視將 Secrets Manager 動態參考新增到您 AWS CloudFormation 範本的說明，請參閱 https://aws.amazon.com/blogs/security/how-to-create-and-retrieve-secrets-managed-in-aws-secrets-manager-using-aws-cloudformation-template/。

適用於 AWS CodePipeline的 Identity and Access Management

AWS Identity and Access Management (IAM) 可協助系統管理員安全地控制 AWS 資源存取權。 AWS 服務 IAM 管理員控制哪些人可以驗證 (登入) 和授權 (具有權限) 以使用 CodePipeline 資源。IAM 是您可以使用的 AWS 服務 ，無需額外付費。

主題

物件

您使用 AWS Identity and Access Management (IAM) 的方式會有所不同，具體取決於您在進行的工作 CodePipeline。

服務使用者 — 如果您使用 CodePipeline 服務執行工作，則管理員會為您提供所需的認證和權限。當您使用更多 CodePipeline 功能來完成工作時，您可能需要其他權限。了解存取許可的管理方式可協助您向管理員請求正確的許可。若您無法存取 CodePipeline 中的某項功能，請參閱 對 AWS CodePipeline 身分與存取進行疑難排解。

服務管理員 — 如果您負責公司的 CodePipeline 資源，您可能擁有完整的存取權 CodePipeline。決定您的服務使用者應該存取哪些 CodePipeline 功能和資源是您的工作。接著，您必須將請求提交給您的 IAM 管理員，來變更您服務使用者的許可。檢閱此頁面上的資訊，了解 IAM 的基本概念。若要進一步瞭解貴公司如何搭配使用 IAM CodePipeline，請參閱如何與 IAM AWS CodePipeline 搭配使用。

IAM 管理員：如果您是 IAM 管理員，建議您掌握如何撰寫政策以管理 CodePipeline 存取權的詳細資訊。若要檢視可在 IAM 中使用的 CodePipeline 基於身分的政策範例，請參閱。AWS CodePipeline 身分型政策範例

使用身分驗證

驗證是您 AWS 使用身分認證登入的方式。您必須以 IAM 使用者身分或假設 IAM 角色進行驗證 (登入 AWS)。 AWS 帳戶根使用者

您可以使用透過 AWS 身分識別來源提供的認證，以聯合身分識別身分登入。 AWS IAM Identity Center (IAM 身分中心) 使用者、貴公司的單一登入身分驗證，以及您的 Google 或 Facebook 登入資料都是聯合身分識別的範例。當您以聯合身分登入時，您的管理員先前已設定使用 IAM 角色的聯合身分。當您使 AWS 用同盟存取時，您會間接擔任角色。

根據您的使用者類型，您可以登入 AWS Management Console 或 AWS 存取入口網站。如需有關登入的詳細資訊 AWS，請參閱《AWS 登入 使用指南》 AWS 帳戶中的如何登入您的。

如果您 AWS 以程式設計方式存取，請 AWS 提供軟體開發套件 (SDK) 和命令列介面 (CLI)，以使用您的認證以加密方式簽署您的要求。如果您不使用 AWS 工具，則必須自行簽署要求。如需使用建議的方法自行簽署請求的詳細資訊，請參閱 IAM 使用者指南中的簽署 AWS API 請求。

無論您使用何種身分驗證方法，您可能都需要提供額外的安全性資訊。例如， AWS 建議您使用多重要素驗證 (MFA) 來增加帳戶的安全性。若要進一步了解，請參閱《AWS IAM Identity Center 使用者指南》中的多重要素驗證和《IAM 使用者指南》中的在 AWS中使用多重要素驗證 (MFA)。

AWS 帳戶根使用者

當您建立時 AWS 帳戶，您會從一個登入身分開始，該身分可完整存取該帳戶中的所有資源 AWS 服務和資源。此身分稱為 AWS 帳戶 root 使用者，可透過使用您用來建立帳戶的電子郵件地址和密碼登入來存取。強烈建議您不要以根使用者處理日常作業。保護您的根使用者憑證，並將其用來執行只能由根使用者執行的任務。如需這些任務的完整清單，了解需以根使用者登入的任務，請參閱《IAM 使用者指南》中的需要根使用者憑證的任務。

IAM 使用者和群組

IAM 使用者是您內部的身分，具 AWS 帳戶 有單一人員或應用程式的特定許可。建議您盡可能依賴暫時性憑證，而不是擁有建立長期憑證 (例如密碼和存取金鑰) 的 IAM 使用者。但是如果特定使用案例需要擁有長期憑證的 IAM 使用者，建議您輪換存取金鑰。如需詳細資訊，請參閱《IAM 使用者指南》https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html#rotate-credentials中的為需要長期憑證的使用案例定期輪換存取金鑰。

IAM 群組是一種指定 IAM 使用者集合的身分。您無法以群組身分登入。您可以使用群組來一次為多名使用者指定許可。群組可讓管理大量使用者許可的過程變得更為容易。例如，您可以擁有一個名為 IAMAdmins 的群組，並給予該群組管理 IAM 資源的許可。

使用者與角色不同。使用者只會與單一人員或應用程式建立關聯，但角色的目的是在由任何需要它的人員取得。使用者擁有永久的長期憑證，但角色僅提供暫時憑證。若要進一步了解，請參閱《IAM 使用者指南》中的建立 IAM 使用者 (而非角色) 的時機。

IAM 角色

IAM 角色是您 AWS 帳戶 內部具有特定許可的身分。它類似 IAM 使用者，但不與特定的人員相關聯。您可以切換角色，在中暫時擔任 IAM 角色。 AWS Management Console 您可以透過呼叫 AWS CLI 或 AWS API 作業或使用自訂 URL 來擔任角色。如需使用角色的方法詳細資訊，請參閱《IAM 使用者指南》中的使用 IAM 角色。

使用暫時憑證的 IAM 角色在下列情況中非常有用：

- 聯合身分使用者存取 – 若要向聯合身分指派許可，請建立角色，並為角色定義許可。當聯合身分進行身分驗證時，該身分會與角色建立關聯，並獲授予由角色定義的許可。如需聯合角色的相關資訊，請參閱《IAM 使用者指南》中的為第三方身分供應商建立角色。如果您使用 IAM Identity Center，則需要設定許可集。為控制身分驗證後可以存取的內容，IAM Identity Center 將許可集與 IAM 中的角色相關聯。如需有關許可集的資訊，請參閱《AWS IAM Identity Center 使用者指南》中的許可集。
- 暫時 IAM 使用者許可 – IAM 使用者或角色可以擔任 IAM 角色來暫時針對特定任務採用不同的許可。
- 跨帳戶存取：您可以使用 IAM 角色，允許不同帳戶中的某人 (信任的委託人) 存取您帳戶的資源。角色是授予跨帳户存取權的主要方式。但是，對於某些策略 AWS 服務，您可以將策略直接附加到資源（而不是使用角色作為代理）。若要了解跨帳戶存取角色和資源型政策間的差異，請參閱《IAM 使用者指南》中的 IAM 角色與資源類型政策的差異。
- 跨服務訪問 — 有些 AWS 服務 使用其他 AWS 服務功能。例如，當您在服務中進行呼叫時，該服務通常會在 Amazon EC2 中執行應用程式或將物件儲存在 Amazon Simple Storage Service (Amazon S3) 中。服務可能會使用呼叫主體的許可、使用服務角色或使用服務連結角色來執行此作業。
  - 轉寄存取工作階段 (FAS) — 當您使用 IAM 使用者或角色在中執行動作時 AWS，您會被視為主體。使用某些服務時，您可能會執行某個動作，進而在不同服務中啟動另一個動作。FAS 會使用主體呼叫的權限 AWS 服務，並結合要求 AWS 服務 向下游服務發出要求。只有當服務收到需要與其 AWS 服務 他資源互動才能完成的請求時，才會發出 FAS 請求。在此情況下，您必須具有執行這兩個動作的許可。如需提出 FAS 請求時的政策詳細資訊，請參閱《轉發存取工作階段》。
  - 服務角色：服務角色是服務擔任的 IAM 角色，可代表您執行動作。IAM 管理員可以從 IAM 內建立、修改和刪除服務角色。如需詳細資訊，請參閱《IAM 使用者指南》中的建立角色以委派許可給 AWS 服務。
  - 服務連結角色 — 服務連結角色是連結至. AWS 服務服務可以擔任代表您執行動作的角色。服務連結角色會顯示在您的中， AWS 帳戶 且屬於服務所有。IAM 管理員可以檢視，但不能編輯服務連結角色的許可。

- 在 Amazon EC2 上執行的應用程式 — 您可以使用 IAM 角色管理在 EC2 執行個體上執行的應用程式以及發出 AWS CLI 或 AWS API 請求的臨時登入資料。這是在 EC2 執行個體內存放存取金鑰的較好方式。若要將 AWS 角色指派給 EC2 執行個體並提供給其所有應用程式，請建立連接至執行個體的執行個體設定檔。執行個體設定檔包含該角色，並且可讓 EC2 執行個體上執行的程式取得臨時性憑證。如需詳細資訊，請參閱《IAM 使用者指南》中的利用 IAM 角色來授予許可給 Amazon EC2 執行個體上執行的應用程式。

若要了解是否要使用 IAM 角色或 IAM 使用者，請參閱《IAM 使用者指南》中的建立 IAM 角色 (而非使用者) 的時機。

使用政策管理存取權

您可以透 AWS 過建立原則並將其附加至 AWS 身分識別或資源來控制中的存取。原則是一個物件 AWS ，當與身分識別或資源相關聯時，會定義其權限。 AWS 當主參與者 (使用者、root 使用者或角色工作階段) 提出要求時，評估這些原則。政策中的許可決定是否允許或拒絕請求。大多數原則會 AWS 以 JSON 文件的形式儲存在中。如需 JSON 政策文件結構和內容的相關資訊，請參閱《IAM 使用者指南》中的 JSON 政策概觀。

管理員可以使用 AWS JSON 政策來指定誰可以存取哪些內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

預設情況下，使用者和角色沒有許可。若要授予使用者對其所需資源執行動作的許可，IAM 管理員可以建立 IAM 政策。然後，管理員可以將 IAM 政策新增至角色，使用者便能擔任這些角色。

IAM 政策定義該動作的許可，無論您使用何種方法來執行操作。例如，假設您有一個允許 `iam:GetRole` 動作的政策。具有該原則的使用者可以從 AWS Management Console AWS CLI、或 AWS API 取得角色資訊。

身分型政策

身分型政策是可以連接到身分 (例如 IAM 使用者、使用者群組或角色) 的 JSON 許可政策文件。這些政策可控制身分在何種條件下能對哪些資源執行哪些動作。若要了解如何建立身分類型政策，請參閱《IAM 使用者指南》中的建立 IAM 政策。

身分型政策可進一步分類成內嵌政策或受管政策。內嵌政策會直接內嵌到單一使用者、群組或角色。受管理的策略是獨立策略，您可以將其附加到您的 AWS 帳戶. 受管政策包括 AWS 受管政策和客戶管理的策略。如需瞭解如何在受管政策及內嵌政策間選擇，請參閱 IAM 使用者指南中的在受管政策和內嵌政策間選擇。

資源型政策

資源型政策是連接到資源的 JSON 政策文件。資源型政策的最常見範例是 IAM 角色信任政策和 Amazon S3 儲存貯體政策。在支援資源型政策的服務中，服務管理員可以使用它們來控制對特定資源的存取權限。對於附加政策的資源，政策會定義指定的主體可以對該資源執行的動作以及在何種條件下執行的動作。您必須在資源型政策中指定主體。主參與者可以包括帳戶、使用者、角色、同盟使用者或。 AWS 服務

資源型政策是位於該服務中的內嵌政策。您無法在以資源為基礎的政策中使用 IAM 的 AWS 受管政策。

其他政策類型

AWS 支援其他較不常見的原則類型。這些政策類型可設定較常見政策類型授予您的最大許可。

- 許可界限：許可界限是一種進階功能，可供您設定身分型政策能授予 IAM 實體 (IAM 使用者或角色) 的最大許可。您可以為實體設定許可界限。所產生的許可會是實體的身分型政策和其許可界限的交集。會在 `Principal` 欄位中指定使用者或角色的資源型政策則不會受到許可界限的限制。所有這類政策中的明確拒絕都會覆寫該允許。如需許可邊界的相關資訊，請參閱《IAM 使用者指南》中的 IAM 實體許可邊界。
- 服務控制策略（SCP）— SCP 是 JSON 策略，用於指定中組織或組織單位（OU）的最大權限。AWS Organizations AWS Organizations 是一種用於分組和集中管理您企業擁 AWS 帳戶 有的多個服務。若您啟用組織中的所有功能，您可以將服務控制政策 (SCP) 套用到任何或所有帳戶。SCP 限制成員帳戶中實體的權限，包括每個 AWS 帳戶根使用者帳戶。如需 Organizations 和 SCP 的相關資訊，請參閱《AWS Organizations 使用者指南》中的 SCP 運作方式。
- 工作階段政策：工作階段政策是一種進階政策，您可以在透過編寫程式的方式建立角色或聯合身分使用者的暫時工作階段時，作為參數傳遞。所產生工作階段的許可會是使用者或角色的身分型政策和工作階段政策的交集。許可也可以來自資源型政策。所有這類政策中的明確拒絕都會覆寫該允許。如需詳細資訊，請參閱《IAM 使用者指南》中的工作階段政策。

如何與 IAM AWS CodePipeline 搭配使用

在您使用 IAM 管理存取權限之前 CodePipeline，您應該瞭解哪些 IAM 功能可搭配使用 CodePipeline。若要深入瞭解如何 CodePipeline 與 IAM 搭配使用的方式和其他 AWS 服務 方式 AWS 服務 ，請參閱 IAM 使用者指南中的 IAM。

主題

- CodePipeline 身分型政策

- CodePipeline 資源型政策
- 以 CodePipeline 標籤為基礎的授權
- CodePipeline IAM 角色

CodePipeline 身分型政策

使用 IAM 身分型政策，您可以指定允許或拒絕的動作和資源，以及在何種條件下允許或拒絕動作。CodePipeline 支援特定動作、資源和條件索引鍵。若要了解您在 JSON 政策中使用的所有元素，請參閱《IAM 使用者指南》中的 JSON 政策元素參考。

動作

管理員可以使用 AWS JSON 政策來指定誰可以存取哪些內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

JSON 政策的 `Action` 元素描述您可以用來允許或拒絕政策中存取的動作。原則動作通常與關聯的 AWS API 作業具有相同的名稱。有一些例外狀況，例如沒有相符的 API 操作的僅限許可動作。也有一些操作需要政策中的多個動作。這些額外的動作稱為相依動作。

政策會使用動作來授與執行相關聯操作的許可。

中的策略動作在動作之前 CodePipeline 使用下列前置詞：`codepipeline:`。

例如，若要准許某人檢視帳戶中現有的管道，您需要在其政策中加入 `codepipeline:GetPipeline` 動作。原則陳述式必須包含`Action`或`NotAction`元素。 CodePipeline 定義了它自己的一組動作，描述您可以使用此服務執行的任務。

若要在單一陳述式中指定多個動作，請用逗號分隔，如下所示：

```
"Action": [
      "codepipeline:action1",
      "codepipeline:action2"
```

您也可以使用萬用字元 (*) 來指定多個動作。例如，若要指定開頭是 `Get` 文字的所有動作，請包含以下動作：

```
"Action": "codepipeline:Get*"
```

如需 CodePipeline 動作清單，請參閱《IAM 使用者指南》 AWS CodePipeline中的定義動作。

資源

管理員可以使用 AWS JSON 政策來指定誰可以存取哪些內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

Resource JSON 政策元素可指定要套用動作的物件。陳述式必須包含 Resource 或 NotResource 元素。最佳實務是使用其 Amazon Resource Name (ARN) 來指定資源。您可以針對支援特定資源類型的動作 (稱為資源層級許可) 來這麼做。

對於不支援資源層級許可的動作 (例如列出作業)，請使用萬用字元 (*) 來表示陳述式適用於所有資源。

```
"Resource": "*"
```

CodePipeline 資源與營運

在中 CodePipeline，主要資源是管線。在政策中，您可以使用 Amazon 資源名稱 (ARN) 來識別該政策適用的資源。 CodePipeline 支援可與主要資源搭配使用的其他資源，例如階段、動作和自訂動作。這些資源稱為「子資源」。這些資源和子資源各與唯一的 Amazon Resource Name (ARN) 相關聯。如需 ARN 的詳細資訊，請參閱中的 Amazon 資源名稱 (ARN) 和 AWS 服務 命名空間。Amazon Web Services 一般參考若要取得與管線相關聯的管線 ARN，您可以在主控台的「設定」 下找到管線 ARN。如需詳細資訊，請參閱 檢視管線 ARN 和服務角色 ARN (主控台)。

資源類型	ARN 格式
管道	*AR: aws: #####:##:##:####*
階段	*arn: aws: #####:##:##:####/####*
動作	*arn: aws: #####:##:##:####/####/####*
自訂動作	*AR: awn: #####:##:##:####:###/##/###/##*
所有 CodePipeline 資源	ARN: aws: 程式碼管線:*
指定區域中指定帳號擁有的所有 CodePipeline 資源	*AR: aws: #####:##:##:**

> **Note**
>
> 大多數服務都 AWS 將冒號 (:) 或正斜杠 (/) 視為 ARN 中的相同字符。但是，在事件模式和規則中 CodePipeline 使用完全匹配。在建立事件模式時，請務必使用正確的 ARN 字元，使這些字元符合您要匹配管道中的 ARN 語法。

在中 CodePipeline，有一些支援資源層級權限的 API 呼叫。資源層級許可表示 API 呼叫是否能指定資源 ARN，或是 API 呼叫是否可以使用萬用字元指定所有資源。CodePipeline 權限參考如需資源層級權限的詳細說明，以及支援資源層級權限的 CodePipeline API 呼叫清單，請參閱。

例如，您可以指出您陳述式中的特定管道 (*myPipeline*) 其他使用其 ARN：

```
"Resource": "arn:aws:codepipeline:us-east-2:111222333444:myPipeline"
```

您也可以透過使用 (*) 萬用字元指定所有屬於特定帳戶的管道，如下所示：

```
"Resource": "arn:aws:codepipeline:us-east-2:111222333444:*"
```

若要指定所有資源，或如果特定的 API 動作不支援 ARN，請在 `Resource` 元素中使用 (*) 萬用字元，如下所示：

```
"Resource": "*"
```

> **Note**
>
> 建立 IAM 政策時，請遵循授予最低權限的標準安全建議，亦即僅授予執行任務所需的許可。若 API 呼叫支援 ARN，表示其支援資源層級許可，您無須使用 (*) 萬用字元。

某些 CodePipeline API 呼叫會接受多個資源 (例如`GetPipeline`)。若要在單一陳述式中指定多個資源，請用逗號分隔他們的 ARN，如下所示：

```
"Resource": ["arn1", "arn2"]
```

CodePipeline 提供了一組操作來處理資 CodePipeline 源。如需可用操作的清單，請參閱 CodePipeline 權限參考。

條件索引鍵

管理員可以使用 AWS JSON 政策來指定誰可以存取哪些內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

Condition 元素 (或 Condition 區塊)可讓您指定使陳述式生效的條件。Condition 元素是選用項目。您可以建立使用條件運算子的條件運算式 (例如等於或小於)，來比對政策中的條件和請求中的值。

若您在陳述式中指定多個 Condition 元素，或是在單一 Condition 元素中指定多個索引鍵， AWS 會使用邏輯 AND 操作評估他們。如果您為單一條件索引鍵指定多個值，請使用邏輯OR運算來 AWS 評估條件。必須符合所有條件，才會授與陳述式的許可。

您也可以在指定條件時使用預留位置變數。例如，您可以只在使用者使用其 IAM 使用者名稱標記時，將存取資源的許可授予該 IAM 使用者。如需詳細資訊，請參閱《IAM 使用者指南》中的 IAM 政策元素：變數和標籤。

AWS 支援全域條件金鑰和服務特定條件金鑰。若要查看所有 AWS 全域條件金鑰，請參閱《IAM 使用者指南》中的AWS 全域條件內容金鑰。

CodePipeline 定義了它自己的一組條件鍵，並且還支持使用一些全局條件鍵。若要查看所有 AWS 全域條件金鑰，請參閱 IAM 使用者指南中的AWS 全域條件內容金鑰。

所有 Amazon EC2 操作都支援 aws:RequestedRegion 和 ec2:Region 條件索引鍵。如需詳細資訊，請參閱範例：將存取限制在特定區域。

若要查看 CodePipeline 條件金鑰清單，請參閱《IAM 使用者指南》 AWS CodePipeline中的條件金鑰。若要瞭解您可以使用條件索引鍵的動作和資源，請參閱動作定義者 AWS CodePipeline。

範例

若要檢視以 CodePipeline 身分為基礎的原則範例，請參閱。AWS CodePipeline 身分型政策範例

CodePipeline 資源型政策

CodePipeline 不支援以資源為基礎的政策。但 CodePipeline 是，針對與之相關的 S3 服務提供以資源為基礎的政策範例。

範例

要查看 CodePipeline 以資源為基礎的策略的實例，請參閱AWS CodePipeline 資源型政策範例，

以 CodePipeline 標籤為基礎的授權

您可以將標籤附加至 CodePipeline 資源，或將要求中的標籤傳遞給 CodePipeline。若要根據標籤控制存取，請使用 `codepipeline:ResourceTag/key-name`、`aws:RequestTag/key-name` 或 `aws:TagKeys` 條件金鑰，在政策的條件元素中，提供標籤資訊。如需標記 CodePipeline 資源的詳細資訊，請參閱 標記 資源。

若要檢視身分型原則範例，以根據該資源上的標籤來限制存取資源，請參閱使用標籤來控制對 CodePipeline 資源的存取。

CodePipeline IAM 角色

IAM 角色是您 AWS 帳戶中具有特定許可的實體。

使用臨時登入資料 CodePipeline

您可以搭配聯合使用暫時憑證、擔任 IAM 角色，或是擔任跨帳戶角色。您可以透過呼叫AssumeRole或等 AWS STS API 作業來取得臨時安全登入資料GetFederationToken。

CodePipeline 支援使用臨時認證。

服務角色

CodePipeline 允許服務代表您擔任服務角色。此角色可讓服務存取其他服務中的資源，以代表您完成動作。服務角色會出現在您的 IAM 帳戶中，且由該帳戶所擁有。這表示 IAM 管理員可以變更此角色的許可。不過，這樣可能會破壞此服務的功能。

CodePipeline 支援服務角色。

AWS CodePipeline 身分型政策範例

根據預設，IAM 使用者和角色不具備建立或修改 CodePipeline 資源的許可。他們也無法使用 AWS Management Console AWS CLI、或 AWS API 執行工作。IAM 管理員必須建立 IAM 政策，授予使用者和角色在指定資源上執行特定 API 操作的所需許可。管理員接著必須將這些政策連接至需要這些許可的 IAM 使用者或群組。

若要了解如何使用這些範例 JSON 原則文件建立 IAM 身分型原則，請參閱《IAM 使用者指南》中的在 JSON 標籤上建立原則。

若要瞭解如何建立使用其他帳戶資源的管道，以及相關範例政策，請參閱在 CodePipeline 中建立使用來自另一個 AWS 帳戶資源的管道。

主題

政策最佳實務

以身分識別為基礎的政策會決定某人是否可以建立、存取或刪除您帳戶中的 CodePipeline 資源。這些動作可能會讓您的 AWS 帳戶產生費用。當您建立或編輯身分型政策時，請遵循下列準則及建議事項：

- 開始使用 AWS 受管原則並邁向最低權限權限 — 若要開始將權限授與使用者和工作負載，請使用可授與許多常見使用案例權限的AWS 受管理原則。它們可用在您的 AWS 帳戶. 建議您透過定義特定於您使用案例的 AWS 客戶管理政策，進一步降低使用權限。如需詳細資訊，請參閱 IAM 使用者指南中的 AWS 受管政策或任務職能的AWS 受管政策。
- 套用最低權限許可：設定 IAM 政策的許可時，請僅授予執行任務所需的許可。為實現此目的，您可以定義在特定條件下可以對特定資源採取的動作，這也稱為最低權限許可。如需如何使用 IAM 套用許可的詳細資訊，請參閱 IAM 使用者指南中的 IAM 中的政策和許可。
- 使用 IAM 政策中的條件進一步限制存取權 – 您可以將條件新增至政策，以限制動作和資源的存取。例如，您可以撰寫政策條件，指定必須使用 SSL 傳送所有請求。您也可以使用條件來授與對服務動作的存取權 (如透過特定) 使用這些動作 AWS 服務，例如 AWS CloudFormation。如需詳細資訊，請參閱《IAM 使用者指南》中的 IAM JSON 政策元素：條件。
- 使用 IAM Access Analyzer 驗證 IAM 政策，確保許可安全且可正常運作 – IAM Access Analyzer 驗證新政策和現有政策，確保這些政策遵從 IAM 政策語言 (JSON) 和 IAM 最佳實務。IAM Access Analyzer 提供 100 多項政策檢查及切實可行的建議，可協助您編寫安全且實用的政策。如需詳細資訊，請參閱《IAM 使用者指南》中的 IAM Access Analyzer 政策驗證。
- 需要多因素身份驗證 (MFA) — 如果您的案例需要 IAM 使用者或根使用者 AWS 帳戶，請開啟 MFA 以獲得額外的安全性。如需在呼叫 API 操作時請求 MFA，請將 MFA 條件新增至您的政策。如需詳細資訊，請參閱《IAM 使用者指南》中的設定 MFA 保護的 API 存取。

如需 IAM 中最佳實務的相關資訊，請參閱 IAM 使用者指南中的 IAM 安全最佳實務。

在主控台檢視資源

CodePipeline 控制台需要有`ListRepositories`權限才能在您登錄的 AWS 地區中顯示您 AWS 帳戶的存儲庫列表。主控台還包含 Go to resource (移至資源) 功能，可快速執行不區分大小寫的資源搜尋。這項搜尋會在您登入所在 AWS 地區的 AWS 帳戶中執行。將會跨以下服務來顯示以下資源：

- AWS CodeBuild：組建專案
- AWS CodeCommit：儲存庫
- AWS CodeDeploy：應用程式
- AWS CodePipeline：管道

若要跨所有服務中的資源執行此搜尋，您必須擁有以下許可：

- CodeBuild: `ListProjects`
- CodeCommit: `ListRepositories`
- CodeDeploy: `ListApplications`
- CodePipeline: `ListPipelines`

如果您沒有某項服務的許可，則不會傳回該服務之資源的結果。即使您有許可來檢視資源，但如果有明確的 `Deny` 而無法檢視部分資源，則不會傳回這些資源。

允許使用者檢視他們自己的許可

此範例會示範如何建立政策，允許 IAM 使用者檢視附加到他們使用者身分的內嵌及受管政策。此原則包含在主控台上或以程式設計方式使用 AWS CLI 或 AWS API 完成此動作的權限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
```

```
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

以身分識別為基礎的政策 (IAM) 範例

您可以將政策連接到 IAM 身分。例如，您可以執行下列動作：

- 將權限原則附加至帳戶中的使用者或群組 — 若要授與使用者在 CodePipeline 主控台中檢視管道的權限，您可以將權限原則附加至使用者所屬的使用者或群組。
- 將許可政策連接至角色 (授予跨帳戶許可)：您可以將身分識別型許可政策連接至 IAM 角色，藉此授予跨帳戶許可。例如，帳戶 A 中的系統管理員可以建立角色，將跨帳戶權限授與另一個 AWS 帳戶 (例如，帳戶 B)， AWS 服務 如下所示：
  1. 帳戶 A 管理員建立 IAM 角色，並將許可政策連接到可授與帳戶 A 中資源許可的角色。
  2. 帳戶 A 管理員將信任政策連接至該角色，識別帳戶 B 做為可擔任該角的委託人。
  3. 然後，帳戶 B 管理員可以將擔任角色的權限委派給帳戶 B 中的任何使用者。這樣做可讓帳戶 B 中的使用者建立或存取帳戶 A 中的資源。如果您想要授與擔任該角色的 AWS 服務 權限，則信任策略中的 AWS 服務 主參與者也可以是主參與者。

  如需使用 IAM 來委派許可的相關資訊，請參閱《IAM 使用者指南》中的存取管理。

以下顯示權限原則的範例，此原則會授與權限，以停用及啟用管線中指定之所有階段之間`MyFirstPipeline`的轉換`us-west-2 region`：

```
{
  "Version": "2012-10-17",
  "Statement" : [
    {
      "Effect" : "Allow",
      "Action" : [
        "codepipeline:EnableStageTransition",
        "codepipeline:DisableStageTransition"
      ],
      "Resource" : [
        "arn:aws:codepipeline:us-west-2:111222333444:MyFirstPipeline/*"
      ]
    }
  ]
}
```

下列範例顯示 111222333444 帳戶中的政策，允許使用者在 CodePipeline 主控台檢視 (但無法變更) 名為 MyFirstPipeline 的管道。此政策以 AWSCodePipeline_ReadOnlyAccess 受管政策為基礎，但因為是 MyFirstPipeline 管道所特有，因此無法直接使用受管政策。若您不希望將政策限制在特定管道，請考慮使用 CodePipeline 所建立及維護的其中一個受管政策。如需詳細資訊，請參閱處理受管政策的相關文章。您必須將此政策附加到為存取而建立的 IAM 角色，例如名為CrossAccountPipelineViewers：

```
{
  "Statement": [
    {
      "Action": [
        "codepipeline:GetPipeline",
        "codepipeline:GetPipelineState",
        "codepipeline:GetPipelineExecution",
        "codepipeline:ListPipelineExecutions",
        "codepipeline:ListActionExecutions",
        "codepipeline:ListActionTypes",
        "codepipeline:ListPipelines",
        "codepipeline:ListTagsForResource",
        "iam:ListRoles",
        "s3:ListAllMyBuckets",
        "codecommit:ListRepositories",
        "codedeploy:ListApplications",
        "lambda:ListFunctions",
        "codestar-notifications:ListNotificationRules",
        "codestar-notifications:ListEventTypes",
```

```
        "codestar-notifications:ListTargets"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:codepipeline:us-west-2:111222333444:MyFirstPipeline"
    },
    {
      "Action": [
        "codepipeline:GetPipeline",
        "codepipeline:GetPipelineState",
        "codepipeline:GetPipelineExecution",
        "codepipeline:ListPipelineExecutions",
        "codepipeline:ListActionExecutions",
        "codepipeline:ListActionTypes",
        "codepipeline:ListPipelines",
        "codepipeline:ListTagsForResource",
        "iam:ListRoles",
        "s3:GetBucketPolicy",
        "s3:GetObject",
        "s3:ListBucket",
        "codecommit:ListBranches",
        "codedeploy:GetApplication",
        "codedeploy:GetDeploymentGroup",
        "codedeploy:ListDeploymentGroups",
        "elasticbeanstalk:DescribeApplications",
        "elasticbeanstalk:DescribeEnvironments",
        "lambda:GetFunctionConfiguration",
        "opsworks:DescribeApps",
        "opsworks:DescribeLayers",
        "opsworks:DescribeStacks"
      ],
      "Effect": "Allow",
      "Resource": "*"
    },
    {
      "Sid": "CodeStarNotificationsReadOnlyAccess",
      "Effect": "Allow",
      "Action": [
        "codestar-notifications:DescribeNotificationRule"
      ],
      "Resource": "*",
      "Condition": {
        "StringLike": {
          "codestar-notifications:NotificationsForResource": "arn:aws:codepipeline:*"
        }
```

```
      }
    }
  ],
  "Version": "2012-10-17"
}
```

建立此政策之後，請在 111222333444 帳戶中建立 IAM 角色，並將該政策附加到該角色。在角色的信任關係中，您必須新增將擔任此角色的 AWS 帳戶。下列範例顯示的策略可讓 111111111111 帳戶的使用者擔任 *1112* 22333444 AWS 帳戶中定義的角色：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111111111111:root"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

下列範例顯示在 *111111111111 ################ 1112* 22333444 AWS 帳戶中指*CrossAccountPipelineViewers*定的角色：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "sts:AssumeRole",
            "Resource": "arn:aws:iam::111222333444:role/CrossAccountPipelineViewers"
        }
    ]
}
```

您可以建立 IAM 政策來限制帳戶中使用者可存取的呼叫和資源，然後將這些政策附加到您的管理使用者。如需有關如何建立 IAM 角色和探索 IAM 政策陳述式範例的詳細資訊 CodePipeline，請參閱客戶受管政策範例。

使用標籤來控制對 CodePipeline 資源的存取

IAM 政策陳述式中的條件是您用來指定動作所需資源權限的語法的一部 CodePipeline 分。在條件中使用標記是控制資源和請求的存取權限的方式之一。如需標記資 CodePipeline 源的資訊，請參閱標記 資源。本主題討論的是標記型的存取控制。

設計 IAM 政策時，您可能會透過授予對特定資源的存取來設定精細許可。隨著您管理的資源數量增加，此任務變得越來越困難。標記資源並在政策陳述式條件中使用標籤，可讓此任務更輕鬆。您可以對具有特定標籤的任何資源大量授予存取。然後，您會在建立期間或之後，對相關資源重複套用此標籤。

可以將標記連接到資源或在請求中將標記傳遞至支援標記的服務。在中 CodePipeline，資源可以有標籤，而某些動作可以包含標籤。在建立 IAM 政策時，可使用標記條件鍵來控制以下項目：

- 可在管道資源上執行動作的使用者 (根據資源已具有的標籤)。
- 可在動作請求中傳遞的標籤。
- 請求中是否可使用特定的標籤鍵。

運用字串條件運算子，您可以建構以索引鍵與字串值的對比為基礎來限制存取的 `Condition` 元素。您可以添加`IfExists`到除 Null 條件以外的任何條件運算符名稱的末尾。如果您是指「如果請求的內容中存在政策索引鍵，則依照政策所述來處理索引鍵。如果該索引鍵不存在，則評估條件元素為 true。」例如，您可以使用`StringEqualsIfExists`來限制其他類型資源中可能不存在的條件鍵。

如需標籤條件金鑰的完整語法和語意，請參閱使用標籤控制存取。如需有關條件索引鍵的其他資訊，請參閱下列資源。本節中的 CodePipeline政策範例會與下列有關條件索引鍵的資訊保持一致，並以細微差別範例 (例 CodePipeline 如資源巢狀) 進行展開。

- 字串條件運算子
- AWS 服務 與 IAM 一起工作
- SCP 語法
- IAM JSON 政策元素：條件
- AWS：RequestTag/標籤鍵
- 條件鍵 CodePipeline

以下範例顯示了如何指定 CodePipeline 使用者政策中的標記條件。

Example 1：根據請求中的標籤限制動作

AWSCodePipeline_FullAccess受管理的使用者原則可提供使用者對任何資源執行任何 CodePipeline 動作的無限權限。

下列原則會限制此功能，並拒絕未經授權的使用者在要求中列出特定標記的管道建立管道的權限。為了達到此種效果，如果該請求指定了名為 Project 的標記，含有 ProjectA 或 ProjectB 的其中一值，其會拒絕 CreatePipeline 動作。(aws:RequestTag 條件索引鍵用來控制哪些標籤可在 IAM 請求中傳遞。)

在下列範例中，原則的目的是拒絕未經授權的使用者使用指定的標籤值建立管道的權限。但是，除了管道本身之外，建立管道還需要存取資源 (例如，管線動作和階段)。由於策略中'Resource'指定的是'*'，因此會根據每個具有 ARN 的資源評估策略，並在建立管線時建立。這些其他資源沒有標籤條件索引鍵，因此StringEquals檢查會失敗，且使用者無法建立任何管線。若要解決此問題，請改用 StringEqualsIfExists 條件運算子。如此一來，僅有在條件索引鍵存在時才會進行測試。

您可以將以下內容讀取為：「如果要檢查的資源具有標籤"RequestTag/Project"條件鍵，則僅在鍵值開頭為時才允許該操作projectA。如果正在檢查的資源沒有該條件索引鍵，則無需擔心它。』

此外，該策略可防止這些未經授權的使用者竄改資源，方法是使用aws:TagKeys條件索引鍵不允許標籤修改動作包含這些相同的標籤值。除了受管使用者政策之外，客戶的管理員還必須將此 IAM 政策附加至未經授權的管理使用者。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "codepipeline:CreatePipeline",
        "codepipeline:TagResource"
      ],
      "Resource": "*",
      "Condition": {
        "StringEqualsIfExists": {
          "aws:RequestTag/Project": ["ProjectA", "ProjectB"]
        }
      }
    },
    {
      "Effect": "Deny",
      "Action": [
```

```
        "codepipeline:UntagResource"
      ],
      "Resource": "*",
      "Condition": {
        "ForAllValues:StringEquals": {
          "aws:TagKeys": ["Project"]
        }
      }
    }
  ]
}
```

Example 2：根據資源標籤限制標記動作

AWSCodePipeline_FullAccess受管理的使用者原則可提供使用者對任何資源執行任何 CodePipeline 動作的無限權限。

以下政策限制此能力，拒絕未經授權的使用者在特定專案管道上執行動作。在作法上，如果資源有名為 Project 的標記，且值為 ProjectA 或 ProjectB，則拒絕某些動作。(aws:ResourceTag 條件索引鍵用於依據資源上的標籤來控制資源的存取。) 除了受管使用者政策之外，客戶的管理員必須將此 IAM 政策連接到未授權的 IAM 使用者。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "codepipeline:TagResource"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Project": ["ProjectA", "ProjectB"]
        }
      }
    }
  ]
}
```

Example 3：根據請求中的標籤允許動作

下列政策會授予使用者許可，在 CodePipeline 中建立開發管道。

若要這樣做，它會在請求指定名為 Project 且值為 ProjectA 的標籤時允許 CreatePipeline 和 TagResource 動作。換句話說，可以指定的唯一標籤鍵是Project，其值必須是ProjectA。

aws:RequestTag條件金鑰可用來控制哪些標籤可以在 IAM 要求中傳遞。aws:TagKeys 條件可確保標籤索引鍵區分大小寫。對於沒有附加AWSCodePipeline_FullAccess受管理的使用者原則的使用者或角色，此原則非常有用。受管理的策略為使用者提供了對任何資源執行任何 CodePipeline 動作的無限權限。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codepipeline:CreatePipeline",
        "codepipeline:TagResource"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestTag/Project": "ProjectA"
        },
        "ForAllValues:StringEquals": {
          "aws:TagKeys": ["Project"]
        }
      }
    }
  ]
}
```

Example 4：根據資源標籤限制取消標記動作

AWSCodePipeline_FullAccess受管理的使用者原則可提供使用者對任何資源執行任何 CodePipeline 動作的無限權限。

以下政策限制此能力，拒絕未經授權的使用者在特定專案管道上執行動作。在作法上，如果資源有名為 Project 的標記，且值為 ProjectA 或 ProjectB，則拒絕某些動作。

此外，該策略可防止這些未經授權的使用者竄改資源，方法是使用aws:TagKeys條件索引鍵不允許標籤修改動作完全移除標Project籤。除了受管使用者政策之外，客戶的管理員還必須將此 IAM 政策附加到未經授權的使用者或角色。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "codepipeline:UntagResource"
      ],
      "Resource": "*",
      "Condition": {
        "ForAllValues:StringEquals": {
          "aws:TagKeys": ["Project"]
        }
      }
    }
  ]
}
```

使用 CodePipeline 主控台所需的許可

若要 CodePipeline 在 CodePipeline 主控台中使用，您必須擁有下列服務的最低權限集：

- AWS Identity and Access Management
- Amazon Simple Storage Service

這些權限可讓您描述 AWS 帳戶的其他 AWS 資源。

根據您納入管道中的其他服務而定，您可能需要以下一或多個項目的許可：

- AWS CodeCommit
- CodeBuild
- AWS CloudFormation
- AWS CodeDeploy
- AWS Elastic Beanstalk
- AWS Lambda
- AWS OpsWorks

如果您建立比最基本必要許可更嚴格的 IAM 政策，則對於採取該 IAM 政策的使用者而言， 主控台就無法如預期運作。若要確保這些使用者仍然可以使用 CodePipeline 主控台，請同時將受`AWSCodePipeline_ReadOnlyAccess`管理的策略附加至使用者，如中所述AWS 受管理的政策 AWS CodePipeline。

對於撥打 AWS CLI 或 CodePipeline API 的使用者，您不需要允許最低主控台權限。

AWS 受管理的政策 AWS CodePipeline

受 AWS 管理的策略是由建立和管理的獨立策略 AWS。 AWS 受管理的策略旨在為許多常見使用案例提供權限，以便您可以開始將權限指派給使用者、群組和角色。

請記住， AWS 受管理的政策可能不會為您的特定使用案例授與最低權限權限，因為這些權限可供所有 AWS 客戶使用。我們建議您定義使用案例專屬的客戶管理政策，以便進一步減少許可。

您無法變更受 AWS 管理策略中定義的權限。如果 AWS 更新 AWS 受管理原則中定義的權限，則此更新會影響附加原則的所有主體識別 (使用者、群組和角色)。 AWS 當新的啟動或新 AWS 服務 的 API 操作可用於現有服務時，最有可能更新 AWS 受管理策略。

如需詳細資訊，請參閱《IAM 使用者指南》中的 AWS 受管政策。

> **Important**
>
> AWS 受管政策`AWSCodePipelineFullAccess`並`AWSCodePipelineReadOnlyAccess`已被替換。使用`AWSCodePipeline_FullAccess`和`AWSCodePipeline_ReadOnlyAccess`策略。

AWS 受管理的策略：**`AWSCodePipeline_FullAccess`**

這是授與完整存取權的政策 CodePipeline。若要在 IAM 主控台中檢視 JSON 政策文件，請參閱AWSCodePipeline_FullAccess。

許可詳細資訊

此政策包含以下許可。

- codepipeline— 授予權限 CodePipeline。
- chatbot— 授與允許主參與者管理中資源的 AWS Chatbot權限。
- cloudformation— 授與權限以允許主體管理中 AWS CloudFormation的資源堆疊。
- cloudtrail— 授與允許主參與者管理中 CloudTrail記錄資源的權限。
- codebuild— 授與允許主參與者存取中 CodeBuild建構資源的權限。
- codecommit— 授與允許主參與者存取中 CodeCommit來源資源的權限。
- codedeploy— 授與權限以允許主參與者存取中 CodeDeploy的部署資源。
- codestar-notifications— 授與權限以允許主參與者存取 AWS CodeStar 「通知」中的資源。
- ec2— 授予許可以允許部署 CodeCatalyst 以管理 Amazon EC2 中的彈性負載平衡。
- ecr— 授予許可以允許存取 Amazon ECR 中的資源。
- elasticbeanstalk— 授與允許主體存取 Elastic Beanstalk 中資源的權限。
- iam— 授予許可以允許主體管理 IAM 中的角色和政策。
- lambda— 授予權限以允許主體管理 Lambda 中的資源。
- events— 授與權限以允許主參與者管理 CloudWatch 事件中的資源。
- opsworks— 授與允許主參與者管理中資源的 AWS OpsWorks權限。
- s3— 授予許可以允許主體管理 Amazon S3 中的資源。
- sns— 授予許可，以允許主體管理 Amazon SNS 中的通知資源。
- states— 授與允許主體檢視中 AWS Step Functions狀態機器的權限。狀態機器由管理工作和狀態之間轉換的狀態集合所組成。

```
{
    "Statement": [
        {
            "Action": [
                "codepipeline:*",
                "cloudformation:DescribeStacks",
                "cloudformation:ListStacks",
                "cloudformation:ListChangeSets",
                "cloudtrail:DescribeTrails",
                "codebuild:BatchGetProjects",
                "codebuild:CreateProject",
```

```
                "codebuild:ListCuratedEnvironmentImages",
                "codebuild:ListProjects",
                "codecommit:ListBranches",
                "codecommit:GetReferences",
                "codecommit:ListRepositories",
                "codedeploy:BatchGetDeploymentGroups",
                "codedeploy:ListApplications",
                "codedeploy:ListDeploymentGroups",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeSubnets",
                "ec2:DescribeVpcs",
                "ecr:DescribeRepositories",
                "ecr:ListImages",
                "ecs:ListClusters",
                "ecs:ListServices",
                "elasticbeanstalk:DescribeApplications",
                "elasticbeanstalk:DescribeEnvironments",
                "iam:ListRoles",
                "iam:GetRole",
                "lambda:ListFunctions",
                "events:ListRules",
                "events:ListTargetsByRule",
                "events:DescribeRule",
                "opsworks:DescribeApps",
                "opsworks:DescribeLayers",
                "opsworks:DescribeStacks",
                "s3:ListAllMyBuckets",
                "sns:ListTopics",
                "codestar-notifications:ListNotificationRules",
                "codestar-notifications:ListTargets",
                "codestar-notifications:ListTagsforResource",
                "codestar-notifications:ListEventTypes",
                "states:ListStateMachines"
            ],
            "Effect": "Allow",
            "Resource": "*",
            "Sid": "CodePipelineAuthoringAccess"
        },
        {
            "Action": [
                "s3:GetObject",
                "s3:ListBucket",
                "s3:GetBucketPolicy",
                "s3:GetBucketVersioning",
```

```
                "s3:GetObjectVersion",
                "s3:CreateBucket",
                "s3:PutBucketPolicy"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:s3::*:codepipeline-*",
            "Sid": "CodePipelineArtifactsReadWriteAccess"
        },
        {
            "Action": [
                "cloudtrail:PutEventSelectors",
                "cloudtrail:CreateTrail",
                "cloudtrail:GetEventSelectors",
                "cloudtrail:StartLogging"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:cloudtrail:*:*:trail/codepipeline-source-trail",
            "Sid": "CodePipelineSourceTrailReadWriteAccess"
        },
        {
            "Action": [
                "iam:PassRole"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:iam::*:role/service-role/cwe-role-*"
            ],
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": [
                        "events.amazonaws.com"
                    ]
                }
            },
            "Sid": "EventsIAMPassRole"
        },
        {
            "Action": [
                "iam:PassRole"
            ],
            "Effect": "Allow",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
```

```
                    "iam:PassedToService": [
                        "codepipeline.amazonaws.com"
                    ]
                }
            },
            "Sid": "CodePipelineIAMPassRole"
        },
        {
            "Action": [
                "events:PutRule",
                "events:PutTargets",
                "events:DeleteRule",
                "events:DisableRule",
                "events:RemoveTargets"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:events:*:*:rule/codepipeline-*"
            ],
            "Sid": "CodePipelineEventsReadWriteAccess"
        },
        {
            "Sid": "CodeStarNotificationsReadWriteAccess",
            "Effect": "Allow",
            "Action": [
                "codestar-notifications:CreateNotificationRule",
                "codestar-notifications:DescribeNotificationRule",
                "codestar-notifications:UpdateNotificationRule",
                "codestar-notifications:DeleteNotificationRule",
                "codestar-notifications:Subscribe",
                "codestar-notifications:Unsubscribe"
            ],
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "codestar-notifications:NotificationsForResource":
"arn:aws:codepipeline:*"
                }
            }
        },
        {
            "Sid": "CodeStarNotificationsSNSTopicCreateAccess",
            "Effect": "Allow",
            "Action": [
```

```
                "sns:CreateTopic",
                "sns:SetTopicAttributes"
            ],
            "Resource": "arn:aws:sns:*:*:codestar-notifications*"
        },
        {
            "Sid": "CodeStarNotificationsChatbotAccess",
            "Effect": "Allow",
            "Action": [
                "chatbot:DescribeSlackChannelConfigurations",
                "chatbot:ListMicrosoftTeamsChannelConfigurations"
            ],
            "Resource": "*"
        }
    ],
    "Version": "2012-10-17"
}
```

AWS 受管理的策略： AWSCodePipeline_ReadOnlyAccess

這是授與唯讀存取權的原則 CodePipeline。若要在 IAM 主控台中檢視 JSON 政策文件，請參閱AWSCodePipeline_ReadOnlyAccess。

許可詳細資訊

此政策包含以下許可。

- codepipeline— 授與中動作的權限 CodePipeline。
- codestar-notifications— 授與權限以允許主參與者存取 AWS CodeStar 「通知」中的資源。
- s3— 授予許可以允許主體管理 Amazon S3 中的資源。
- sns— 授予許可，以允許主體管理 Amazon SNS 中的通知資源。

```
{
    "Statement": [
        {
            "Action": [
                "codepipeline:GetPipeline",
                "codepipeline:GetPipelineState",
```

```
                "codepipeline:GetPipelineExecution",
                "codepipeline:ListPipelineExecutions",
                "codepipeline:ListActionExecutions",
                "codepipeline:ListActionTypes",
                "codepipeline:ListPipelines",
                "codepipeline:ListTagsForResource",
                "s3:ListAllMyBuckets",
                "codestar-notifications:ListNotificationRules",
                "codestar-notifications:ListEventTypes",
                "codestar-notifications:ListTargets"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Action": [
                "s3:GetObject",
                "s3:ListBucket",
                "s3:GetBucketPolicy"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:s3::*:codepipeline-*"
        },
        {
            "Sid": "CodeStarNotificationsReadOnlyAccess",
            "Effect": "Allow",
            "Action": [
                "codestar-notifications:DescribeNotificationRule"
            ],
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "codestar-notifications:NotificationsForResource":
 "arn:aws:codepipeline:*"
                }
            }
        }
    ],
    "Version": "2012-10-17"
}
```

AWS 受管理的策略：**AWSCodePipelineApproverAccess**

這是授與核准或拒絕手動核准動作之權限的原則。若要在 IAM 主控台中檢視 JSON 政策文件，請參閱 AWSCodePipelineApproverAccess..

許可詳細資訊

此政策包含以下許可。

- codepipeline— 授與中動作的權限 CodePipeline。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "codepipeline:GetPipeline",
                "codepipeline:GetPipelineState",
                "codepipeline:GetPipelineExecution",
                "codepipeline:ListPipelineExecutions",
                "codepipeline:ListPipelines",
                "codepipeline:PutApprovalResult"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

AWS 受管理的策略： AWSCodePipelineCustomActionAccess

這是一項政策，授予在建置或測試動作中建立自訂動作 CodePipeline或整合 Jenkins 資源的權限。若要在 IAM 主控台中檢視 JSON 政策文件，請參閱AWSCodePipelineCustomActionAccess。

許可詳細資訊

此政策包含以下許可。

- codepipeline— 授與中動作的權限 CodePipeline。

```
{
    "Statement": [
        {
            "Action": [
                "codepipeline:AcknowledgeJob",
                "codepipeline:GetJobDetails",
                "codepipeline:PollForJobs",
                "codepipeline:PutJobFailureResult",
                "codepipeline:PutJobSuccessResult"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ],
    "Version": "2012-10-17"
}
```

CodePipeline 受管理的策略和通知

CodePipeline 支援通知，可通知使用者管道的重要變更。 CodePipeline 包含通知功能的政策聲明的受管理策略。如需詳細資訊，請參閱什麼是通知？。

完整存取受管政策中的通知相關許可

此受管理策略會授 CodePipeline 與相關服務 CodeCommit、 CodeBuild CodeDeploy、和 AWS CodeStar 通知的權限。該政策還授予您使用與管道整合的其他服務所需的許可，例如 Amazon S3、Elastic Beanstalk CloudTrail、Amazon EC2 和。 AWS CloudFormation套用此受管政策的使用者也可以針對通知建立和管理 Amazon SNS 主題、訂閱和取消訂閱使用者主題、列出要選擇作為通知規則目標的主題，以及列出針對 Slack 設定的用 AWS Chatbot 戶端。

`AWSCodePipeline_FullAccess` 受管政策包含下列陳述式，允許對通知的完整存取權限。

```
    {
        "Sid": "CodeStarNotificationsReadWriteAccess",
        "Effect": "Allow",
        "Action": [
            "codestar-notifications:CreateNotificationRule",
            "codestar-notifications:DescribeNotificationRule",
            "codestar-notifications:UpdateNotificationRule",
            "codestar-notifications:DeleteNotificationRule",
            "codestar-notifications:Subscribe",
            "codestar-notifications:Unsubscribe"
```

```
        ],
        "Resource": "*",
        "Condition" : {
            "StringLike" : {"codestar-notifications:NotificationsForResource" :
 "arn:aws:codepipeline:us-west-2:111222333444:MyFirstPipeline"}
        }
    },
    {
        "Sid": "CodeStarNotificationsListAccess",
        "Effect": "Allow",
        "Action": [
            "codestar-notifications:ListNotificationRules",
            "codestar-notifications:ListTargets",
            "codestar-notifications:ListTagsforResource",
            "codestar-notifications:ListEventTypes"
        ],
        "Resource": "*"
    },
    {
        "Sid": "CodeStarNotificationsSNSTopicCreateAccess",
        "Effect": "Allow",
        "Action": [
            "sns:CreateTopic",
            "sns:SetTopicAttributes"
        ],
        "Resource": "arn:aws:sns:*:*:codestar-notifications*"
    },
    {
        "Sid": "SNSTopicListAccess",
        "Effect": "Allow",
        "Action": [
            "sns:ListTopics"
        ],
        "Resource": "*"
    },
    {
        "Sid": "CodeStarNotificationsChatbotAccess",
        "Effect": "Allow",
        "Action": [
            "chatbot:DescribeSlackChannelConfigurations",
            "chatbot:ListMicrosoftTeamsChannelConfigurations"
          ],
       "Resource": "*"
```

```
    }
```

唯讀受管政策中的通知相關許可

AWSCodePipeline_ReadOnlyAccess 受管政策包含下列陳述式，允許對通知的唯讀存取權限。套用此政策的使用者可以檢視資源的通知，但無法建立、管理或訂閱通知。

```
  {
       "Sid": "CodeStarNotificationsPowerUserAccess",
       "Effect": "Allow",
       "Action": [
           "codestar-notifications:DescribeNotificationRule"
       ],
       "Resource": "*",
       "Condition" : {
           "StringLike" : {"codestar-notifications:NotificationsForResource" :
"arn:aws:codepipeline:us-west-2:111222333444:MyFirstPipeline"}
       }
   },
   {
       "Sid": "CodeStarNotificationsListAccess",
       "Effect": "Allow",
       "Action": [
           "codestar-notifications:ListNotificationRules",
           "codestar-notifications:ListEventTypes",
           "codestar-notifications:ListTargets"
       ],
       "Resource": "*"
   }
```

如需 IAM 和通知的詳細資訊，請參閱通知的 Identity and Access Management。AWS CodeStar

AWS CodePipelineAWS 受管理策略的更新

檢視 CodePipeline 自此服務開始追蹤這些變更以來的 AWS 受管理策略更新詳細資料。如需自動收到有關此頁面變更的提醒，請前往 CodePipeline 文件歷史記錄頁面上訂閱 RSS 摘要。

變更	描述	日期
AWSCodePipeline_FullAccess— 現有政策的更新	CodePipeline 已新增此原則的權限，以便`ListStacks` 在中支援 AWS CloudFormation。	2024年3月15日
AWSCodePipeline_FullAccess— 現有政策的更新	此原則已更新為新增使用權限 AWS Chatbot。如需詳細資訊，請參閱 CodePipeline 受管理的策略和通知。	2023 年 6 月 21 日
AWSCodePipeline_FullAccess和AWSCodePipeline_ReadOnlyAccess受管理的策略 — 現有策略的更新	CodePipeline 已新增權限至這些原則，以支援使用 AWS Chatbot、的其他通知類型`chatbot:ListMicrosoftTeamsChannelConfigurations` 。	2023 年 5 月 16 日
AWSCodePipelineFullAccess— 已棄用	此政策已被 `AWSCodePipeline_FullAccess` 取代。 2022 年 11 月 17 日之後，此原則無法附加至任何新使用者、群組或角色。如需詳細資訊，請參閱 AWS 受管理的政策 AWS CodePipeline。	2022 年 11 月 17 日
AWSCodePipelineReadOnlyAccess— 已棄用	此政策已被 `AWSCodePipeline_ReadOnlyAccess` 取代。 2022 年 11 月 17 日之後，此原則無法附加至任何新使用者、群組或角色。如需詳細資訊，請參閱 AWS 受管理的政策 AWS CodePipeline。	2022 年 11 月 17 日

變更	描述	日期
CodePipeline 開始追蹤變更	CodePipeline 開始追蹤其 AWS 受管理策略的變更。	2021 年 3 月 12 日

客戶受管政策範例

在本節中，您可以找到授與各種 CodePipeline 動作權限的範例使用者策略。當您使用 CodePipeline API、 AWS SDK 或. AWS CLI當您使用主控台時，您必須對主控台授予特定的其他許可。如需詳細資訊，請參閱 使用 CodePipeline 主控台所需的許可。

> **Note**
>
> 所有範例皆使用美國西部 (奧勒岡) 區域 (us-west-2) 及虛構帳戶 ID。

範例

- 範例 1：授予許可，取得管道的狀態
- 範例 2：授予許可，啟用和停用階段間的轉換
- 範例 3：授予許可，取得所有可用動作類型的清單
- 範例 4：授予許可，核准或拒絕手動核准動作
- 範例 5：授予許可，輪詢自訂動作的任務
- 範例 6：附加或編輯 Jenkins 整合的政策 AWS CodePipeline
- 範例 7：設定管道的跨帳戶存取
- 範例 8：在管道中使用與另一個帳戶建立關聯的 AWS 資源

範例 1：授予許可，取得管道的狀態

下列範例會授予許可，取得名為 MyFirstPipeline 的管道狀態：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
```

```
                "codepipeline:GetPipelineState"
            ],
            "Resource": "arn:aws:codepipeline:us-west-2:111222333444:MyFirstPipeline"
        }
    ]
}
```

範例 2：授予許可，啟用和停用階段間的轉換

下列範例會授予許可，停用和啟用名為 MyFirstPipeline 的管道中所有階段間的轉換：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codepipeline:DisableStageTransition",
                "codepipeline:EnableStageTransition"
            ],
            "Resource": "arn:aws:codepipeline:us-west-2:111222333444:MyFirstPipeline/*"
        }
    ]
}
```

若要允許使用者停用和啟用管道中單一階段的轉換，您必須指定階段。例如，若要允許使用者在名為 MyFirstPipeline 的管道中，對名為 Staging 的階段啟用和停用轉換：

```
"Resource": "arn:aws:codepipeline:us-west-2:111222333444:MyFirstPipeline/Staging"
```

範例 3：授予許可，取得所有可用動作類型的清單

下列範例授予許可，以取得 us-west-2 區域中的管道可用的所有動作類型的清單：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codepipeline:ListActionTypes"
            ],
```

```
            "Resource": "arn:aws:codepipeline:us-west-2:111222333444:actiontype:*"
        }
    ]
}
```

範例 4：授予許可，核准或拒絕手動核准動作

下列範例授予許可，在名為 MyFirstPipeline 的管道中名為 Staging 階段內，核准或拒絕手動核准動作：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codepipeline:PutApprovalResult"
            ],
            "Resource": "arn:aws:codepipeline:us-west-2:111222333444:MyFirstPipeline/
Staging/*"
        }
    ]
}
```

範例 5：授予許可，輪詢自訂動作的任務

下列範例授予許可，以輪詢名為 TestProvider 的自訂動作在所有管道中的任務，而該動作的第一個版本為 Test 動作類型：

> **Note**
>
> 自訂動作的工作背景工作者可能會在不同的 AWS 帳戶下設定，或者需要特定的 IAM 角色才能運作。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
```

```
                "codepipeline:PollForJobs"
            ],
            "Resource": [
                "arn:aws:codepipeline:us-
west-2:111222333444:actionType:Custom/Test/TestProvider/1"
            ]
        }
    ]
}
```

範例 6：附加或編輯 Jenkins 整合的政策 AWS CodePipeline

如果您將管道設定為使用 Jenkins 進行建置或測試，請為該整合建立個別身分，並附加 IAM 政策，該政策具有 Jenkins 和之間整合所需的最低許可。 CodePipeline此政策與 AWSCodePipelineCustomActionAccess 受管政策相同。下面的例子顯示了詹金斯集成的策略：

```
{
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codepipeline:AcknowledgeJob",
                "codepipeline:GetJobDetails",
                "codepipeline:PollForJobs",
                "codepipeline:PutJobFailureResult",
                "codepipeline:PutJobSuccessResult"
            ],
            "Resource": "*"
        }
    ],
    "Version": "2012-10-17"
}
```

範例 7：設定管道的跨帳戶存取

您可以為另一個 AWS 帳戶中的使用者和群組設定管道存取。建議的方法是在建立管道的帳戶中建立角色。該角色應允許來自其他 AWS 帳戶的使用者擔任該角色並存取管道。如需詳細資訊，請參閱演練：使用角色進行跨帳戶存取。

下列範例顯示 80398EXAMPLE 帳戶中的政策，該帳戶允許使用者檢視 (但不能變更) 在主控台MyFirstPipeline中命名的管道。 CodePipeline 此政策以 AWSCodePipeline_ReadOnlyAccess 受管政策為基礎，但因為是 MyFirstPipeline 管道所特

有，因此無法直接使用受管政策。若您不希望將政策限制在特定管道，請考慮使用 CodePipeline 所建立及維護的其中一個受管政策。如需詳細資訊，請參閱處理受管政策的相關文章。您必須將此政策附加到為存取而建立的 IAM 角色，例如名為`CrossAccountPipelineViewers`：

```
{
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codepipeline:GetPipeline",
                "codepipeline:GetPipelineState",
                "codepipeline:ListActionTypes",
                "codepipeline:ListPipelines",
                "iam:ListRoles",
                "s3:GetBucketPolicy",
                "s3:GetObject",
                "s3:ListAllMyBuckets",
                "s3:ListBucket",
                "codedeploy:GetApplication",
                "codedeploy:GetDeploymentGroup",
                "codedeploy:ListApplications",
                "codedeploy:ListDeploymentGroups",
                "elasticbeanstalk:DescribeApplications",
                "elasticbeanstalk:DescribeEnvironments",
                "lambda:GetFunctionConfiguration",
                "lambda:ListFunctions"
            ],
            "Resource": "arn:aws:codepipeline:us-east-2:80398EXAMPLE:MyFirstPipeline"
        }
    ],
    "Version": "2012-10-17"
}
```

建立此政策之後，請在 80398EXAMPLE 帳戶中建立 IAM 角色，並將該政策附加到該角色。在角色的信任關係中，您必須新增擔任此角色的 AWS 帳戶。下列範例顯示的策略可讓 *111111111111* AWS 帳戶的使用者擔任 80398範例帳戶中定義的角色：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
```

```
            "Principal": {
                "AWS": "arn:aws:iam::111111111111:root"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

下列範例顯示在 *111111111111* AWS 帳戶中建立的策略，可讓使用者擔任 80398範例帳戶中指CrossAccountPipelineViewers定的角色：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "sts:AssumeRole",
            "Resource": "arn:aws:iam::80398EXAMPLE:role/CrossAccountPipelineViewers"
        }
    ]
}
```

範例 8：在管道中使用與另一個帳戶建立關聯的 AWS 資源

您可以設定允許使用者建立使用其他 AWS 帳戶資源的管道的策略。在建立管道的帳戶 (AccountA) 和建立資源以用於管道中的帳戶 (AccountB) 中，都需要設定政策和角色。您還必須建立客戶管理的金鑰， AWS Key Management Service 才能用於跨帳戶存取。如需詳細資訊和 step-by-step 範例，請參閱在 CodePipeline 中建立使用來自另一個 AWS 帳戶資源的管道和為 Amazon S3 中存放的成品設定伺服器端加密 CodePipeline。

下列範例針對用於存放管道成品的 S3 儲存貯體，顯示 AccountA 所設定的政策。此政策將存取權授予 AccountB。在下列範例中，AccountB 的 ARN 為 012ID_ACCOUNT_B。S3 儲存貯體的 ARN 為 codepipeline-us-east-2-1234567890。以您要允許存取的 S3 儲存貯體和帳戶的 ARN，取代這些 ARN：

```
{
    "Version": "2012-10-17",
    "Id": "SSEAndSSLPolicy",
    "Statement": [
        {
            "Sid": "DenyUnEncryptedObjectUploads",
            "Effect": "Deny",
```

```
            "Principal": "*",
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::codepipeline-us-east-2-1234567890/*",
            "Condition": {
                "StringNotEquals": {
                    "s3:x-amz-server-side-encryption": "aws:kms"
                }
            }
        },
        {
            "Sid": "DenyInsecureConnections",
            "Effect": "Deny",
            "Principal": "*",
            "Action": "s3:*",
            "Resource": "arn:aws:s3:::codepipeline-us-east-2-1234567890/*",
            "Condition": {
                "Bool": {
                    "aws:SecureTransport": false
                }
            }
        },
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::012ID_ACCOUNT_B:root"
            },
            "Action": [
                "s3:Get*",
                "s3:Put*"
            ],
            "Resource": "arn:aws:s3:::codepipeline-us-east-2-1234567890/*"
        },
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::012ID_ACCOUNT_B:root"
            },
            "Action": "s3:ListBucket",
            "Resource": "arn:aws:s3:::codepipeline-us-east-2-1234567890"
        }
    ]
```

```
}
```

下列範例顯示由 AccountA 設定的政策，允許 AccountB 取得角色。此政策必須套用至 CodePipeline 的服務角色 (CodePipeline_Service_Role)。如需如何在 IAM 中將政策套用至角色的詳細資訊，請參閱修改角色。在下列範例中，012ID_ACCOUNT_B 是 AccountB 的 ARN：

```
{
    "Version": "2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": "sts:AssumeRole",
        "Resource": [
            "arn:aws:iam::012ID_ACCOUNT_B:role/*"
        ]
    }
}
```

下列範例顯示 AccountB 設定並套用至的 EC2 執行個體角色的 CodeDeploy政策。此政策授予對 AccountA 用來存放管道成品 (*codepipeline-us-east-2-* 1234567890) 的 S3 儲存貯體的存取權：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:Get*"
            ],
            "Resource": [
                "arn:aws:s3:::codepipeline-us-east-2-1234567890/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::codepipeline-us-east-2-1234567890"
            ]
        }
```

```
    ]
}
```

下列範例顯示在 Account TA 中建立並設定為 AWS KMS 允許 AccountB 使用之客戶管理金鑰的 ARN 的原則：***arn:aws:kms:us-east-1:012ID_ACCOUNT_A:key/2222222-3333333-4444-556677EXAMPLE***

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:DescribeKey",
                "kms:GenerateDataKey*",
                "kms:Encrypt",
                "kms:ReEncrypt*",
                "kms:Decrypt"
            ],
            "Resource": [
                "arn:aws:kms:us-
east-1:012ID_ACCOUNT_A:key/2222222-3333333-4444-556677EXAMPLE"
            ]
        }
    ]
}
```

下列範例顯示 AccountB 建立的 IAM 角色 (`CrossAccount_Role`) 的內嵌政策，該角色允許存取 AccountA 中管道所需的 CodeDeploy 動作。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codedeploy:CreateDeployment",
                "codedeploy:GetDeployment",
                "codedeploy:GetDeploymentConfig",
                "codedeploy:GetApplicationRevision",
                "codedeploy:RegisterApplicationRevision"
            ],
```

```
            "Resource": "*"
        }
    ]
}
```

下列範例顯示 AccountB 建立的 IAM 角色 (CrossAccount_Role) 的內嵌政策，該角色允許存取 S3 儲存貯體以下載輸入成品和上傳輸出成品：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject*",
                "s3:PutObject",
                "s3:PutObjectAcl"
            ],
            "Resource": [
                "arn:aws:s3:::codepipeline-us-east-2-1234567890/*"
            ]
        }
    ]
}
```

如需如何編輯管道以跨帳戶存取資源的詳細資訊，請參閱步驟 2：編輯管道 。

AWS CodePipeline 資源型政策範例

其他服務 (例如 Amazon S3) 也支援以資源為基礎的許可政策。例如，您可以將政策連接至 S3 儲存貯體，以管理該儲存貯體的存取許可。雖然 CodePipeline 不支援以資源為基礎的政策，但它會在版本控制的 S3 儲存貯體中存放要用於管道的成品。

Example 針對作為 CodePipeline 成品存放區的 S3 儲存貯體建立政策

您可以使用任何版本控制的 S3 儲存貯體做為. CodePipeline 若您使用 Create Pipeline (建立管道) 精靈建立第一個管道，則會為您建立此 S3 儲存貯體，以確保所有上傳至成品存放區的物件都經過加密，且儲存貯體的連線也很安全。若您建立自己的 S3 儲存貯體，根據最佳實務，請考慮將下列政策或其元素新增至儲存貯體。在此政策中，S3 儲存貯體的 ARN 為 codepipeline-us-east-2-1234567890。以您的 S3 儲存貯體的 ARN 取代此 ARN：

```
{
```

```
    "Version": "2012-10-17",
    "Id": "SSEAndSSLPolicy",
    "Statement": [
        {
            "Sid": "DenyUnEncryptedObjectUploads",
            "Effect": "Deny",
            "Principal": "*",
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::codepipeline-us-east-2-1234567890/*",
            "Condition": {
                "StringNotEquals": {
                    "s3:x-amz-server-side-encryption": "aws:kms"
                }
            }
        },
        {
            "Sid": "DenyInsecureConnections",
            "Effect": "Deny",
            "Principal": "*",
            "Action": "s3:*",
            "Resource": "arn:aws:s3:::codepipeline-us-east-2-1234567890/*",
            "Condition": {
                "Bool": {
                    "aws:SecureTransport": false
                }
            }
        }
    ]
}
```

對 AWS CodePipeline 身分與存取進行疑難排解

使用下列資訊可協助您診斷和修正使用和 IAM 時可能會遇到的 CodePipeline 常見問題。

主題

- 我沒有執行操作的授權 CodePipeline
- 我沒有授權執行 iam：PassRole
- 我是系統管理員，想要允許其他人存取 CodePipeline
- 我想允許 AWS 帳戶以外的人員存取我的 CodePipeline資源

我沒有執行操作的授權 CodePipeline

如果 AWS Management Console 告訴您您沒有執行動作的授權，您必須聯絡您的管理員尋求協助。您的管理員是提供您使用者名稱和密碼的人員。

當 mateojackson IAM 使用者嘗試使用主控台檢視管道的詳細資料，但沒有codepipeline:GetPipeline權限時，就會發生下列範例錯誤。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 codepipeline:GetPipeline on resource: my-pipeline
```

在此情況下，Mateo 會請求管理員更新他的政策，允許他使用 my-pipeline 動作存取 codepipeline:GetPipeline 資源。

我沒有授權執行 iam：PassRole

若您收到錯誤，告知您未獲授權執行 iam:PassRole 動作，您必須聯絡管理員以取得協助。您的管理員是提供您使用者名稱和密碼的人員。要求該人員更新您的政策，允許您將角色傳遞給 CodePipeline。

有些 AWS 服務 允許您將現有角色傳遞給該服務，而不是建立新的服務角色或服務連結角色。如需執行此作業，您必須擁有將角色傳遞至該服務的許可。

名為 marymajor 的 IAM 使用者嘗試使用主控台在 CodePipeline 中執行動作時，發生下列範例錯誤。但是，動作要求服務具備服務角色授予的許可。Mary 沒有將角色傳遞至該服務的許可。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

在這種情況下，Mary 會請求管理員更新她的政策，允許她執行 iam:PassRole 動作。

我是系統管理員，想要允許其他人存取 CodePipeline

若要允許其他人存取 CodePipeline，您必須為需要存取的人員或應用程式建立 IAM 實體 (使用者或角色)。他們將使用該實體的憑證來存取 AWS。您接著必須將政策連接到實體，在 CodePipeline 中授予他們正確的許可。

若要立即開始使用，請參閱《IAM 使用者指南》中的建立您的第一個 IAM 委派使用者及群組。

我想允許 AWS 帳戶以外的人員存取我的 CodePipeline資源

您可以建立一個角色，讓其他帳戶中的使用者或您組織外部的人員存取您的資源。您可以指定要允許哪些信任物件取得該角色。針對支援基於資源的政策或存取控制清單 (ACL) 的服務，您可以使用那些政策來授予人員存取您資源的許可。

如需進一步了解，請參閱以下內容：

- 若要瞭解是否 CodePipeline 支援這些功能，請參閱如何與 IAM AWS CodePipeline 搭配使用。
- 若要了解如何提供對您所擁有資源 AWS 帳戶 的存取權，請參閱《IAM 使用者指南》中您擁有的另一 AWS 帳戶 個 IAM 使用者提供存取權限。
- 若要了解如何向第三方提供對資源的存取權 AWS 帳戶，請參閱 IAM 使用者指南中的提供第三方 AWS 帳戶 擁有的存取權。
- 若要了解如何透過聯合身分提供存取權，請參閱《IAM 使用者指南》中的將存取權提供給在外部進行身分驗證的使用者 (聯合身分)。
- 若要了解使用角色和資源型政策進行跨帳戶存取之間的差異，請參閱《IAM 使用者指南》中的 IAM 角色與資源型政策的差異。

CodePipeline 權限參考

當您設定存取控制和撰寫可附加至 IAM 身分 (身分型政策) 的許可政策時，請使用下表作為參考。下表列出每個 CodePipeline API 操作和對應的動作，您可以針對這些項目授予執行動作的許可。對於支援資源層級權限的作業，此表格會列出您可以授與權限的 AWS 資源。您可以在政策的 `Action` 欄位中指定動作。

資源層級權限是指允許您指定允許使用者對哪些資源執行動作的權限。 AWS CodePipeline 提供資源層級權限的部分支援。這表示對於某些 AWS CodePipeline API 呼叫，您可以根據必須符合的條件或允許使用者使用的資源，控制何時允許使用者使用這些動作。例如，您可以授予使用者許可，列出管道執行資訊，但僅限特定管道。

> **Note**
>
> Resources (資源) 欄會列出支援資源層級許可之 API 呼叫所需的資源。對於不支援資源層級許可的 API 呼叫，您可以授與使用者使用該呼叫的許可，但您必須針對政策說明中的資源元素指定萬用字元 (*)。

CodePipeline 動作的 API 作業和必要權限

AcknowledgeJob

動作：codepipeline:AcknowledgeJob

檢視指定任務資訊，以及了解該任務是否已由任務工作者接收的必要許可。僅用於自訂動作。

資源：僅支援政策 Resource 元素中的萬用字元 (*)。

AcknowledgeThirdPartyJob

動作：codepipeline:AcknowledgeThirdPartyJob

確認任務工作者已接收到指定任務的必要許可。僅用於合作夥伴動作。

資源：僅支援政策 Resource 元素中的萬用字元 (*)。

CreateCustomActionType

動作：codepipeline:CreateCustomActionType

需要建立可用於與 AWS 帳戶相關聯的所有管道的新自訂動作。僅用於自訂動作。

資源：

動作類型

arn:aws:codepipeline:*region*:*account*:actiontype:*owner*/*category*/*provider*/*versio*

CreatePipeline

動作：codepipeline:CreatePipeline

建立管道的必要許可。

資源：

管道

arn:aws:codepipeline:*region*:*account*:*pipeline-name*

DeleteCustomActionType

動作：codepipeline:DeleteCustomActionType

將自訂動作標記為「已刪除」的必要許可。當 PollForJobs 的自訂動作標記為刪除後，此動作會失敗。僅用於自訂動作。

資源：

動作類型

arn:aws:codepipeline:*region*:*account*:actiontype:*owner*/*category*/*provider*/*versio*

DeletePipeline

動作：codepipeline:DeletePipeline

刪除管道的必要許可。

資源：

管道

arn:aws:codepipeline:*region*:*account*:*pipeline-name*

DeleteWebhook

動作：codepipeline:DeleteWebhook

刪除 Webhook 的必要許可。

資源：

Webhook

arn:aws:codepipeline:*region*:*account*:webhook:*webhook-name*

DeregisterWebhookWithThirdParty

動作：codepipeline:DeregisterWebhookWithThirdParty

在刪除 webhook 之前，需要刪除由創建的 webhook 與具有要檢測到事件 CodePipeline 的外部工具之間的連接。目前僅針對以動作類型為目標的 Webhook 支援。 GitHub

資源：

Webhook

arn:aws:codepipeline:*region*:*account*:webhook:*webhook-name*

DisableStageTransition

動作：codepipeline:DisableStageTransition

防止管道中成品轉換至管道內下一個階段的必要許可。

資源：

管道

arn:aws:codepipeline:*region*:*account*:*pipeline-name*

EnableStageTransition

動作：codepipeline:EnableStageTransition

允許管道中成品轉換至管道內下一個階段的必要許可。

資源：

管道

arn:aws:codepipeline:*region*:*account*:*pipeline-name*

GetJobDetails

動作：codepipeline:GetJobDetails

擷取任務相關資訊的必要許可。僅用於自訂動作。

資源：不需要資源。

GetPipeline

動作：codepipeline:GetPipeline

擷取管道結構、階段、動作和中繼資料 (包含管道 ARN) 的必要許可。

資源：

管道

arn:aws:codepipeline:*region*:*account*:*pipeline-name*

GetPipelineExecution

動作：codepipeline:GetPipelineExecution

擷取管道執行相關資訊 (包含成品詳細資訊、管道執行 ID、管道名稱、版本、管道狀態) 的必要許可。

資源：

管道

arn:aws:codepipeline:*region*:*account*:*pipeline-name*

GetPipelineState

動作：codepipeline:GetPipelineState

擷取管道狀態相關資訊 (包含階段和動作) 的必要許可。

資源：

管道

arn:aws:codepipeline:*region*:*account*:*pipeline-name*

GetThirdPartyJobDetails

動作：codepipeline:GetThirdPartyJobDetails

請求第三方動作任務詳細資訊的必要許可。僅用於合作夥伴動作。

資源：僅支援政策 Resource 元素中的萬用字元 (*)。

ListActionTypes

動作：codepipeline:ListActionTypes

需要產生與您帳戶相關聯的所有 CodePipeline 動作類型的摘要。

資源：

動作類型

arn:aws:codepipeline:*region*:*account*:actiontype:*owner*/*category*/*provider*/*versio*

ListPipelineExecutions

動作：codepipeline:ListPipelineExecutions

產生最近管道執行摘要的必要許可。

資源：

管道

arn:aws:codepipeline:*region*:*account*:*pipeline-name*

ListPipelines

動作：codepipeline:ListPipelines

必要許可，產生所有與您帳戶建立關聯的管道摘要。

資源：

具有萬用字元的管線 ARN (不支援管線名稱層級的資源層級權限)

arn:aws:codepipeline:*region*:*account*:*

ListTagsForResource

動作：codepipeline:ListTagsForResource

必須列出特定資源的標籤。

資源：

動作類型

arn:aws:codepipeline:*region*:*account*:actiontype:*owner*/*category*/*provider*/*versio*

管道

arn:aws:codepipeline:*region*:*account*:*pipeline-name*

Webhook

arn:aws:codepipeline:*region*:*account*:webhook:*webhook-name*

ListWebhooks

動作：codepipeline:ListWebhooks

必要許可，列出帳戶在該區域中的所有 Webhook。

資源：

Webhook

arn:aws:codepipeline:*region*:*account*:webhook:*webhook-name*

PollForJobs

動作：codepipeline:PollForJobs

擷取要執行處理之任何工作的 CodePipeline 相關資訊所需。

資源：

動作類型

arn:aws:codepipeline:*region*:*account*:actiontype:*owner*/*category*/*provider*/*version*

PollForThirdPartyJobs

動作：codepipeline:PollForThirdPartyJobs

必要許可，判斷是否有任何需要任務工作者處理的第三方任務。僅用於合作夥伴動作。

資源：僅支援政策 Resource 元素中的萬用字元 (*)。

PutActionRevision

動作：codepipeline:PutActionRevision

必須將資訊報 CodePipeline 告給來源的新修訂版本。

資源：

動作

arn:aws:codepipeline:*region*:*account*:*pipeline-name*/*stage-name*/*action-name*

PutApprovalResult

動作：codepipeline:PutApprovalResult

必要許可，向 CodePipeline 報告手動核准請求的回應。有效回應為 Approved 和 Rejected。

資源：

動作

`arn:aws:codepipeline:region:account:pipeline-name/stage-name/action-name`

> **Note**
>
> 此 API 呼叫支援資源層級的許可。但是，若您使用 IAM 主控台或政策產生器，使用指定資源 ARN 的 "codepipeline:PutApprovalResult" 建立政策，便可能發生錯誤。若您發生錯誤，您可以使用 IAM 主控台中的 JSON 標籤或 CLI 來建立政策。

PutJobFailureResult

動作：codepipeline:PutJobFailureResult

將任務工作者傳回管道的任務失敗進行報告的必要許可。僅用於自訂動作。

資源：僅支援政策 Resource 元素中的萬用字元 (*)。

PutJobSuccessResult

動作：codepipeline:PutJobSuccessResult

將任務工作者傳回管道的任務成功進行報告的必要許可。僅用於自訂動作。

資源：僅支援政策 Resource 元素中的萬用字元 (*)。

PutThirdPartyJobFailureResult

動作：codepipeline:PutThirdPartyJobFailureResult

將任務工作者傳回管道的第三方任務失敗進行報告的必要許可。僅用於合作夥伴動作。

資源：僅支援政策 Resource 元素中的萬用字元 (*)。

PutThirdPartyJobSuccessResult

動作：codepipeline:PutThirdPartyJobSuccessResult

將任務工作者傳回管道的第三方任務成功進行報告的必要許可。僅用於合作夥伴動作。

資源：僅支援政策 Resource 元素中的萬用字元 (*)。

PutWebhook

動作：codepipeline:PutWebhook

必須具備才能建立 Webhook。

資源：

Webhook

arn:aws:codepipeline:*region*:*account*:webhook:*webhook-name*

RegisterWebhookWithThirdParty

動作：codepipeline:RegisterWebhookWithThirdParty

資源：

必要許可，在 Webhook 建立之後，設定第三方呼叫產生的 Webhook URL。

Webhook

arn:aws:codepipeline:*region*:*account*:webhook:*webhook-name*

RetryStageExecution

動作：codepipeline:RetryStageExecution

透過重試階段中最後一次失敗動作，繼續執行管道的必要許可。

資源：

管道

arn:aws:codepipeline:*region*:*account*:*pipeline-name*

StartPipelineExecution

動作：codepipeline:StartPipelineExecution

必要許可，啟動指定的管線 (具體而言，就是開始處理管道中指定的來源位置的最新遞交)。

資源：

管道

arn:aws:codepipeline:*region*:*account*:*pipeline-name*

TagResource

動作：codepipeline:TagResource

必要許可，標記指定的資源。

資源：

動作類型

arn:aws:codepipeline:*region*:*account*:actiontype:*owner*/*category*/*provider*/*versio*

管道

arn:aws:codepipeline:*region*:*account*:*pipeline-name*

Webhook

arn:aws:codepipeline:*region*:*account*:webhook:*webhook-name*

UntagResource

動作：codepipeline:UntagResource

必要許可，標記指定的資源。

資源：

動作類型

arn:aws:codepipeline:*region*:*account*:actiontype:*owner*/*category*/*provider*/*versio*

管道

arn:aws:codepipeline:*region*:*account*:*pipeline-name*

Webhook

arn:aws:codepipeline:*region*:*account*:webhook:*webhook-name*

UpdatePipeline

動作：codepipeline:UpdatePipeline

在編輯或變更其結構之後，更新指定管道的必要許可。

資源：

管道

```
arn:aws:codepipeline:region:account:pipeline-name
```

管理服 CodePipeline 務角色

CodePipeline 服務角色配置了一或多個策略，這些策略可控制對管道所使用 AWS 資源的存取。您可能想要將更多策略附加到此角色，編輯附加到該角色的策略，或在中設定其他服務角色的策略 AWS。在設定跨帳戶存取您的管道時，您可能也會想要將政策連接到角色。

> **Important**
>
> 修改政策說明或將其他政策連接到角色，可能會導致您的管道停止運作。 CodePipeline 在以任何方式修改的服務角色之前，請確定您已瞭解其含意。對服務角色進行任何變更之後，務必測試您的管道。

> **Note**
>
> 在主控台，在 2018 年 9 月之前建立的服務角色是以名稱 `oneClick_AWS-CodePipeline-Service_ID-Number` 建立。
> 2018 年 9 月之後建立的服務角色使用服務角色名稱格式 `AWSCodePipelineServiceRole-Region-Pipeline_Name`。例如，對於名為 `MyFirstPipeline` in 的管道`eu-west-2`，控制台會命名角色和策略`AWSCodePipelineServiceRole-eu-west-2-MyFirstPipeline`。

從 CodePipeline 服務角色移除許可

您可以編輯服務角色說明，移除您未使用的資源存取。例如，如果您的管道中沒有包含 Elastic Beanstalk，您可以編輯政策陳述式以移除授與 Elastic Beanstalk 資源存取權的區段。

同樣地，如果您的管道都不包含 CodeDeploy，您可以編輯原則陳述式，移除授與 CodeDeploy 資源存取權的區段：

```
{
"Action": [
    "codedeploy:CreateDeployment",
    "codedeploy:GetApplicationRevision",
    "codedeploy:GetDeployment",
```

```
        "codedeploy:GetDeploymentConfig",
        "codedeploy:RegisterApplicationRevision"
    ],
    "Resource": "*",
    "Effect": "Allow"
},
```

將許可新增至 CodePipeline 服務角色

您必須使用尚 AWS 服務 未包含在預設服務角色原則陳述式中的權限來更新您的服務角色原則陳述式，才能在管線中使用該陳述式。

如果您用於管道的服務角色是在新增支援之前建立的，這一點尤其重要 CodePipeline AWS 服務。

下表顯示為其他新增支援的時間 AWS 服務。

AWS 服務	CodePipeline 支援日期
AWS CloudFormation StackSets 動作	2020 年 12 月 30 日
CodeCommit 完整克隆輸出假影格式	2020 年 11 月 11 日
CodeBuild 批次建置	2020 年 7 月 30 日
AWS AppConfig	2020 年 6 月 22 日
AWS Step Functions	2020 年 5 月 27 日
AWS CodeStar 連接	2019 年 12 月 18 日
`CodeDeployToECS` 動作	2018 年 11 月 27 日
Amazon ECR	2018 年 11 月 27 日
Service Catalog	2018 年 10 月 16 日
AWS Device Farm	2018 年 7 月 19 日
Amazon ECS	2017 年 12 月 12 日/二零一七年七月二十一日選擇加入標籤授權的更新
CodeCommit	2016 年 4 月 18 日

AWS 服務	CodePipeline 支援日期
AWS OpsWorks	2016 年 6 月 2 日
AWS CloudFormation	2016 年 11 月 3 日
AWS CodeBuild	2016 年 12 月 1 日
Elastic Beanstalk	初始服務啟動

請依照下列步驟為支援的服務新增許可：

1. 登入 AWS Management Console 並開啟身分與存取權管理主控台，網址為 https://console.aws.amazon.com/iam/。
2. 在 IAM 主控台的導覽窗格中，選擇 [角色]，然後從AWS-CodePipeline-Service角色清單中選擇您的角色。
3. 在 [權限] 索引標籤的 [內嵌原則] 中，在服務角色原則的列中，選擇 [編輯原則]。
4. 在 [原則] 文件方塊中新增必要的權限。

> **Note**
>
> 建立 IAM 政策時，請遵循授予最低權限的標準安全建議，亦即僅授予執行任務所需的許可。某些 API 呼叫支援資源型許可，並且允許限制存取。例如，在此範例中，如要限制呼叫 `DescribeTasks` 和 `ListTasks` 時的許可，您可以將萬用字元 (*) 取代成資源 ARN，或是使用包含萬用字元 (*) 的資源 ARN。如需建立授與最低權限存取權之原則的詳細資訊，請參閱。https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html#grant-least-privilege

例如，對於 CodeCommit 支持，請將以下內容添加到您的政策聲明中：

```
{
  "Effect": "Allow",
  "Action": [
      "codecommit:GetBranch",
      "codecommit:GetCommit",
      "codecommit:UploadArchive",
```

```
      "codecommit:GetUploadArchiveStatus",
      "codecommit:CancelUploadArchive"
           ],
  "Resource": "resource_ARN"
},
```

如需 AWS OpsWorks 支援，請在您的政策聲明中新增以下內容：

```
{
    "Effect": "Allow",
    "Action": [
        "opsworks:CreateDeployment",
        "opsworks:DescribeApps",
        "opsworks:DescribeCommands",
        "opsworks:DescribeDeployments",
        "opsworks:DescribeInstances",
        "opsworks:DescribeStacks",
        "opsworks:UpdateApp",
        "opsworks:UpdateStack"
    ],
    "Resource": "resource_ARN"
},
```

如需 AWS CloudFormation 支援，請在您的政策聲明中新增以下內容：

```
{
    "Effect": "Allow",
    "Action": [
        "cloudformation:CreateStack",
        "cloudformation:DeleteStack",
        "cloudformation:DescribeStackEvents",
        "cloudformation:DescribeStacks",
        "cloudformation:UpdateStack",
        "cloudformation:CreateChangeSet",
        "cloudformation:DeleteChangeSet",
        "cloudformation:DescribeChangeSet",
        "cloudformation:ExecuteChangeSet",
        "cloudformation:SetStackPolicy",
        "cloudformation:ValidateTemplate",
        "iam:PassRole"
    ],
    "Resource": "resource_ARN"
```

```
},
```

請注意，cloudformation:DescribeStackEvents權限是可選的。它允許動 AWS CloudFormation 作顯示更詳細的錯誤訊息。如果您不想在管道錯誤訊息中顯示資源詳細資訊，則可以從 IAM 角色撤銷此權限。如需詳細資訊，請參閱 AWS CloudFormation。

如需 CodeBuild 支援，請在您的政策聲明中新增以下內容：

```
{
    "Effect": "Allow",
    "Action": [
        "codebuild:BatchGetBuilds",
        "codebuild:StartBuild"
    ],
    "Resource": "resource_ARN"
},
```

> **Note**
>
> 日後已新增對批次建置的 Support 援。如需新增至批次組建服務角色的權限，請參閱步驟 11。

如需 AWS Device Farm 支援，請在您的政策聲明中新增以下內容：

```
{
    "Effect": "Allow",
    "Action": [
        "devicefarm:ListProjects",
        "devicefarm:ListDevicePools",
        "devicefarm:GetRun",
        "devicefarm:GetUpload",
        "devicefarm:CreateUpload",
        "devicefarm:ScheduleRun"
    ],
    "Resource": "resource_ARN"
},
```

如需 Service Catalog 支援，請將下列項目新增至您的原則陳述式：

```
{
    "Effect": "Allow",
    "Action": [
        "servicecatalog:ListProvisioningArtifacts",
        "servicecatalog:CreateProvisioningArtifact",
        "servicecatalog:DescribeProvisioningArtifact",
        "servicecatalog:DeleteProvisioningArtifact",
        "servicecatalog:UpdateProduct"
    ],
    "Resource": "resource_ARN"
},
{
    "Effect": "Allow",
    "Action": [
        "cloudformation:ValidateTemplate"
    ],
    "Resource": "resource_ARN"
}
```

5. 如需 Amazon ECR 支援，請將下列內容新增至您的政策聲明中：

```
{
    "Effect": "Allow",
    "Action": [
        "ecr:DescribeImages"
    ],
    "Resource": "resource_ARN"
},
```

6. 對於 Amazon ECS，以下是使用 Amazon ECS 部署動作建立管道所需的最低許可。

```
{
    "Effect": "Allow",
    "Action": [
        "ecs:DescribeServices",
        "ecs:DescribeTaskDefinition",
        "ecs:DescribeTasks",
        "ecs:ListTasks",
        "ecs:RegisterTaskDefinition",
        "ecs:TagResource",
        "ecs:UpdateService"
    ],
```

```
    "Resource": "resource_ARN"
},
```

您可以在 Amazon ECS 中選擇使用標記授權。選擇加入後，您必須授予以下權限：ecs:TagResource。如需有關如何選擇加入以及判斷是否需要權限以及強制執行標籤授權的詳細資訊，請參閱 Amazon 彈性容器服務開發人員指南中的標記授權時間表。

您還必須新增iam:PassRole許可，才能將 IAM 角色用於工作。如需詳細資訊，請參閱 Amazon ECS 任務執行 IAM 角色和適用於任務的 IAM 角色。請使用下列原則文字。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": [

 "arn:aws:iam::aws_account_ID:role/ecsTaskExecutionRole_or_TaskRole_name"
            ]
        }
    ]
}
```

7. 對於CodeDeployToECS動作 (藍/綠部署)，以下是使用至 Amazon ECS 藍/綠部署動作建立管道所需的最低許可。 CodeDeploy

```
{
    "Effect": "Allow",
    "Action": [
        "codedeploy:CreateDeployment",
        "codedeploy:GetDeployment",
        "codedeploy:GetApplication",
        "codedeploy:GetApplicationRevision",
        "codedeploy:RegisterApplicationRevision",
        "codedeploy:GetDeploymentConfig",
        "ecs:RegisterTaskDefinition",
        "ecs:TagResource"
    ],
    "Resource": "resource_ARN"
},
```

您可以在 Amazon ECS 中選擇使用標記授權。選擇加入後，您必須授予以下權限：ecs:TagResource。如需有關如何選擇加入以及判斷是否需要權限以及強制執行標籤授權的詳細資訊，請參閱 Amazon 彈性容器服務開發人員指南中的標記授權時間表。

您還必須新增iam:PassRole許可，才能將 IAM 角色用於工作。如需詳細資訊，請參閱 Amazon ECS 任務執行 IAM 角色和適用於任務的 IAM 角色。請使用下列原則文字。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": [

 "arn:aws:iam::aws_account_ID:role/ecsTaskExecutionRole_or_TaskRole_name"
            ]
        }
    ]
}
```

您也可以在iam:PassedToService條件下新增ecs-tasks.amazonaws.com至服務清單，如此範例所示。

```
{
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": "resource_ARN",
            "Condition": {
                "StringEqualsIfExists": {
                    "iam:PassedToService": [
                        "cloudformation.amazonaws.com",
                        "elasticbeanstalk.amazonaws.com",
                        "ec2.amazonaws.com",
                        "ecs-tasks.amazonaws.com"
                    ]
                }
```

```
            }
        },
```

8. 對於 AWS CodeStar 連接，需要以下權限才能使用連接的源創建管道，例如 Bitbucket Cloud。

```
{
    "Effect": "Allow",
    "Action": [
        "codestar-connections:UseConnection"
    ],
    "Resource": "resource_ARN"
},
```

如需有關連線的 IAM 許可的詳細資訊，請參閱連線許可參考資料。

9. 對於StepFunctions動作，以下是使用「Step Functions」叫用動作建立管線所需的最低權限。

```
{
    "Effect": "Allow",
    "Action": [
        "states:DescribeStateMachine",
        "states:DescribeExecution",
        "states:StartExecution"
    ],
    "Resource": "resource_ARN"
},
```

10對於此AppConfig動作，以下是使 AWS AppConfig 用 invoke 動作建立管線所需的最低權限。

```
{
    "Effect": "Allow",
    "Action": [
        "appconfig:StartDeployment",
        "appconfig:GetDeployment",
        "appconfig:StopDeployment"
    ],
    "Resource": "resource_ARN"
},
```

11如需批次建置的 CodeBuild 支援，請將下列內容新增至您的政策陳述式：

```
{
    "Effect": "Allow",
```

```
    "Action": [
        "codebuild:BatchGetBuildBatches",
        "codebuild:StartBuildBatch"
    ],
    "Resource": "resource_ARN"
},
```

12對於 AWS CloudFormation StackSets 動作，需要下列最低權限。

- 對於該CloudFormationStackSet操作，請將以下內容添加到您的政策聲明中：

```
{
    "Effect": "Allow",
    "Action": [
        "cloudformation:CreateStackSet",
        "cloudformation:UpdateStackSet",
        "cloudformation:CreateStackInstances",
        "cloudformation:DescribeStackSetOperation",
        "cloudformation:DescribeStackSet",
        "cloudformation:ListStackInstances"
    ],
    "Resource": "resource_ARN"
},
```

- 對於該CloudFormationStackInstances操作，請將以下內容添加到您的政策聲明中：

```
{
    "Effect": "Allow",
    "Action": [
        "cloudformation:CreateStackInstances",
        "cloudformation:DescribeStackSetOperation"
    ],
    "Resource": "resource_ARN"
},
```

13如需完整複製選項的 CodeCommit 支援，請將下列內容新增至您的政策陳述式：

```
{
    "Effect": "Allow",
    "Action": [
        "codecommit:GetRepository"
    ],
    "Resource": "resource_ARN"
```

```
},
```

> **Note**
>
> 為了確保您的 CodeBuild 動作可以將完整複製選項與 CodeCommit 來源搭配使用，您還必須將`codecommit:GitPull`權限新增至專案 CodeBuild 服務角色的政策陳述式。

14對於 Elastic Beanstalk，以下是使用`ElasticBeanstalk`部署動作建立管道所需的最低權限。

```
{
    "Effect": "Allow",
    "Action": [
        "elasticbeanstalk:*",
        "ec2:*",
        "elasticloadbalancing:*",
        "autoscaling:*",
        "cloudwatch:*",
        "s3:*",
        "sns:*",
        "cloudformation:*",
        "rds:*",
        "sqs:*",
        "ecs:*"
    ],
    "Resource": "resource_ARN"
},
```

> **Note**
>
> 您應該將資源策略中的萬用字元取代為您要限制存取的帳號的資源。如需建立授與最低權限存取權之原則的詳細資訊，請參閱。https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html#grant-least-privilege

15對於您要為 CloudWatch 記錄設定的管道，以下是您需要新增至 CodePipeline 服務角色的最低權限。

```
{
    "Effect": "Allow",
    "Action": [
        "logs:DescribeLogGroups",
```

```
            "logs:PutRetentionPolicy"
        ],
        "Resource": "resource_ARN"
},
```

> **Note**
>
> 您應該將資源策略中的萬用字元取代為您要限制存取的帳號的資源。如需建立授與最低權限存取權之原則的詳細資訊，請參閱。https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html#grant-least-privilege

16選擇檢閱政策，確保政策沒有任何錯誤。當原則沒有錯誤時，請選擇 [套用原則]。

登錄和監控 CodePipeline

您可以使用中的記錄功能 AWS 來確定使用者在您的帳戶中執行的動作以及使用的資源。日誌檔顯示：

- 動作的時間和日期。
- 動作的來源 IP 地址。
- 哪些動作因許可不足而失敗。

下列提供記錄功能 AWS 服務：

- AWS CloudTrail 可用 AWS 來記錄由或代表 AWS 帳戶. 如需詳細資訊，請參閱 使用 AWS CloudTrail 記錄 CodePipeline API 呼叫。
- Amazon CloudWatch 事件可用來監控您的 AWS 雲端 資源和執行的應用程式 AWS。您可以根據您定義的指標在 Amazon CloudWatch 事件中建立提醒。如需詳細資訊，請參閱 監控 CodePipeline 事件。

符合性驗證 AWS CodePipeline

若要瞭解 AWS 服務 是否屬於特定規範遵循方案的範圍內，請參閱AWS 服務 遵循規範計劃方案中的，並選擇您感興趣的合規方案。如需一般資訊，請參閱AWS 規範計劃AWS。

您可以使用下載第三方稽核報告 AWS Artifact。如需詳細資訊，請參閱下載中的報告中的 AWS Artifact。

您在使用時的合規責任取決 AWS 服務 於資料的敏感性、公司的合規目標以及適用的法律和法規。AWS 提供下列資源以協助遵循法規：

- 安全性與合規性快速入門指南 — 這些部署指南討論架構考量，並提供部署以安全性和合規性 AWS 為重點的基準環境的步驟。
- 在 Amazon Web Services 上架構 HIPAA 安全性與合規性 — 本白皮書說明公司如何使用建立符合 HIPAA 資格的應 AWS 用程式。

> **Note**
>
> 並非所有人 AWS 服務 都符合 HIPAA 資格。如需詳細資訊，請參閱 HIPAA 資格服務參照。

- AWS 合規資源AWS — 此工作簿和指南集合可能適用於您的產業和所在地。
- AWS 客戶合規指南 — 透過合規的角度瞭解共同的責任模式。這份指南總結了在多個架構 (包括美國國家標準技術研究所 (NIST)、支付卡產業安全標準委員會 (PCI) 和國際標準化組織 (ISO)) 中，保 AWS 服務 護指引並對應至安全控制的最佳實務。
- 使用AWS Config 開發人員指南中的規則評估資源 — 此 AWS Config 服務會評估您的資源組態符合內部實務、產業準則和法規的程度。
- AWS Security Hub— 這 AWS 服務 提供了內部安全狀態的全面視圖 AWS。Security Hub 使用安全控制，可評估您的 AWS 資源並檢查您的法規遵循是否符合安全業界標準和最佳實務。如需支援的服務和控制清單，請參閱 Security Hub controls reference。
- AWS Audit Manager— 這 AWS 服務 有助於您持續稽核您的 AWS 使用情況，以簡化您管理風險的方式，以及遵守法規和業界標準的方式。

韌性 AWS CodePipeline

AWS 全球基礎架構是圍繞區 AWS 域和可用區域建立的。 AWS 區域提供多個實體分離和隔離的可用區域，這些區域透過低延遲、高輸送量和高度備援的網路連線。透過可用區域，您可以設計與操作的應用程式和資料庫，在可用區域之間自動容錯移轉而不會發生中斷。可用區域的可用性、容錯能力和擴充能力，均較單一或多個資料中心的傳統基礎設施還高。

如需區域和可用區域的相關 AWS 資訊，請參閱AWS 全域基礎結構。

基礎結構安全 AWS CodePipeline

作為託管服務， AWS CodePipeline 受到 AWS 全球網絡安全的保護。有關 AWS 安全服務以及如何 AWS 保護基礎架構的詳細資訊，請參閱AWS 雲端安全 若要使用基礎架構安全性的最佳做法來設計您的 AWS 環境，請參閱安全性支柱架構良 AWS 好的架構中的基礎結構保護。

您可以使用 AWS 已發佈的 API 呼叫透 CodePipeline 過網路存取。用戶端必須支援下列項目：

- Transport Layer Security (TLS)。我們需要 TLS 1.2 並建議使用 TLS 1.3。
- 具備完美轉送私密(PFS)的密碼套件，例如 DHE (Ephemeral Diffie-Hellman)或 ECDHE (Elliptic Curve Ephemeral Diffie-Hellman)。現代系統(如 Java 7 和更新版本)大多會支援這些模式。

此外，請求必須使用存取索引鍵 ID 和與 IAM 主體相關聯的私密存取索引鍵來簽署。或者，您可以透過 AWS Security Token Service (AWS STS) 來產生暫時安全憑證來簽署請求。

安全最佳實務

CodePipeline 在您開發和實作自己的安全性原則時，提供許多安全性功能供您考量。以下最佳實務為一般準則，並不代表完整的安全解決方案。這些最佳實務可能不適用或無法滿足您的環境需求，因此請將其視為實用建議就好，而不要當作是指示。

您可以針對連線至管道的來源儲存庫使用加密和身分驗證。以下是安全性的 CodePipeline 最佳做法：

- 如果您建立的管線或動作組態需要包含密碼 (例如 Token 或密碼)，請勿直接在動作組態中輸入密碼，或在管線層級或組 AWS CloudFormation 態中定義的變數預設值，因為資訊會顯示在記錄檔中。使用 Secrets Manager 來設定和儲存密碼，然後在管線和動作組態中使用參照的密碼，如中所述 用 AWS Secrets Manager 於追蹤資料庫密碼或第三方 API 金鑰。
- 如果您建立使用 S3 來源儲存貯體的管道，請 CodePipeline 按照中所述，透過管理 AWS KMS keys 方式為 Amazon S3 中存放的成品設定伺服器端加密為 Amazon S3 中存放的成品設定伺服器端加密 CodePipeline。
- 如果您使用 Jenkins 建置提供者，當您針對管道的建置或測試動作使用 Jenkins 建置提供者時，請在 EC2 執行個體上安裝 Jenkins，並設定個別 EC2 執行個體描述檔。確保執行個體設定檔僅授予 Jenkins 執行專案任務所需的 AWS 許可，例如從 Amazon S3 擷取檔案。若要了解如何針對 Jenkins 執行個體描述檔建立該角色，請參閱建立 IAM 角色以用於詹金斯整合中的步驟。

AWSCodePipeline 參考

執行 AWS CodePipeline 命令時請使用此參考，並做為 AWS CLI 使用者指南與 AWS CLI 參考中記錄的資訊之補充資料。

在使用 AWS CLI 前，請確認您已完成 開始使用 CodePipeline 中的事前準備。

若要查看所有可用的 CodePipeline 命令的清單，請執行下列命令：

```
aws codepipeline help
```

若要查看特定 CodePipeline 命令的相關資訊，請執行下列命令，其中*###*是下列所列之其中一項命令的名稱（例如create-pipeline）：

```
aws codepipeline command-name help
```

若要開始學習如何使用 CodePipeline 擴充功能的AWS CLI下，請前往下列其中一或多個部分：

- 建立自訂動作
- 建立管道 (CLI)
- 刪除管道 (CLI)
- 停用或啟用轉換 (CLI)
- 檢視管道詳細資訊與歷程記錄 (CLI)
- 重試失敗的動作 (CLI)
- 手動啟動管道 (CLI)
- 編輯管道 (AWS CLI)

您也可以在 CodePipeline 教程 中，檢視如何使用大部分命令的範例。

CodePipeline 配管結構參照

根據預設，任何您於 AWS CodePipeline 成功建立的管道都具有有效結構。但是，若您手動建立或編輯 JSON 檔案來從 AWS CLI 建立管道更新管道，您必須建立不具效力的結構。下列參考可協助您更進一步了解管道結構的要求，以及如何對問題進行故障排除。請參閱 AWS CodePipeline 中的配額 中的限制，這些限制適用於所有管道。

主題

中的有效動作類型和提供者 CodePipeline

管道結構格式是用於建置管道中的動作和階段。動作類型包含動作類別和供應商類型。

以下是 CodePipeline 中有效的動作類別：

- 來源
- 組建
- 測試
- 部署
- 核准
- Invoke

每個動作類別都有指定的一組供應商。每個動作提供者 (例如 Amazon S3) 都有一個提供者名稱`S3`，例如，必須在管道結構中動作類別的`Provider`欄位中使用。

管道結構的動作類別部分 `Owner` 欄位有三個有效的值：`AWS`、`ThirdParty` 和 `Custom`。

若要尋找提供者名稱和動作提供者的擁有者資訊，請參閱 動作結構參考 或 每個動作類型的輸入和輸出成品數目。

下表依動作類型列出有效的供應商。

> **Note**
>
> 如需 Bitbucket 雲端 GitHub、 GitHub 企業伺服器或 GitLab .com 動作，請參閱CodeStarSourceConnection 適用於 Bitbucket 雲端 GitHub、 GitHub 企業伺服器、GitLab .com 和 GitLab 自我管理動作動作參考主題。

依動作類型的有效動作提供者

動作類別	有效動作供應商	動作參考
來源	Amazon S3	Amazon S3 來源動作
	Amazon ECR	Amazon ECR
	CodeCommit	CodeCommit
	CodeStarSourceConnection (適用於 Bitbucket 雲端 GitHub、 GitHub 企業伺服器或 GitLab .com 動作)	CodeStarSourceConnection 適用於 Bitbucket 雲端 GitHub、 GitHub 企業伺服器、 GitLab .com 和 GitLab 自我管理動作
組建	CodeBuild	AWS CodeBuild
	自訂 CloudBees	每個動作類型的輸入和輸出成品數目
	自訂 Jenkins	每個動作類型的輸入和輸出成品數目

動作類別	有效動作供應商	動作參考
	自訂 TeamCity	每個動作類型的輸入和輸出成品數目
測試	CodeBuild	AWS CodeBuild
	AWS Device Farm	每個動作類型的輸入和輸出成品數目
	ThirdParty GhostInspector	每個動作類型的輸入和輸出成品數目
	自訂 Jenkins	每個動作類型的輸入和輸出成品數目
	ThirdParty 微聚焦 StormRunner 負載	每個動作類型的輸入和輸出成品數目
	ThirdParty 努沃拉	每個動作類型的輸入和輸出成品數目
部署	Amazon S3	Amazon S3 動作
	AWS CloudFormation	AWS CloudFormation
	AWS CloudFormation StackSets (包括`CloudFormationStackSet` 和`CloudFormationStackInstances` 動作)	AWS CloudFormation StackSets

動作類別	有效動作供應商	動作參考
	CodeDeploy	每個動作類型的輸入和輸出成品數目
	Amazon ECS	每個動作類型的輸入和輸出成品數目
	Amazon ECS (Blue/Green) (這是 `CodeDeployToECS` 動作)	每個動作類型的輸入和輸出成品數目
	Elastic Beanstalk	每個動作類型的輸入和輸出成品數目
	AWS AppConfig	AWSAppConfig
	AWS OpsWorks	每個動作類型的輸入和輸出成品數目
	Service Catalog	每個動作類型的輸入和輸出成品數目
	Amazon Alexa	每個動作類型的輸入和輸出成品數目
	自訂 XebiaLabs	每個動作類型的輸入和輸出成品數目
核准	手動	每個動作類型的輸入和輸出成品數目

動作類別	有效動作供應商	動作參考
Invoke	AWS Lambda	AWS Lambda
	AWS Step Functions	AWS Step Functions

中的某些動作 CodePipeline 類型僅適用於特定AWS區域。「AWS區域」中可能有動作類型可用，但該動作類型的AWS提供者無法使用。

如需每個動作供應商的詳細資訊，請參閱與 CodePipeline 動作類型的整合。

以下章節針對每個動作類型的供應商資訊和組態屬性提供範例。

CodePipeline 中的管道與階段結構要求

兩階段管道擁有下列基本結構：

```
{
    "roleArn": "An IAM ARN for a service role, such as arn:aws:iam::80398EXAMPLE:role/
CodePipeline_Service_Role",
    "stages": [
        {
            "name": "SourceStageName",
            "actions": [
            ... See ######  CodePipeline ...
            ]
        },
        {
            "name": "NextStageName",
            "actions": [
            ... See ######  CodePipeline ...
            ]
        }
    ],
    "artifactStore": {
        "type": "S3",
        "location": "The name of the Amazon S3 bucket automatically generated for you
 the first time you create a pipeline
            using the console, such as codepipeline-us-east-2-1234567890, or any Amazon
 S3 bucket you provision for this purpose"
```

```
    },
    "name": "YourPipelineName",
    "version": 1
}
```

管道結構具有下列要求：

- 管道必須包含至少兩個階段。
- 管道的第一階段必須包含至少一項來源動作。它只能包含來源動作。
- 只有管道的第一個階段可包含來源動作。
- 各管道至少要有一個階段，包含不是來源動作的動作。
- 管道中的所有階段名稱必須是唯一的。
- 階段名稱無法在 CodePipeline 主控台中編輯。若您使用 AWS CLI 來編輯階段名稱，而該階段包含一個含有一項或多項秘密參數 (例如 OAuth 符記) 的動作，那些秘密參數的值不會保留。您必須手動輸入參數的值 (在 AWS CLI 傳回的 JSON 中以四個星號遮蓋)，並將這些值包含在 JSON 結構中。
- 此artifactStore欄位包含管道的人工因素值區類型和位置，其中包含在相同AWS區域中的所有動作。如果您在與管道不同的「區域」中新增動作，則會使用artifactStores對應來列出每個執行動作之「AWS區域」的成品值區。當您建立或編輯管道時，您必須在管道區域中擁有一個成品儲存貯體，然後對於每個您計劃執行動作的區域，都必須擁有一個成品儲存貯體。

  以下範例顯示管道的基本結構，具有使用 artifactStores 參數的跨區域動作：

```
    "pipeline": {
        "name": "YourPipelineName",
        "roleArn": "CodePipeline_Service_Role",
        "artifactStores": {
            "us-east-1": {
                "type": "S3",
                "location": "S3 artifact bucket name, such as codepipeline-us-
east-1-1234567890"
            },
            "us-west-2": {
                "type": "S3",
                "location": "S3 artifact bucket name, such as codepipeline-us-
west-2-1234567890"
            }
        },
        "stages": [
            {
```

```
...
```

- 管道中繼資料欄位與管道結構不同，且無法編輯。當您更新管道時，updated 中繼資料欄位中的日期將會自動更改。
- 當您編輯或更新管道時，管道名稱無法更改。

> **Note**
>
> 若您想要重新命名現有管道，可以使用 CLI `get-pipeline` 命令來建置包含您管道結構的 JSON 檔案。您可以接著使用 CLI `create-pipeline` 命令來建立含有該結構的管道，並賦予它新名稱。

管道的版本編號將會自動產生，並在每次您更新管道時更新。

動作結構需求 CodePipeline

動作具有下列高階結構：

```
[
                {
                   "inputArtifacts": [
                        An input artifact structure, if supported for the action
 category
                    ],
                   "name": "ActionName",
                   "region": "Region",
                   "namespace": "source_namespace",
                   "actionTypeId": {
                        "category": "An action category",
                        "owner": "AWS",
                        "version": "1"
                        "provider": "A provider type for the action category",
                   },
                   "outputArtifacts": [
                        An output artifact structure, if supported for the action
 category
                   ],
                   "configuration": {
                        Configuration details appropriate to the provider type
```

```
                },
                "runOrder": A positive integer that indicates the run order within
the stage,
            }
        ]
```

如需適用於供應商類型的範例 configuration 詳細資訊，請參閱 依提供者類型的組態詳細資訊。

動作結構具有下列要求：

- 階段內的所有動作名稱必須是唯一的。
- 動作的輸入加工品必須完全符合上一個動作中宣告的輸出成品。例如，若前述動作包含下列宣告：

  ```
  "outputArtifacts": [
      {
      "MyApp"
      }
  ],
  ```

  且沒有其他輸出成品，那麼下列動作的輸入成品必須為：

  ```
  "inputArtifacts": [
      {
      "MyApp"
      }
  ],
  ```

  這對所有動作皆適用，無論是否處於相同階段或者處於下列階段中，但是輸入成品無需嚴格排列在提供了輸出成品的動作之後。平行動作可以宣告不同的輸出成品套件，由下列的不同動作輪流消耗使用。
- 管道中的輸出成品名稱皆必須獨一無二。例如，管道可包含一個含有名為 "MyApp" 的輸出成品之動作，而另一個動作則有名為 "MyBuiltApp" 的輸出成品。不過，管道不可包含兩個動作，而兩個動作都有名為 "MyApp" 的輸出成品。
- 跨區域作業使用Region欄位來指定要在其中建立作業的「AWS區域」。為此動作建立的AWS資源必須在欄位中提供的相同「區域region」中建立。您無法針對以下動作類型建立跨區域動作：
  - 來源動作
  - 依第三方供應商的動作
  - 依自訂供應商的動作

- 動作可以使用變數來設定。您可以使用 namespace 欄位來設定執行變數的命名空間和變數資訊。如需執行變數和動作輸出變數的參考資訊，請參閱Variables。
- 對於目前支援的所有動作類型，唯一有效的擁有者字串為AWSThirdParty、或Custom。如需詳細資訊，請參閱 CodePipeline API 參考。
- 動作的預設 runOrder 值為 1。值必須是正整數 (自然數)。您不可使用分數、小數、負數或零。若要指定動作的編號順序，請使用第一個動作的最小編號以及順序中各個其他動作的較大編號。若要指定平行動作，請為您想要平行執行的各項動作使用相同整數。在主控台中，您可以在要執行動作的階段層級選擇「新增動作群組」來指定動作的序列順序，或者選擇「新增」動作來指定 parallel 序列。動作群組是指相同層級中一個或多個動作的執行順序。

  例如，若您想要三項動作在階段中連續執行，需給予第一項動作 runOrder 值 1，第二個動作的 runOrder value 值為 2、第三個動作的 runOrder 值則為 3。但是，若您想要平行執行第二或第三項動作，必須給予第一項動作 runOrder 值 1，而第二與第三項動作的 runOrder 值則為 2。

  > **Note**
  >
  > 序列動作的編號不需為嚴格連續順序。例如，若您有連續三個動作並決定要移除第二項動作，無需重新為第三項動作的 runOrder 值編號。因為該動作的 runOrder 值 (3) 高於第一項動作的 runOrder 值 (1)，會在該階段中的第一項動作之後根據序號執行。

- 當您使用 Amazon S3 儲存貯體做為部署位置時，您也可以指定物件金鑰。物件金鑰可以是檔案名稱 (物件) 或字首 (資料夾路徑) 和檔案名稱的組合。您可以使用變數指定您想要管道使用的位置名稱。Amazon S3 部署動作支援在 Amazon S3 物件金鑰中使用下列變數。

  在 Amazon S3 中使用變量

變數	主控台輸入的範例	輸出
datetime	js-application/{datetime}.zip	此格式的 UTC 時間戳記：<YYYY>-<MM>-DD>_<HH>-<MM>-<SS> 範例： js-application/2019-01-10_07-39-57.zip
uuid	js-application/{uuid}.zip	UUID 是全域唯一識別符，保證不同於任何其他識別符。此格式的 UUID (所有數字皆採用十六進位格式)：<8 位數>-<4 位數>-4 位數>-<4 位數>-<12 位數>

變數	主控台輸入的範例	輸出
		範例： js-application/54a60075-b96a-4bf3-9013-db3a9EXAMPLE.zip

- 這些是下列項目的有效`actionTypeId`類別 CodePipeline：
  - `Source`
  - `Build`
  - `Approval`
  - `Deploy`
  - `Test`
  - `Invoke`

  這裡提供部分提供者類型與組態選項。
- 動作類別的有效提供者類型根據類別而定。例如，對於來源動作類別來說，有效的提供者類型為 S3、GitHub、CodeCommit 或 Amazon ECR。此範例顯示來源動作的結構，具有 S3 供應商：

```
"actionTypeId": {
  "category": "Source",
  "owner": "AWS",
  "version": "1",
  "provider": "S3"},
```

- 每項動作都必須具備有效的動作組態，根據該動作的提供者類型而定。下表列出各個有效提供者類型所需的動作組態元素：

  提供者類型的動作組態屬性

提供者名稱	動作類型中的提供者名稱	組態屬性	必要的屬性？
Amazon S3（部署動作提供商）	如需詳細資訊，包括與 Amazon S3 部署動作參數相關的範例，請參閱Amazon S3 動作。		

提供者名稱	動作類型中的提供者名稱	組態屬性	必要的屬性？
Amazon S3（源操作提供商）	如需詳細資訊，包括與 Amazon S3 來源動作參數相關的範例，請參閱Amazon S3 來源動作。		
Amazon ECR	如需詳細資訊，包括與 Amazon ECR 參數相關的範例，請參閱Amazon ECR。		
CodeCommit	如需詳細資訊，包括與參數相關的範例，請 CodeCommit 參閱CodeCommit。		
GitHub	如需詳細資訊，包括與參數相關的範例，請 GitHub 參閱GitHub 版本 1 來源動作結構參考。		
AWS CloudFormation	如需詳細資訊，包括與 AWS CloudFormation 參數相關的範例，請參閱 AWS CloudFormation。		
CodeBuild	如需參數相關的詳細描述和範例，請 CodeBuild 參閱AWS CodeBuild。		
CodeDeploy	如需參數相關的詳細描述和範例，請 CodeDeploy 參閱AWS CodeDeploy。		
AWS Device Farm	如需 AWS Device Farm 參數相關的詳細說明和範例，請參閱 AWS Device Farm。		
AWS Elastic Beanstalk	`ElasticBeanstalk`	`ApplicationName`	必要
		`EnvironmentName`	必要
AWS Lambda	如需詳細資訊，包括與 AWS Lambda 參數相關的範例，請參閱 AWS Lambda。		
AWS OpsWorks Stacks	`OpsWorks`	`Stack`	必要
		`Layer`	選用

提供者名稱	動作類型中的提供者名稱	組態屬性	必要的屬性？
		App	必要
Amazon ECS	如需與 Amazon ECS 參數相關的詳細說明和範例，請參閱Amazon Elastic Container Service。		
亞馬遜 ECS 和 CodeDeploy（藍色/綠色）	如需 Amazon ECS 和 CodeDeploy 藍/綠參數相關的詳細說明和範例，請參閱。Amazon Elastic Coner Coner Coner Coner Coner ConCodeDeploy		
Service Catalog	`ServiceCatalog`	`TemplateFilePath`	必要
		`ProductVersionName`	必要
		`ProductType`	必要
		`ProductVersionDescription`	選用
		`ProductId`	必要
Alexa Skills Kit	`AlexaSkillsKit`	`ClientId`	必要
		`ClientSecret`	必要
		`RefreshToken`	必要
		`SkillId`	必要
Jenkins	您在 Jenkins 的 CodePipeline 外掛程式中提供的動作名稱 (例如，*MyJenkins ProviderName*)	`ProjectName`	必要
手動核准	`Manual`	`CustomData`	選用

提供者名稱	動作類型中的提供者名稱	組態屬性	必要的屬性？
		ExternalEntityLink	選用
		NotificationArn	選用

主題

- 每個動作類型的輸入和輸出成品數目
- PollForSourceChanges 參數的預設設定
- 依提供者類型的組態詳細資訊

每個動作類型的輸入和輸出成品數目

根據動作類型，您可以使用下列的輸入與輸出成品編號：

動作類型對成品的限制

Owner	動作類型	供應商	輸入成品的有效編號	輸出成品的有效編號
AWS	來源	Amazon S3	0	1
AWS	來源	CodeCommit	0	1
AWS	來源	Amazon ECR	0	1
ThirdParty	來源	GitHub	0	1
AWS	組建	CodeBuild	1 到 5	0 到 5
AWS	測試	CodeBuild	1 到 5	0 到 5
AWS	測試	AWS Device Farm	1	0
AWS	核准	手動	0	0

Owner	動作類型	供應商	輸入成品的有效編號	輸出成品的有效編號
AWS	部署	Amazon S3	1	0
AWS	部署	AWS CloudFormation	0 到 10	0 到 1
AWS	部署	CodeDeploy	1	0
AWS	部署	AWS Elastic Beanstalk	1	0
AWS	部署	AWS OpsWorks Stacks	1	0
AWS	部署	Amazon ECS	1	0
AWS	部署	Service Catalog	1	0
AWS	Invoke	AWS Lambda	0 到 5	0 到 5
ThirdParty	部署	Alexa Skills Kit	1 到 2	0
Custom	組建	Jenkins	0 到 5	0 到 5
Custom	測試	Jenkins	0 到 5	0 到 5
Custom	任何支援的類別	如自訂動作中指定	0 到 5	0 到 5

PollForSourceChanges 參數的預設設定

PollForSourceChanges 參數預設值取決於用來建立管道的方法，如下表中所述。在許多情況下，PollForSourceChanges 參數會預設為 True，而且必須停用。

當 PollForSourceChanges 參數預設為 true 時，您應該執行下列動作：

- 將 PollForSourceChanges 參數新增至 JSON 檔案或 AWS CloudFormation 範本。
- 建立變更偵測資源 (CloudWatch 事件規則，如適用)。

- 將 PollForSourceChanges 參數設為 false。

> **Note**
>
> 如果您建立 CloudWatch 事件規則或 webhook，則必須將參數設定為 false，以避免多次觸發管線。

此PollForSourceChanges參數不適用於 Amazon ECR 來源動作。

- PollForSourceChanges 參數預設值

來源	建立方法	「組態」JSON 結構輸出範例
CodeCommit	管道是透過主控台建立 (而變更偵測資源是由主控台建立)。該參數會顯示在管道結構輸出中且預設為 false。	`BranchName": "main", "PollForSourceChanges": "false", "RepositoryName": "my-repo"`
	管道是使用 CLI 或建立的AWS CloudFormation，且PollForSourceChanges 參數不會顯示在 JSON 輸出中，但會設定為 true .[2]	`BranchName": "main", "RepositoryName": "my-repo"`
Amazon S3	管道是透過主控台建立 (而變更偵測資源是由主控台建立)。該參數會顯示在管道結構輸出中且預設為 false。	`"S3Bucket": "my-bucket", "S3ObjectKey": "object.zip", "PollForSourceChanges": "false"`
	管道是使用 CLI 或建立的AWS CloudFormation，且PollForSourceChanges 參數不會顯示在 JSON 輸出中，但會設定為 true .[2]	`"S3Bucket": "my-bucket", "S3ObjectKey": "object.zip"`

來源	建立方法	「組態」JSON 結構輸出範例
GitHub	管道是透過主控台建立 (而變更偵測資源是由主控台建立)。該參數會顯示在管道結構輸出中且預設為 `false`。	`"Owner": "MyGitHubAccountName ", "Repo": " MyGitHubRepositoryName " "PollForSourceChanges": "false", "Branch": " main" "OAuthToken": " ****"`
	管道是使用 CLI 或建立的AWS CloudFormation，且`PollForSourceChanges` 參數不會顯示在 JSON 輸出中，但會設定為 `true`.[2]	`"Owner": "MyGitHubAccountName ", "Repo": "MyGitHubRepositoryName ", "Branch": " main", "OAuthToken": " ****"`
	[2] 如果`PollForSourceChanges` 已在任何時候添加到 JSON 結構或AWS CloudFormation模板中，則顯示如下： `"PollForSourceChanges": "true",` [3] 如需適用於每個來源提供者之變更偵測資源的相關資訊，請參閱變更偵測方法。	

依提供者類型的組態詳細資訊

本部分列出每個動作供應商的有效 `configuration` 參數。

下列範例顯示使用 Service Catalog 之部署動作的有效組態，適用於在主控台中建立的管線，而不需要個別組態檔案：

```
"configuration": {
  "TemplateFilePath": "S3_template.json",
  "ProductVersionName": "devops S3 v2",
  "ProductType": "CLOUD_FORMATION_TEMPLATE",
  "ProductVersionDescription": "Product version description",
```

```
  "ProductId": "prod-example123456"
}
```

下列範例顯示使用 Service Catalog 之部署動作的有效組態，適用於在主控台中使用個別`sample_config.json`組態檔建立的管線：

```
"configuration": {
  "ConfigurationFilePath": "sample_config.json",
  "ProductId": "prod-example123456"
}
```

以下範例顯示使用 Alexa Skills Kit 之部署動作的有效組態：

```
"configuration": {
  "ClientId": "amzn1.application-oa2-client.aadEXAMPLE",
  "ClientSecret": "****",
  "RefreshToken": "****",
  "SkillId": "amzn1.ask.skill.22649d8f-0451-4b4b-9ed9-bfb6cEXAMPLE"
}
```

以下範例顯示手動核准的有效組態：

```
"configuration": {
  "CustomData": "Comments on the manual approval",
  "ExternalEntityLink": "http://my-url.com",
  "NotificationArn": "arn:aws:sns:us-west-2:12345EXAMPLE:Notification"
}
```

動作結構參考

本節僅供動作組態參考。如需管道結構的概念介紹，請參閱 CodePipeline 配管結構參照。

中的每個動作提供者都 CodePipeline 會在配管結構中使用一組必要和選用的組態欄位。本節依動作提供者提供下列參考資訊：

- 管道結構動作區塊中包含 `ActionType` 欄位的有效值，例如 `Owner` 和 `Provider`。
- 管道結構動作區段中包含 `Configuration` 參數的說明和其他參考資訊 (必要和選用)。
- 有效的 JSON 範例和 YAML 動作欄位。

本節會定期更新，包含更多的動作提供者。目前提供下列動作提供者的參考資訊：

主題

- Amazon ECR
- Amazon Elastic Coner Coner Coner Coner Coner ConCodeDeploy
- Amazon Elastic Container Service
- Amazon S3 動作
- Amazon S3 來源動作
- AWSAppConfig
- AWS CloudFormation
- AWS CloudFormation StackSets
- AWS CodeBuild
- CodeCommit
- AWS CodeDeploy
- CodeStarSourceConnection 適用於 Bitbucket 雲端 GitHub、 GitHub 企業伺服器、 GitLab .com 和 GitLab 自我管理動作
- AWS Device Farm
- AWS Lambda
- Snyk 動作結構參考
- AWS Step Functions

Amazon ECR

當新映像推送至 Amazon ECR 儲存庫時觸發管道。此動作提供映像定義檔，以參考推送至 Amazon ECR 的映像的 URI。此來源動作通常與另一個來源動作搭配使用，例如 CodeCommit，以提供所有其他來源成品的來源位置。如需詳細資訊，請參閱 教學課程：使用 Amazon ECR 來源和 ECS 到部署建立管道 CodeDeploy 。

當您使用主控台建立或編輯管道時， CodePipeline 會建立 CloudWatch 當儲存庫發生變更時啟動管道的事件規則。

您必須先建立 Amazon ECR 儲存庫並推送映像，才能透過 Amazon ECR 動作連接管道。

主題

動作類型

- 類別：`Source`
- 擁有者：`AWS`
- 提供者：`ECR`
- 版本：`1`

組態參數

RepositoryName

：必要 是

推入映像的 Amazon ECR 儲存庫的名稱。

ImageTag

：必要 否

用於映像的標籤。

> **Note**
>
> 如果未指定 ImageTag 的數值，則預設值為 latest。

Input artifacts (輸入成品)

- 成品數量： 0
- 描述：輸入成品不適用於此動作類型。

輸出成品

- 成品數量： 1
- 描述：此操作會生成一個工件，其中包含imageDetail.json檔案，而此檔案包含觸發管道執行之映像的 URI。如需 imageDetail.json 詳細資訊，請參閱 適用於 Amazon ECS 藍色/綠色部署動作的 imageDetail.json 檔案。

輸出變數

設定時，此動作會產生變數，供管道中的下游動作的動作組態所參考。即使此動作沒有命名空間，此動作產生的變數仍可視為輸出變數。您可以設定動作的命名空間，讓這些變數可供下游動作的組態使用。

如需詳細資訊，請參閱 Variables。

RegistryId

所以此AWS帳户 ID 與內含儲存庫之登錄檔建立關聯。

RepositoryName

推入映像的 Amazon ECR 儲存庫的名稱。

ImageTag

用於映像的標籤。

ImageDigest

映像資訊清單的 sha256 摘要。

ImageURI

映像的 URI。

動作宣告 (亞馬遜 ECR 範例)

YAML

```
Name: Source
Actions:
  - InputArtifacts: []
    ActionTypeId:
      Version: '1'
      Owner: AWS
      Category: Source
      Provider: ECR
    OutputArtifacts:
      - Name: SourceArtifact
    RunOrder: 1
    Configuration:
      ImageTag: latest
      RepositoryName: my-image-repo

    Name: ImageSource
```

JSON

```
{
    "Name": "Source",
    "Actions": [
        {
            "InputArtifacts": [],
            "ActionTypeId": {
                "Version": "1",
                "Owner": "AWS",
                "Category": "Source",
                "Provider": "ECR"
            },
```

```
            "OutputArtifacts": [
                {
                    "Name": "SourceArtifact"
                }
            ],
            "RunOrder": 1,
            "Configuration": {
                "ImageTag": "latest",
                "RepositoryName": "my-image-repo"
            },
            "Name": "ImageSource"
        }
    ]
},
```

另請參閱

以下相關資源可協助您使用此動作。

- 教學課程：使用 Amazon ECR 來源和 ECS 到部署建立管道 CodeDeploy — 本教學課程提供範例應用程式規格檔案及範例 CodeDeploy 應用程序和部署組創建具有 CodeCommit 和部署到亞馬遜雲服務器實例的 Amazon ECR 源。

Amazon Elastic Coner Coner Coner Coner Coner ConCodeDeploy

您可以設為藍/綠部署的AWS CodePipeline Container Coner Coner Coner Coner Coner Coner Coner Coner Coner Coner Con 在藍/綠部署中，對此您可以啟動舊版本的新版本之應用程式版本，並對此您可以測試新版本，然後再將流量路由至該版本。您也可以監控部署程序，並在發生問題時快速復原。

完成的管道會偵測映像檔或任務定義檔案的變更，並CodeDeploy使用將流量路由和部署到 Amazon ECS 叢集和負載平衡器。CodeDeploy在負載平衡器上創建一個新的偵聽器，該偵聽器可以通過特殊端口定位新任務。您也可以將管道設定為使用儲存 Amazon ECS 任務定義的來源位置 (例如儲存CodeCommit庫)。

在建立管道之前，您必須已經建立 Amazon ECS 資源、資CodeDeploy源以及負載平衡器和目標群組。您必須已將影像標記並儲存在影像存放庫中，並將工作定義和AppSpec檔案上載到您的檔案存放庫。

> **Note**
>
> 本主題描述的 Amazon ECS 轉換為CodeDeploy藍色/綠色部署動作CodePipeline。如需有關 Amazon ECS 標準部署動作的參考資訊CodePipeline，請參閱Amazon Elastic Container Service。

主題

動作類型

- 類別：`Deploy`
- 擁有者：`AWS`
- 提供者：`CodeDeployToECS`
- 版本：`1`

組態參數

ApplicationName

必要：是

中應用程式的名稱CodeDeploy。在建立管線之前，您必須已在中建立應用程式CodeDeploy。

DeploymentGroupName

必要：是

針對您為CodeDeploy應用程式建立的 Amazon ECS 任務集指定的部署群組。在建立管線之前，您必須已在中建立部署群組CodeDeploy。

TaskDefinitionTemplateArtifact

必要：是

提供工作定義檔案給部署動作的輸入成品名稱。一般而言，即為來源動作動作成品物成品物成品物成品物成品物物 使用主控台時，來源動作輸出人工因素的預設名稱為SourceArtifact。

AppSpecTemplateArtifact

必要：是

提供AppSpec檔案給部署動作的輸入成品名稱。當您的管道執行時，此值即會更新。一般而言，即為來源動作動作成品物成品物成品物成品物成品物物 使用主控台時，來源動作輸出人工因素的預設名稱為SourceArtifact。對於TaskDefinition在文AppSpec件中，您可以保<TASK_DEFINITION>留佔位符文本，如下所示。

AppSpecTemplatePath

必要：否

儲存在管線來源AppSpec檔案位置的檔案名稱，例如管線的存CodeCommit放庫。預設檔案名稱為appspec.yaml。如果您的AppSpec檔案具有相同的名稱，且儲存在檔案儲存庫的根層級，則不需要提供檔案名稱。如果路徑不是預設路徑，請輸入路徑和檔案名稱。

TaskDefinitionTemplatePath

必要：否

儲存在管線檔案來源位置的任務定義檔案名稱，例如管線的存CodeCommit放庫。預設檔案名稱為taskdef.json。如果您的任務定義檔案具有相同的名稱，且儲存在檔案存放庫的根層級，則不需要提供檔案名稱。如果路徑不是預設路徑，請輸入路徑和檔案名稱。

影像 <Number>ArtifactName

必要：否

提供影像給部署動作的輸入成品名稱。這通常是影像儲存庫的輸出成品，例如 Amazon ECR 來源動作的輸出。

的可用值<Number>為 1 到 4。

影像 <Number>ContainerName

必要：否

可從映像儲存庫取得的映像名稱，例如 Amazon ECR 來源儲存庫。

的可用值<Number>為 1 到 4。

Input artifacts (輸入成品)

- 成品數量：1 to 5
- 說明：此動CodeDeployToECS作會先在來源檔案儲存庫中尋找作業定義檔案和檔案，接著在影像儲存庫中尋找影像，然後動態產生任務定義的新修訂版本，最後執行AppSpec命令將作業集和容器部署到叢集。AppSpec

  CodeDeployToECS動作會尋找將影像 URI 對應至影像的imageDetail.json檔案。當您對 Amazon ECR 映像儲存庫提交變更時，管道 ECR 來源動作會為該提交建立imageDetail.json檔案。您也可以為未自動執行動作的管道手動新增imageDetail.json檔案。如需 imageDetail.json 詳細資訊，請參閱 適用於 Amazon ECS 藍色/綠色部署動作的 imageDetail.json 檔案。

  動作會CodeDeployToECS動態產生任務定義的新修訂版本。在這個階段中，這個動作將任務定義文件中的佔位符替換為從 ImageDetails .json 文件中檢索的圖像 URI。例如，如果您將 IMAGE1_NAME 設定為 Image1ContainerName 參數，則應將預留位置指<IMAGE1_NAME>定為任務定義檔案中影像欄位的值。在這種情況下，CodeDeployToECS 動作會<IMAGE1_NAME>將預留位置取代為從您指定為影像 1 的成品中的 ImageDetails .json 擷取的實際影像 URIArtifactName。

  對於工作定義更新，CodeDeployAppSpec.yaml檔案會包含TaskDefinition屬性。

  ```
  TaskDefinition: <TASK_DEFINITION>
  ```

  建立新任務定義之後，CodeDeployToECS動作會更新此內容。

  對於TaskDefinition欄位的值，預留位置文字必須是<TASK_DEFINITION>。動作會以CodeDeployToECS動態產生的任務定義的實際 ARN 取代此預留位置。

輸出成品

- 成品數量：0
- 描述：輸出人工因素不適用於此動作類型。

動作宣告

YAML

```
Name: Deploy
Actions:
  - Name: Deploy
    ActionTypeId:
      Category: Deploy
      Owner: AWS
      Provider: CodeDeployToECS
      Version: '1'
    RunOrder: 1
    Configuration:
      AppSpecTemplateArtifact: SourceArtifact
      ApplicationName: ecs-cd-application
      DeploymentGroupName: ecs-deployment-group
      Image1ArtifactName: MyImage
      Image1ContainerName: IMAGE1_NAME
      TaskDefinitionTemplatePath: taskdef.json
      AppSpecTemplatePath: appspec.yaml
      TaskDefinitionTemplateArtifact: SourceArtifact
    OutputArtifacts: []
    InputArtifacts:
      - Name: SourceArtifact
      - Name: MyImage
    Region: us-west-2
    Namespace: DeployVariables
```

JSON

```
{
    "Name": "Deploy",
    "Actions": [
        {
            "Name": "Deploy",
            "ActionTypeId": {
                "Category": "Deploy",
                "Owner": "AWS",
                "Provider": "CodeDeployToECS",
                "Version": "1"
            },
            "RunOrder": 1,
```

```
            "Configuration": {
                "AppSpecTemplateArtifact": "SourceArtifact",
                "ApplicationName": "ecs-cd-application",
                "DeploymentGroupName": "ecs-deployment-group",
                "Image1ArtifactName": "MyImage",
                "Image1ContainerName": "IMAGE1_NAME",
                "TaskDefinitionTemplatePath": "taskdef.json",
                "AppSpecTemplatePath": "appspec.yaml",
                "TaskDefinitionTemplateArtifact": "SourceArtifact"
            },
            "OutputArtifacts": [],
            "InputArtifacts": [
                {
                    "Name": "SourceArtifact"
                },
                {
                    "Name": "MyImage"
                }
            ],
            "Region": "us-west-2",
            "Namespace": "DeployVariables"
        }
    ]
}
```

另請參閱

以下相關資源可協助您使用此動作。

- 教學課程：使用 Amazon ECR 來源和 ECS 到部署建立管道 CodeDeploy — 本教學將CodeDeploy逐步引導您建立藍/綠部署所需的 Amazon ECS 資源。本教學將說明如何將 Docker 映像推送至 Amazon ECR，並建立列出 Docker 映像名稱、容器名稱、Amazon ECS 服務名稱和負載平衡器組態的 Amazon ECS 任務定義。此自學課程接著會逐步引導您建立部署的AppSpec檔案和管道。

> **Note**
>
> 本主題和教學課程說明的CodeDeploy /ECS 藍/綠動作CodePipeline。如需中 ECS 標準動作的相關資訊CodePipeline，請參閱教學課程：使用CodePipeline.

- AWS CodeDeploy使用者指南 — 如需如何在藍/綠部署中使用負載平衡器、生產接聽程式、目標群組和 Amazon ECS 應用程式的相關資訊，請參閱教學：部署 Amazon ECS 服務。AWS CodeDeploy 使用者指南中的這項參考資訊提供使用 Amazon ECS 和的藍/綠部署概觀AWS CodeDeploy。
- Amazon 彈性容器服務開發人員指南 — 如需使用 Docker 映像和容器、ECS 服務和叢集以及 ECS 任務集的相關資訊，請參閱什麼是 Amazon ECS？

Amazon Elastic Container Service

您可以使用亞馬遜 ECS 動作來部署 Amazon ECS 服務和任務集。Amazon ECS 服務是部署到 Amazon ECS 叢集的容器應用程式。Amazon ECS 叢集是將容器應用程式託管在雲端的執行個體集合。部署需要您在 Amazon ECS 中建立的任務定義以及影像定義檔 CodePipeline 用於部署映像。

> **Important**
>
> 亞馬遜 ECS 標準部署動作 CodePipeline 根據 Amazon ECS 服務使用的修訂，建立自己的任務定義修訂版本。如果您在不更新 Amazon ECS 服務的情況下為任務定義建立新的修訂版本，則部署動作將忽略這些修訂。

在建立管道之前，您必須已經建立 Amazon ECS 資源、將映像標記並存放在映像儲存庫中，然後上傳 BuildSpec 檔案至您的檔案儲存庫。

> **Note**
>
> 本參考主題說明下列項目的 Amazon ECS 標準部署動作 CodePipeline。有關亞馬遜 ECS 的參考信息 CodeDeploy 藍/綠部署動作 CodePipeline，請參閱Amazon Elastic Coner Coner Coner Coner Coner ConCodeDeploy。

主題

- 動作類型
- 組態參數
- Input artifacts (輸入成品)
- 輸出成品
- 動作宣告
- 另請參閱

動作類型

- 類別：Deploy
- 擁有者：AWS
- 提供者：ECS
- 版本：1

組態參數

ClusterName

必要：是

亞馬遜 ECS 中的亞馬遜 ECS 集群。

ServiceName

必要：是

您在亞馬遜 ECS 中創建的亞馬遜 ECS 服務。

FileName

必要：否

映像定義檔案的名稱、說明您的服務容器名稱、映像及標籤的 JSON 檔案。您可以將此檔案用於 ECS 標準部署。如需詳細資訊，請參閱Input artifacts (輸入成品)及適用於 Amazon ECS 標準部署動作的 imagedefinitions.json 檔案。

DeploymentTimeout

必要：否

Amazon ECS 部署動作逾時 (以分鐘為單位)。逾時值可設定為不超過此動作的預設逾時值上限。例如：

```
"DeploymentTimeout": "15"
```

Input artifacts (輸入成品)

- 成品數量： 1

- 描述：此動作會尋找`imagedefinitions.json`管線的來源檔案儲存庫中的檔案。映像定義文件為 JSON 檔案，說明 Amazon ECS 容器名稱、映像及標籤。 CodePipeline 使用檔案從映像儲存庫 (例如 Amazon ECR) 擷取映像。您可以手動新增`imagedefinitions.json`文件的管道，其中操作不是自動的。如需 `imagedefinitions.json` 詳細資訊，請參閱 適用於 Amazon ECS 標準部署動作的 imagedefinitions.json 檔案。

  此動作需要已經推送至映像儲存庫的現有映像檔。因為圖像映射由`imagedefinitions.json`檔案中，動作不需要將 Amazon ECR 來源納入管道中作為來源動作。

輸出成品

- 成品數量： `0`
- 描述：輸出成品不適用於此動作類型。

動作宣告

YAML

```
Name: DeployECS
ActionTypeId:
  Category: Deploy
  Owner: AWS
  Provider: ECS
  Version: '1'
RunOrder: 2
Configuration:
  ClusterName: my-ecs-cluster
  ServiceName: sample-app-service
  FileName: imagedefinitions.json
  DeploymentTimeout: '15'
OutputArtifacts: []
InputArtifacts:
  - Name: my-image
```

JSON

```
{
    "Name": "DeployECS",
    "ActionTypeId": {
```

```
        "Category": "Deploy",
        "Owner": "AWS",
        "Provider": "ECS",
        "Version": "1"
    },
    "RunOrder": 2,
    "Configuration": {
        "ClusterName": "my-ecs-cluster",
        "ServiceName": "sample-app-service",
        "FileName": "imagedefinitions.json",
        "DeploymentTimeout": "15"
    },
    "OutputArtifacts": [],
    "InputArtifacts": [
        {
            "Name": "my-image"
        }
    ]
},
```

另請參閱

以下相關資源可協助您使用此動作。

- 教學課程：使用 持續部署 CodePipeline— 本教學課程說明如何建立 Dockerfile (例如) CodeCommit。接下來，教學課程會讓您了解如何合併 CodeBuild BuildSpec 該文件可構建您的 Docker 映像並將其推送到亞馬遜 ECR 並創建您的圖像定義 .json 文件。最後，您建立 Amazon ECS 服務和任務定義，然後使用 Amazon ECS 部署動作建立管道。

  > **Note**
  >
  > 本主題和教學課程說明下列項目的 Amazon ECS 標準部署動作： CodePipeline。有關亞馬遜 ECS 的信息 CodeDeploy 藍/綠部署動作 CodePipeline，請參閱教學課程：使用 Amazon ECR 來源和 ECS 到部署建立管道 CodeDeploy 。

- Amazon Elastic Container Service 開發人員指南— 如需使用 Docker 映像檔和容器、Amazon ECS 服務和叢集以及 Amazon ECS 任務集的相關資訊，請參閱什麼是 Amazon ECS？

Amazon S3 動作

您可以使用 Amazon S3 部署動作將檔案部署到 Amazon S3 儲存貯體，以進行靜態網站託管或存檔。您可以指定是否在上傳到值區之前擷取部署檔案。

> **Note**
>
> 本參考主題描述 Amazon S3 部署動作，CodePipeline其中部署平台為設定用於託管的 Amazon S3 儲存貯體。如需有關中 Amazon S3 來源動作的參考資訊CodePipeline，請參閱Amazon S3 來源動作。

主題

動作類型

- 類別：`Deploy`
- 擁有者：`AWS`
- 提供者：`S3`
- 版本：`1`

組態參數

BucketName

必要：是

Amazon S3 儲存貯體名稱。

提取

必要：是

如果為 true，則指定要在上傳之前解壓縮檔案。否則，應用程式檔案會保持壓縮以供上傳，例如在託管靜態網站的情況下。如果為假，ObjectKey則需要。

ObjectKey

條件式。在 Extract = false 時需要

Amazon S3 儲存貯體中物件的 Amazon S3 儲存貯體名稱。

公里 EncryptionKey ARN

必要：否

主機儲存貯體之AWS KMS加密金鑰的 ARN。KMSEncryptionKeyARN參數會使用提AWS KMS key供的加密上傳的加工品。如果是 KMS 金鑰，您可以使用金鑰 ID、金鑰 ARN 或別名 ARN。

> **Note**
>
> 只會在建立 KMS 金鑰的帳戶中辨識別名。如果是跨帳戶動作，您只可以使用金鑰 ID 或金鑰 ARN 來識別金鑰。跨帳戶動作涉及使用來自其他帳戶 (AccountB) 的角色，因此指定金鑰 ID 將使用其他帳戶 (AccountB) 中的金鑰。

> **Important**
>
> CodePipeline只支援對稱 KMS 金鑰。請勿使用非對稱 KMS 金鑰來加密 S3 儲存貯體中的資料。

CannedACL

必要：否

此CannedACL參數會將指定的固定 ACL 套用至部署到 Amazon S3 的物件。這會覆寫已套用至物件的任何現有 ACL。

CacheControl

必要：否

該CacheControl參數可控制值區中對象的請求/響應的緩存行為。如需有效值的清單，請參閱 HTTP 操作的 Cache-Control 標頭欄位。若要在 CacheControl 中輸入多個值，請在各值之間使用逗號。如此 CLI 範例所示，您可以在每個逗號後面加上空格 (選用)：

```
"CacheControl": "public, max-age=0, no-transform"
```

Input artifacts (輸入成品)

- 成品數量：1
- 說明：用於部署或歸檔的檔案是從來源儲存庫取得、壓縮和上傳者CodePipeline。

輸出成品

- 人工因素數目：0
- 描述：輸出人工因素不適用於此動作類型。

動作組態範例

以下顯示動作組態的範例。

設定為時Extract的範例組態 false

下列範例顯示在Extract欄位設定為的情況下建立動作時的預設動作配置false。

YAML

```
Name: Deploy
Actions:
  - Name: Deploy
    ActionTypeId:
      Category: Deploy
      Owner: AWS
      Provider: S3
      Version: '1'
    RunOrder: 1
    Configuration:
      BucketName: website-bucket
      Extract: 'false'
```

```
    OutputArtifacts: []
    InputArtifacts:
      - Name: SourceArtifact
    Region: us-west-2
    Namespace: DeployVariables
```

JSON

```
{
    "Name": "Deploy",
    "Actions": [
        {
            "Name": "Deploy",
            "ActionTypeId": {
                "Category": "Deploy",
                "Owner": "AWS",
                "Provider": "S3",
                "Version": "1"
            },
            "RunOrder": 1,
            "Configuration": {
                "BucketName": "website-bucket",
                "Extract": "false"
                },
            "OutputArtifacts": [],
            "InputArtifacts": [
                {
                    "Name": "SourceArtifact"
                }
            ],
            "Region": "us-west-2",
            "Namespace": "DeployVariables"
        }
    ]
},
```

設定為時**Extract**的範例組態 **true**

下列範例顯示在Extract欄位設定為的情況下建立動作時的預設動作配置true。

YAML

```
Name: Deploy
Actions:
  - Name: Deploy
    ActionTypeId:
      Category: Deploy
      Owner: AWS
      Provider: S3
      Version: '1'
    RunOrder: 1
    Configuration:
      BucketName: website-bucket
      Extract: 'true'
      ObjectKey: MyWebsite
    OutputArtifacts: []
    InputArtifacts:
      - Name: SourceArtifact
    Region: us-west-2
    Namespace: DeployVariables
```

JSON

```
{
    "Name": "Deploy",
    "Actions": [
        {
            "Name": "Deploy",
            "ActionTypeId": {
                "Category": "Deploy",
                "Owner": "AWS",
                "Provider": "S3",
                "Version": "1"
            },
            "RunOrder": 1,
            "Configuration": {
                "BucketName": "website-bucket",
                "Extract": "true",
                "ObjectKey": "MyWebsite"
                },
            "OutputArtifacts": [],
            "InputArtifacts": [
                {
```

```
                    "Name": "SourceArtifact"
                }
            ],
            "Region": "us-west-2",
            "Namespace": "DeployVariables"
        }
    ]
},
```

另請參閱

以下相關資源可協助您使用此動作。

- 教學課程：建立使用 Amazon S3 做為部署供應商的管道— 本教學將引導您逐步瞭解使用 S3 部署動作建立管道的兩個範例。您可以下載範例檔案、將檔案上傳到CodeCommit儲存庫、建立 S3 儲存貯體，以及設定儲存貯體以進行託管。接下來，您可以使用CodePipeline主控台建立管道，並指定 Amazon S3 部署組態。
- Amazon S3 來源動作— 此動作參考提供中 Amazon S3 來源動作的參考資訊和範例CodePipeline。

Amazon S3 來源動作

當新物件上傳至已設定的儲存貯體和物件金鑰時觸發管道。

> **Note**
>
> 本參考主題 Amazon S3 了針對 CodePipeline ，其中來源位置是 Amazon S3 儲存儲體，以配置為版本控制。如需 Amazon S3 部署動作的參考資 CodePipeline 請參Amazon S3 動作。

您可以創建 Amazon S3 存儲桶，用作應用程序文件的源位置。

> **Note**
>
> 當您建立來源儲存貯體時，請確定您對儲存貯體啟用版本控制。如果您想使用現有 Amazon S3 儲存儲體，請參使用版本控制以啟用現有儲存儲體的版本控制。

如果您使用主控台建立或編輯管道， CodePipeline 建立 CloudWatch S3 來源儲存儲體發生變更時啟動管道的事件規則。

您必須先建立 Amazon S3 來源儲存儲體，並以單一 ZIP 檔案上傳來源檔案，才能透過 Amazon S3 動作連接管道。

> **Note**
>
> Amazon S3 是您管道的來源提供者時，您可將來源檔案壓縮成單一 .zip，然後將 .zip 上傳至來源儲存儲體。您也可以上傳單一解壓縮檔案；不過，預期 .zip 檔案的下游動作會失敗。

主題

動作類型

- 類別：`Source`
- 擁有者：`AWS`
- 提供者：`S3`
- 版本：`1`

組態參數

S3 儲存貯體

：必要 是

要偵測來源變更的 Amazon S3 儲存儲體名稱。

S3ObjectKey

：必要 是

要偵測來源變更的 Amazon S3 數據元密鑰名稱。

PollForSourceChanges

：必要 否

`PollForSourceChanges`控制是否 CodePipeline 會向 Amazon S3 來源儲存儲體輪詢來源變更。建議您使用 CloudWatch 事件和 CloudTrail 以偵測來源變更。如需如何設定 CloudWatch 事件，請參閱使用 S3 來源和 CloudTrail 追蹤 (CLI) 移轉輪詢管道或者使用 S3 來源和 CloudTrail 追蹤 (AWS CloudFormation範本) 移轉輪詢管道。

> **Important**
>
> 如果您打算配置 CloudWatch 事件，您必須設置`PollForSourceChanges`至`false`以避免重複的管道執行。

此參數的有效值：

- `true`：如果設置， CodePipeline 會向您的來源位置輪詢來源變更。

  > **Note**
  >
  > 如果省略`PollForSourceChanges`、 CodePipeline 默認為向您的來源位置輪詢來源變更。此行為同於包含 `PollForSourceChanges` 且設定為 `true`。

- `false`：如果設置， CodePipeline 不會向您的來源位置輪詢來源變更。如果您想要設定 CloudWatch 事件規則，以偵測來源變更。

Input artifacts (輸入成品)

- 成品數量：`0`
- 描述：輸入成品不適用於此動作類型。

輸出成品

- 成品數量： 1
- 描述：提供配置為連接至管道的來源儲存儲體中可用的成品。Amazon S3 動作的輸出成品是 Amazon S3 動作的輸出成品。Amazon S3 數據元數據（ETag 和版本 ID）顯示在 CodePipeline 作為觸發的管道執行的來源修訂。

輸出變數

設定時，此動作會產生變數，供管道中的下游動作的動作組態所參考。即使此動作沒有命名空間，此動作產生的變數仍可視為輸出變數。您可以設定動作的命名空間，讓這些變數可供下游動作的組態使用。

如需變數的詳細資訊，請參Variables。

ETag

物件的實體標籤，此物件與觸發管道的來源變更有關。ETag 是物件的 MD5 雜湊。ETag 只會反映物件內容的變更，而非中繼資料的變更。

VersionId

物件版本的版本 ID，此物件與觸發管道的來源變更有關。

動作宣告

YAML

```
Name: Source
Actions:
  - RunOrder: 1
    OutputArtifacts:
      - Name: SourceArtifact
    ActionTypeId:
      Provider: S3
      Owner: AWS
      Version: '1'
      Category: Source
    Region: us-west-2
    Name: Source
    Configuration:
```

```
      S3Bucket: my-bucket-oregon
      S3ObjectKey: my-application.zip
      PollForSourceChanges: 'false'
    InputArtifacts: []
```

JSON

```
{
    "Name": "Source",
    "Actions": [
        {
            "RunOrder": 1,
            "OutputArtifacts": [
                {
                    "Name": "SourceArtifact"
                }
            ],
            "ActionTypeId": {
                "Provider": "S3",
                "Owner": "AWS",
                "Version": "1",
                "Category": "Source"
            },
            "Region": "us-west-2",
            "Name": "Source",
            "Configuration": {
                "S3Bucket": "my-bucket-oregon",
                "S3ObjectKey": "my-application.zip",
                "PollForSourceChanges": "false"
            },
            "InputArtifacts": []
        }
    ]
},
```

另請參閱

以下相關資源可協助您使用此動作。

- 教學：建立簡易管道 (S3 儲存貯體)— 本教學課程提供示例應用程式規格檔案和 CodeDeploy 應用程式和部署組。使用本教學課程建立具有 Amazon S3 來源的管道，以部署至 Amazon EC2 實例。

AWSAppConfig

AWS AppConfig 是一項功能AWS Systems Manager。 AppConfig 支援對任何大小應用程式的受控制部署，並包括內建的驗證檢查和監視。您可以使用 AppConfig 使用在 Amazon EC2 執行個體上託管的應用程式AWS Lambda、容器、行動應用程式或 IoT 裝置。

所以此AppConfig部署操作是AWS CodePipeline操作，該操作將存儲在管道源位置的配置部署到指定的 AppConfig 應用程式、環境，和配置設定檔。它使用 AppConfig 部署策略。

動作類型

- 類別：Deploy
- 擁有者：AWS
- 提供者：AppConfig
- 版本：1

組態參數

應用

：必要 是

的 IDAWS AppConfig 應用程序以及配置和部署的詳細信息。

Environment (環境)

：必要 是

的 IDAWS AppConfig 環境中部署配置。

配置配置文件

：必要 是

的 IDAWS AppConfig 要部署的組態檔。

輸入配置路徑

：必要 是

要部署的輸入對象中配置數據的文件路徑。

部署戰略

：必要 否

所以此AWS AppConfig 部署策略以用於部署。

Input artifacts (輸入成品)

- 成品數量： 1
- 描述：部署操作的輸入對象。

輸出成品

不適用.

動作組態範例

YAML

```
name: Deploy
actions:
  - name: Deploy
    actionTypeId:
      category: Deploy
      owner: AWS
      provider: AppConfig
      version: '1'
    runOrder: 1
    configuration:
      Application: 2s2qv57
      ConfigurationProfile: PvjrpU
      DeploymentStrategy: frqt7ir
      Environment: 9tm27yd
      InputArtifactConfigurationPath: /
    outputArtifacts: []
    inputArtifacts:
      - name: SourceArtifact
    region: us-west-2
    namespace: DeployVariables
```

JSON

```
{
    "name": "Deploy",
    "actions": [
        {
            "name": "Deploy",
            "actionTypeId": {
                "category": "Deploy",
                "owner": "AWS",
                "provider": "AppConfig",
                "version": "1"
            },
            "runOrder": 1,
            "configuration": {
                "Application": "2s2qv57",
                "ConfigurationProfile": "PvjrpU",
                "DeploymentStrategy": "frqt7ir",
                "Environment": "9tm27yd",
                "InputArtifactConfigurationPath": "/"
            },
            "outputArtifacts": [],
            "inputArtifacts": [
                {
                    "name": "SourceArtifact"
                }
            ],
            "region": "us-west-2",
            "namespace": "DeployVariables"
        }
    ]
}
```

另請參閱

以下相關資源可協助您使用此動作。

- AWSAppConfig— 如需的相關資訊AWS AppConfig 部署的組態，請參。AWS Systems Manager使用者指南。
- 教學：建立將 AWS AppConfig 做為部署提供者使用的管道— 本教程讓您開始設置簡單的部署配置文件和 AppConfig 資源，並向您展示如何使用主控台來建立具有AWS AppConfig 部署操作。

AWS CloudFormation

在 AWS CloudFormation 堆疊上執行操作。堆疊是一組 AWS 資源，您可將它視為單一單位進行管理。堆疊中的資源都是由堆疊的 AWS CloudFormation 範本定義。變更集會建立比較，無須變更原始堆疊即可檢視。如需可在堆疊和變更集上執行之 AWS CloudFormation 動作類型的資訊，請參閱 `ActionMode` 參數。

若要針對堆疊作業失敗的AWS CloudFormation動作建構錯誤訊息，請 CodePipeline 呼叫 AWS CloudFormation `DescribeStackEvents` API。如果動作 IAM 角色具有存取該 API 的權限，則有關第一個失敗資源的詳細資訊將包含在 CodePipeline 錯誤訊息中。否則，如果角色原則沒有適當的權限， CodePipeline 將忽略存取 API 並改為顯示一般錯誤訊息。若要這麼做，必須將`cloudformation:DescribeStackEvents`權限新增至管道的服務角色或其他 IAM 角色。

如果您不希望資源詳細資訊顯示在管道錯誤訊息中，可以移除權限來撤銷此動作 IAM 角色的`cloudformation:DescribeStackEvents`權限。

主題

動作類型

- 類別：`Deploy`
- 擁有者：`AWS`
- 提供者：`CloudFormation`
- 版本：`1`

組態參數

ActionMode

必要：是

ActionMode 是在 AWS CloudFormation 執行堆疊或變更集上的動作名稱。以下是可用的動作模式：

- CHANGE_SET_EXECUTE 根據一組指定的資源更新，為資源堆疊執行變更集。使用此動作，AWS CloudFormation 會開始改變堆疊。
- CHANGE_SET_REPLACE 根據您提交的堆疊名稱和範本建立變更組 (若不存在的話)。若變更組已存在，AWS CloudFormation 會刪除它，並建立新的變更組。
- CREATE_UPDATE 建立堆疊 (如果堆疊不存在)。若堆疊存在，則 AWS CloudFormation 會更新堆疊。使用此動作來更新現有堆疊。不同於REPLACE_ON_FAILURE，如果堆棧存在並處於失敗狀態，則 CodePipeline 不會刪除並替換堆棧。
- DELETE_ONLY 刪除堆疊。若您指定不存在的堆疊，動作會成功完成，而不會刪除任何堆疊。
- REPLACE_ON_FAILURE 建立堆疊 (若堆疊不存在)。若堆疊存在且處於故障狀態，則 AWS CloudFormation 會刪除堆疊並建立新的堆疊。若堆疊並未處於故障狀態，AWS CloudFormation 會更新它。

  若在 AWS CloudFormation 中顯示下列任何狀態類型，堆疊會處於故障狀態：

  - ROLLBACK_FAILED
  - CREATE_FAILED
  - DELETE_FAILED
  - UPDATE_ROLLBACK_FAILED

  使用此動作來自動取代故障的堆疊，無須復原或故障診斷。

  > **Important**
  >
  > 建議您使用 REPLACE_ON_FAILURE 僅作為測試目的，因為它可能會刪除您的堆疊。

StackName

必要：是

StackName 是現有堆疊或您希望建立之堆疊的名稱。

功能

必要：有條件

使用 Capabilities 可確認範本具有自行建立和更新一些資源的功能，而且這些功能是根據範本中的資源類型來決定。

如果您的堆疊範本中有 IAM 資源，或您直接從包含巨集的範本建立堆疊，則此屬性為必要。為了讓 AWS CloudFormation 動作可成功以這種方式操作，您必須明確確認您要使用下列其中一個功能執行此操作：

- CAPABILITY_IAM
- CAPABILITY_NAMED_IAM
- CAPABILITY_AUTO_EXPAND

您可以透過在功能間使用逗號 (無空格) 來指定多個功能。動作宣告 中的範例顯示具有 CAPABILITY_IAM 和 CAPABILITY_AUTO_EXPAND 屬性的項目。

有關更多信息Capabilities，請參閱 AWS CloudFormationAPI 參考UpdateStack中的屬性。

ChangeSetName

必要：有條件

ChangeSetName 是現有變更集或您希望為指定堆疊建立之新變更集的名稱。

針對下列動作模式，此屬性為必要：CHANGE_SET_REPLACE 和 CHANGE_SET_EXECUTE。針對其他所有動作模式，可忽略此屬性。

RoleArn

必要：有條件

RoleArn 是在指定堆疊中的資源上操作時 AWS CloudFormation 所假設 IAM 服務角色的 ARN。執行變更集時，不會套用 RoleArn。如果您不使用 CodePipeline 來建立變更集，請確定變更集或堆疊具有相關聯的角色。

> **Note**
>
> 此角色必須與正在執行之動作的角色位於相同的帳戶中，如動作宣告中所配置RoleArn。

針對下列動作模式，此屬性為必要：

- CREATE_UPDATE
- REPLACE_ON_FAILURE
- DELETE_ONLY
- CHANGE_SET_REPLACE

> **Note**
>
> AWS CloudFormation會為範本提供 S3 簽署的 URL；因此，這`RoleArn`不需要存取成品值區的權限。不過，動作`RoleArn`確實需要存取成品值區的權限，才能產生已簽署的 URL。

TemplatePath

必要：有條件

`TemplatePath` 代表 AWS CloudFormation 範本檔案。您將檔案包含在此動作的輸入成品中。該檔案名稱遵循此格式：

`Artifactname``::`*`TemplateFileName`*

`Artifactname` 是出現在 CodePipeline 中的輸入成品名稱。例如，來源階段的成品名稱為 `SourceArtifact` 且 `template-export.json` 檔案名稱建立 `TemplatePath` 名稱，如此範例所顯示：

```
"TemplatePath": "SourceArtifact::template-export.json"
```

針對下列動作模式，此屬性為必要：

- CREATE_UPDATE
- REPLACE_ON_FAILURE
- CHANGE_SET_REPLACE

針對其他所有動作模式，可忽略此屬性。

> **Note**
>
> 包含AWS CloudFormation模板主體的模板文件的最小長度為 1 個字節，最大長度為 1 MB。對於中的AWS CloudFormation部署動作 CodePipeline，輸入人工因素大小上限一律

為 256 MB。如需詳細資訊，請參閱AWS CodePipeline 中的配額和 AWS CloudFormation 的限制。

OutputFileName

必要：否

用OutputFileName於指定輸出檔案名稱，例如CreateStackOutput.json，為此動作 CodePipeline 新增至管線輸出人工因素。JSON 檔案包含 AWS CloudFormation 堆疊中 Outputs 區段的內容。

如果未指定名稱，則不 CodePipeline 會產生輸出檔案或成品。

ParameterOverrides

必要：否

參數會定義於您的堆疊範本中，並可讓您在建立或更新堆疊時為其提供值。您可以使用 JSON 物件在您的範本中設定參數值。(這些值會覆寫範本組態檔案中設定的值。) 如需使用參數覆寫的詳細資訊，請參閱 組態屬性 (JSON 物件)。

我們建議您大多數的參數值都使用範本組態檔案。僅對管道執行前未知的值使用參數覆寫。如需詳細資訊，請參閱《使用指南》中的〈搭 CodePipeline 配管線使用參數覆寫函數〉。AWS CloudFormation

> **Note**
>
> 所有參數名稱都必須在堆疊範本中存在。

TemplateConfiguration

必要：否

TemplateConfiguration 是範本組態檔案。您將檔案包含在此動作的輸入成品中。它可以包含範本參數值和堆疊政策。如需範本組態檔案格式的詳細資訊，請參閱AWS CloudFormation人工因素。

範本組態檔案名稱遵循此格式：

Artifactname::*TemplateConfigurationFileName*

`Artifactname` 是出現在 CodePipeline 中的輸入成品名稱。例如，來源階段的成品名稱為 `SourceArtifact` 且 `test-configuration.json` 檔案名稱建立 `TemplateConfiguration` 名稱，如此範例所顯示：

```
"TemplateConfiguration": "SourceArtifact::test-configuration.json"
```

Input artifacts (輸入成品)

- 人工因素數目：`0 to 10`
- Description: (描述：) 做為輸入，AWS CloudFormation 動作會選擇性地接受成品用於這些用途：
  - 提供要執行的堆疊範本檔案。(請參閱 `TemplatePath` 參數。)
  - 提供要使用的範本組態檔案。(請參閱 `TemplateConfiguration` 參數。) 如需範本組態檔案格式的詳細資訊，請參閱AWS CloudFormation人工因素。
  - 部署作為 AWS CloudFormation 堆疊的一部分 Lambda 函數提供成品。

輸出成品

- 人工因素數目：`0 to 1`
- Description: (描述：) 如果指定 `OutputFileName` 參數，則此動作會產生一個包含具有指定名稱的 JSON 檔案的輸出成品。JSON 檔案包含 AWS CloudFormation 堆疊中 Outputs (輸出) 區段的內容。

  如需您可為 AWS CloudFormation 動作建立輸出區段的詳細資訊，請參閱 Outputs (輸出)。

輸出變數

設定時，此動作會產生變數，供管道中的下游動作的動作組態所參考。您可以設定動作的命名空間，讓這些變數可供下游動作的組態使用。

對於 AWS CloudFormation 動作，變數是透過在堆疊範本的 `Outputs` 區段中指定的任何值而產生的。請注意，生成輸出的唯一 CloudFormation動作模式是導致創建或更新堆棧的操作模式，例如堆棧創建，堆棧更新和更改集執行。產生變數的相應動作模式如下：

- `CHANGE_SET_EXECUTE`
- `CHANGE_SET_REPLACE`

- CREATE_UPDATE
- REPLACE_ON_FAILURE

如需詳細資訊，請參閱Variables。如需說明如何在使用 CloudFormation輸出變數的管線中使用 CloudFormation 部署動作建立管線的教學課程，請參閱教學：建立透過 AWS CloudFormation 部署動作使用變數的管道。

動作宣告

YAML

```
Name: ExecuteChangeSet
ActionTypeId:
  Category: Deploy
  Owner: AWS
  Provider: CloudFormation
  Version: '1'
RunOrder: 2
Configuration:
  ActionMode: CHANGE_SET_EXECUTE
  Capabilities: CAPABILITY_NAMED_IAM,CAPABILITY_AUTO_EXPAND
  ChangeSetName: pipeline-changeset
  ParameterOverrides: '{"ProjectId": "my-project","CodeDeployRole":
 "CodeDeploy_Role_ARN"}'
  RoleArn: CloudFormation_Role_ARN
  StackName: my-project--lambda
  TemplateConfiguration: 'my-project--BuildArtifact::template-configuration.json'
  TemplatePath: 'my-project--BuildArtifact::template-export.yml'
OutputArtifacts: []
InputArtifacts:
  - Name: my-project-BuildArtifact
```

JSON

```
{
    "Name": "ExecuteChangeSet",
    "ActionTypeId": {
        "Category": "Deploy",
        "Owner": "AWS",
        "Provider": "CloudFormation",
        "Version": "1"
```

```
    },
    "RunOrder": 2,
    "Configuration": {
        "ActionMode": "CHANGE_SET_EXECUTE",
        "Capabilities": "CAPABILITY_NAMED_IAM,CAPABILITY_AUTO_EXPAND",
        "ChangeSetName": "pipeline-changeset",
        "ParameterOverrides": "{\"ProjectId\": \"my-project\",\"CodeDeployRole\":
 \"CodeDeploy_Role_ARN\"}",
        "RoleArn": "CloudFormation_Role_ARN",
        "StackName": "my-project--lambda",
        "TemplateConfiguration": "my-project--BuildArtifact::template-
configuration.json",
        "TemplatePath": "my-project--BuildArtifact::template-export.yml"
    },
    "OutputArtifacts": [],
    "InputArtifacts": [
        {
             "Name": "my-project-BuildArtifact"
        }
    ]
},
```

另請參閱

以下相關資源可協助您使用此動作。

- 配置屬性參考 — 《AWS CloudFormation使用指南》中的本參考章節提供了這些 CodePipeline 參數的更多描述和範例。
- AWS CloudFormationAPI 參考 — AWS CloudFormationAPI CreateStack參考中的參數描述了AWS CloudFormation範本的堆疊參數。

AWS CloudFormation StackSets

CodePipeline 提供了作為 CI/CD 過程的一部分執行AWS CloudFormation StackSets 操作的能力。您可以使用堆疊集，透過使用單一AWS CloudFormation範本在跨AWS區域的AWS帳戶中建立堆疊。每個堆棧中包含的所有資源均由堆棧集的AWS CloudFormation模板定義。建立堆疊集時，您可以指定要使用的範本，以及範本需要的任何參數和功能。

> **Note**
>
> 您必須使用「Or AWS ganizations」管理帳戶來部署AWS CloudFormation StackSets。您無法針對此動作使用委派的管理帳戶。

若要取得有關概念的更多資訊 AWS CloudFormation StackSets，請參閱《AWS CloudFormation使用指南》中的StackSets 概念。

您可以AWS CloudFormation StackSets 透過兩種不同的動作類型一起使用來整合管道：

- 此`CloudFormationStackSet`動作會從儲存在管線來源位置的範本建立或更新堆疊集或堆疊執行個體。每次建立或更新堆疊集時，都會對指定的執行個體啟動這些變更的部署。在主控台中，您可以在建立或編輯管道時選擇「CloudFormation 堆疊集合」動作提供者。
- 此`CloudFormationStackInstances`動作會將動作的變更部署到指定的執`CloudFormationStackSet`行個體、建立新的堆疊執行個體，以及定義指定執行個體的參數覆寫。在主控台中，您可以在編輯現有管道時選擇「CloudFormation 堆疊執行個體」動作提供者。

> **Note**
>
> 該`CloudFormationStackSet`和`CloudFormationStackInstances`行動不適用於亞太區域 (香港)、歐洲 (蘇黎世)、歐洲 (米蘭)、非洲 (開普敦) 及中東 (巴林) 等地區。若要參考其他可用動作，請參閱產品與服務整合 CodePipeline。

您可以使用這些動作來部署至目標AWS帳戶或目標組 Organ AWS izations 單位 ID。

> **Note**
>
> 若要部署到目標組 Organ AWS izations 帳戶或組織單位 ID，並使用服務管理的權限模型，您必須啟用AWS CloudFormation StackSets 和組 Organ AWS izations 之間的信任存取。如需詳細資訊，請參閱使用 AWS CloudFormation Stackset 啟用受信任存取。

主題

- AWS CloudFormation StackSets 動作如何運作
- 如何在管道中建構 StackSets動作

AWS CloudFormation StackSets 動作如何運作

CloudFormationStackSet動作會根據動作是否第一次執行，建立或更新資源。

此CloudFormationStackSet動作會建立或更新堆疊集，並將這些變更部署到指定的執行個體。

> **Note**
>
> 如果您使用此動作進行包含新增堆疊執行個體的更新，則會先部署新執行個體，更新最後完成。新執行個體會先收到舊版本，然後將更新套用至所有執行個體。

- 建立：當未指定執行個體且堆疊集不存在時，CloudFormationStackSet動作會建立堆疊集，而不建立任何執行個體。
- 更新：針對已建立的堆疊集執行CloudFormationStackSet動作時，動作會更新堆疊集。如果未指定執行個體且堆疊集已存在，則會更新所有執行個體。如果使用此動作更新特定執行個體，則所有剩餘的執行個體都會移至「過期」狀態。

  您可以透過下列方式使用此CloudFormationStackSet動作來更新堆疊集合。

  - 更新部分或全部執行個體的範本。
  - 更新部分或全部執行個體的參數。
  - 更新堆疊集的執行角色 (這必須符合系統管理員角色中指定的執行角色)。
  - 變更權限模型 (僅在尚未建立實例的情況下)。
  - AutoDeployment如果堆疊設定權限模型為啟用/停用Service Managed。
  - 更新管理員角色。
  - 更新堆疊集的描述。
  - 將部署目標新增至堆疊集更新，以建立新的堆疊執行個體。

此CloudFormationStackInstances動作會建立新堆疊執行個體或更新過期的堆疊執行個體 當堆疊集合更新時，執行個體就會過期，但並非其中的所有執行個體都會更新。

- 建立：如果堆疊已存在，CloudFormationStackInstances動作只會更新執行個體，而不會建立堆疊執行個體。
- 更新：執行CloudFormationStackSet動作後，如果僅在某些實例中更新了模板或參數，則其餘的將被標記OUTDATED。在稍後的管線階段中，CloudFormationStackInstances更新堆疊中以波浪形式設定的其餘執行個體，以便標記所有執行個體CURRENT。此動作也可用於在新執行個體或現有執行個體上新增其他執行個體或覆寫參數。

作為更新的一部分，CloudFormationStackSet和CloudFormationStackInstances動作可以指定新的部署目標，以建立新的堆疊執行個體。

作為更新的一部分，CloudFormationStackSet和CloudFormationStackInstances動作不會刪除堆疊集、執行個體或資源。當動作更新堆疊但未指定所有要更新的執行個體時，未指定要更新的執行個體會從更新中移除，並將其狀態設定為OUTDATED。

在部署期間，堆疊執行個體也可以顯示部署至執行個體是OUTDATED否失敗的狀態。

如何在管道中建構 StackSets動作

最佳作法是建構管道，以便建立堆疊集，並初始部署至子集或單一執行個體。測試部署並檢視產生的堆疊集之後，請新增CloudFormationStackInstances動作，以便建立和更新剩餘的執行個體。

使用主控台或 CLI 建立建議的管線結構，如下所示：

1. 使用來源動作 (必要) 和動作做為部署CloudFormationStackSet動作來建立管道。執行您的管道。
2. 當您的管道第一次執行時，CloudFormationStackSet動作會建立您的堆疊集和至少一個初始執行個體。驗證堆疊集的建立，並檢閱初始執行個體的部署。例如，對於帳戶 Account-A 的初始堆疊集建立，其中us-east-1是指定的區域，會使用堆疊集建立堆疊執行個體：

堆棧實例	區域	Status
StackInstance一号	us-east-1	CURRENT

3. 編輯管道以新增CloudFormationStackInstances為第二個部署動作，以針對您指定的目標建立/更新堆疊執行個體。例如，針對指定和 Region 的帳戶建立堆疊執行Account-A個體us-

east-2，eu-central-1則會建立剩餘的堆疊執行個體，並且初始執行個體會保持更新，如下所示：

堆棧實例	區域	Status
StackInstance一号	us-east-1	CURRENT
StackInstanceID-2	us-east-2	CURRENT
StackInstanceID-3	eu-central-1	CURRENT

4. 視需要執行管道以更新堆疊集並更新或建立堆疊執行個體。

當您起始堆疊更新 (其中您已從動作組態中移除部署目標) 時，未指定要更新的堆疊執行個體會從部署中移除，並移至「過期」狀態。例如，對於從動作配置中移除「us-east-2區域」Account-A 的帳戶的堆疊執行個體更新，則會建立剩餘的堆疊執行個體，並將移除的執行個體設定為「過期」，如下所示：

堆棧實例	區域	Status
StackInstance一号	us-east-1	CURRENT
StackInstanceID-2	us-east-2	過時的
StackInstanceID-3	eu-central-1	CURRENT

如需有關部署堆疊集的最佳作法的詳細資訊，請參閱《AWS CloudFormation使用者指南》 StackSets中的最佳作法。

CloudFormationStackSet 動作

此動作會從儲存在管線來源位置的範本建立或更新堆疊集。

定義堆疊集之後，您可以在組態參數中指定的目標帳戶和區域中建立、更新或刪除堆疊。建立、更新和刪除堆疊時，您可以指定其他偏好設定，例如要執行作業的區域順序、堆疊作業停止的失敗容錯百分比，以及在堆疊上同時執行作業的帳戶數目。

堆疊集是區域性資源。如果您在一個AWS區域中建立堆疊組合，則無法從其他區域存取該堆疊集。

使用此動作做為堆疊集的更新動作時，如果沒有部署至少一個堆疊執行個體，則不允許對堆疊進行更新。

主題

動作類型

- 類別：`Deploy`
- 擁有者：`AWS`
- 提供者：`CloudFormationStackSet`
- 版本：`1`

組態參數

StackSetName

必要：是

要與堆疊集相關聯的名稱。此名稱在建立該名稱的區域中必須是唯一的。

名稱只能包含英數字元和連字號字元。它必須以字母字元開頭，且不得超過 128 個字元。

描述

必要：否

堆疊集的描述。您可以使用它來描述堆棧集的目的或其他相關信息。

TemplatePath

必要：是

定義堆疊集中資源的範本位置。這必須指向具有 460,800 字節的最大大小的模板。

以格式輸入來源人工因素名稱和範本檔案的路徑"InputArtifactName::TemplateFileName"，如下列範例所示。

```
SourceArtifact::template.txt
```

參數

必要：否

部署期間更新之堆疊集的範本參數清單。

您可以將參數提供為常值清單或檔案路徑：

- 您可以使用下列速記語法格式輸入參數：.
  ParameterKey=string,ParameterValue=string,UsePreviousValue=boolean,ResolvedV
  ParameterKey=string,ParameterValue=string,UsePreviousValue=boolean,ResolvedV
  如需這些資料類型的詳細資訊，請參閱模板參數數據類型。

  下列範例會示範以 my-值命名BucketName的參數bucket。

  ```
  ParameterKey=BucketName,ParameterValue=my-bucket
  ```

  下列範例顯示具有多個參數的項目：

  ```
  ParameterKey=BucketName,ParameterValue=my-bucket
  ParameterKey=Asset1,ParameterValue=true
  ParameterKey=Asset2,ParameterValue=true
  ```

- 您可以輸入包含以格式輸入的範本參數取代清單的檔案位置"InputArtifactName::ParametersFileName"，如下列範例所示。

  ```
  SourceArtifact::parameters.txt
  ```

  下列範例顯示的檔案內容parameters.txt。

  ```
  [
  ```

```
    {
        "ParameterKey": "KeyName",
        "ParameterValue": "true"
    },
    {
        "ParameterKey": "KeyName",
        "ParameterValue": "true"
    }
]
```

功能

必要：否

指出範本可以建立和更新資源，具體取決於範本中的資源類型。

如果您的堆疊範本中有 IAM 資源，或者直接從包含巨集的範本建立堆疊，則必須使用此屬性。若要讓AWS CloudFormation動作以這種方式成功運作，您必須使用下列其中一項功能：

- CAPABILITY_IAM
- CAPABILITY_NAMED_IAM

您可以使用逗號指定多個功能，而且權能之間不能使用空格。中的範例示例CloudFormationStackSet操作配置顯示具有多個權能的項目。

PermissionModel

必要：否

決定建立和管理 IAM 角色的方式。如果未指定欄位，則使用預設值。如需相關資訊，請參閱 堆疊集作業的權限模型。

有效的 值如下：

- SELF_MANAGED(預設值)：您必須建立管理員和執行角色，才能部署到目標帳戶。
- SERVICE_MANAGED：AWS CloudFormation StackSets 自動建立部署到由 Organ AWS izations 管理的帳戶所需的 IAM 角色。這需要一個帳戶是組織的成員。

> **Note**
>
> 只有當堆疊集中沒有堆疊執行個體時，才能變更此參數。

AdministrationRoleArn

> **Note**
>
> 由於跨多個帳戶AWS CloudFormation StackSets 執行作業，因此您必須先在這些帳戶中定義必要的權限，才能建立堆疊集。

必要：否

> **Note**
>
> 此參數對於 SELF_MANAGED 權限模型是可選的，不適用於服務 _ 託管權限模型。

管理員帳戶中 IAM 角色的 ARN，用於執行堆疊集作業。

名稱可以包含字母數字字元，以及下列任何字元：_+=、.@-，且不可包含空格。名稱不區分大小寫。此角色名稱的長度下限必須為 20 個字元，長度上限為 2048 個字元。角色名稱在帳戶中必須是唯一的。此處指定的角色名稱必須是現有的角色名稱。如果您未指定角色名稱，則會將其設定為 AWSCloudFormationStackSetAdministrationRole。如果您指定 ServiceManaged，則不能定義角色名稱。

ExecutionRoleName

> **Note**
>
> 由於跨多個帳戶AWS CloudFormation StackSets 執行作業，因此您必須先在這些帳戶中定義必要的權限，才能建立堆疊集。

必要：否

> **Note**
>
> 此參數對於 SELF_MANAGED 權限模型是可選的，不適用於服務 _ 託管權限模型。

用於執行堆疊集作業的目標帳戶中 IAM 角色的名稱。名稱可以包含字母數字字元，以及下列任何字元：_+=、.@-，且不可包含空格。名稱不區分大小寫。此角色名稱的長度下限必須為 1 個字元，

且長度上限為 64 個字元。角色名稱在帳戶中必須是唯一的。此處指定的角色名稱必須是現有的角色名稱。如果您使用自訂的執行角色，請勿指定此角色。如果您未指定角色名稱，則會將其設定為AWSCloudFormationStackSetExecutionRole。如果您將服務 _ 託管設置為真，則不能定義角色名稱。

OrganizationsAutoDeployment

必要：否

> **Note**
>
> 此參數對於「服務 _ 託管」權限模型是可選的，不適用於 SELF_MANAGED 權限模型。

描述是否AWS CloudFormation StackSets 自動部署至AWS新增至目標組織或組織單位 (OU) 的組織帳戶。如果OrganizationsAutoDeployment已指定，請勿指定DeploymentTargets和Regions。

> **Note**
>
> 如果未提供任何輸入OrganizationsAutoDeployment，則預設值為Disabled。

有效的 值如下：

- Enabled。 需要：否。

  StackSets 自動將其他堆疊執行個體部署到AWS新增至指定區域中目標組織或組織單位 (OU) 的組織帳戶。如果帳戶從目標組織或 OU 中移AWS CloudFormation StackSets 除，請從指定區域中的帳戶中刪除堆疊執行個體。

- Disabled。 需要：否。

  StackSets 不會自動將其他堆疊執行個體部署到AWS新增至指定區域中目標組織或組織單位 (OU) 的組織帳戶。

- EnabledWithStackRetention。 需要：否。

  從目標組織或 OU 移除帳號時，會保留堆疊資源。

DeploymentTargets

必要：否

> **Note**
>
> 對於 SERVICE_MANAGED 權限模型，您可以為部署目標提供組織根 ID 或組織單位識別碼。對於 SELF_MANAGED 權限模型，您只能提供帳戶。

> **Note**
>
> 選取此參數時，您還必須選取「區域」(Region)。

應在其中建立/更新堆疊集執行個體的AWS帳戶或組織單位 ID 清單。

- 帳戶：

  您可以將帳戶提供為常值清單或檔案路徑：

  - 常值：以速記語法格式輸入參數`account_ID,account_ID`，如下列範例所示。

    ```
    111111222222,333333444444
    ```

  - 檔案路徑：包含應在其中建立/更新堆疊集實例的AWS帳戶清單的檔案位置，並以格式輸入。`InputArtifactName::AccountsFileName`如果您使用檔案路徑來指定帳戶或 OrganizationalUnitIds，檔案格式必須為 JSON，如下列範例所示。

    ```
    SourceArtifact::accounts.txt
    ```

    下列範例顯示的檔案內容`accounts.txt`。

    ```
    [
        "111111222222"
    ]
    ```

    下列範例顯示列出多個帳號`accounts.txt`時的檔案內容：

    ```
    [
        "111111222222","333333444444"
    ]
    ```

- OrganizationalUnitIds:

> **Note**
>
> 此參數對於「服務 _ 託管」權限模型是可選的，不適用於 SELF_MANAGED 權限模型。如果您選取，請勿使用此選項OrganizationsAutoDeployment。

要在其中更新關聯堆疊執行個體的AWS組織單位。

您可以提供組織單位 ID 做為常值清單或檔案路徑：

- 常值：輸入以逗號分隔的字串陣列，如下列範例所示。

```
ou-examplerootid111-exampleouid111,ou-examplerootid222-exampleouid222
```

- 檔案路徑：包含要 OrganizationalUnitIds 在其中建立或更新堆疊集實例的清單的檔案位置。如果您使用檔案路徑來指定帳戶或 OrganizationalUnitIds，檔案格式必須為 JSON，如下列範例所示。

  以格式輸入檔案的路徑`InputArtifactName::OrganizationalUnitIdsFileName`。

```
SourceArtifact::OU-IDs.txt
```

  下列範例顯示的檔案內容`OU-IDs.txt`：

```
[
    "ou-examplerootid111-exampleouid111","ou-examplerootid222-exampleouid222"
]
```

區域

必要：否

> **Note**
>
> 選取此參數時，您也必須選取DeploymentTargets。

建立或更新堆疊集執行個體的AWS區域清單。區域會依照輸入的順序進行更新。

以格式輸入有效「AWS區域」清單`Region1,Region2`，如下列範例所示。

```
us-west-2,us-east-1
```

FailureTolerancePercentage

必要：否

在AWS CloudFormation停止該區域的作業之前，此堆疊作業可能會失敗的每個區域的帳戶百分比。如果「區域」中的作業已停止，則AWS CloudFormation不會在後續「區域」中嘗試該作業。根據指定的百分比計算帳戶數目時，AWS CloudFormation 會無條件捨去到下一個整數。

MaxConcurrentPercentage

必要：否

執行此操作時，一次可用的帳戶百分比上限。根據指定的百分比計算帳戶數目時，AWS CloudFormation 會無條件捨去到下一個整數。如果向下舍入會導致零，則將數字AWS CloudFormation設置為一。雖然您使用此設定來指定最大值，但對於大型部署，由於服務節流，實際並行處理的帳戶數目可能會降低。

RegionConcurrencyType

必要：否

您可以設定 Region 並行部署參數，指定堆疊集應AWS 區域依序或 parallel 部署。如果指定區域 parallel 部署多AWS 區域個堆疊，可以加快整體部署時間。

- 並行：只要區域的部署失敗不超過指定的故障容忍，就會同時執行堆疊集合部署。
- 循序：只要區域的部署失敗不超過指定的故障容忍，就會一次執行一個堆疊集部署。順序部署是預設選項。

ConcurrencyMode

必要：否

並行模式可讓您選擇並行層級在堆疊集合作業期間的行為方式，無論是具有嚴格或軟式失敗容錯。嚴格容錯能力會降低部署速度，由於每次故障會使並行值減少，因此堆疊集操作會發生故障。軟性容錯能力會優先考慮部署速度，同時仍然利用 AWS CloudFormation 安全功能。

- STRICT_FAILURE_TOLERANCE：此選項會動態降低並行層次，以確保失敗的帳戶數目永遠不會超過特定的失敗容錯。這是預設行為。
- SOFT_FAILURE_TOLERANCE：此選項將失敗容忍與實際並行分離。這可讓堆疊集合作業在設定的並行層級執行，而不論失敗次數為何。

Input artifacts (輸入成品)

您必須在CloudFormationStackSet動作中包含至少一個包含堆疊集範本的輸入成品。您可以為部署目標、帳戶和參數清單包含更多輸入成品。

- 人工因素數目：1 to 3
- 描述：您可以包含人工因素以提供：
  - 堆疊範本檔案。(請參閱 TemplatePath 參數。)
  - 參數檔案。(請參閱 Parameters 參數。)
  - 帳戶檔案。(請參閱 DeploymentTargets 參數。)

輸出成品

- 人工因素數目：0
- 描述：輸出人工因素不適用於此動作類型。

輸出變數

如果您配置此動作，它會產生可供管線中下游動作的動作組態參照的變數。您可以設定動作的命名空間，讓這些變數可供下游動作的組態使用。

- StackSetId：堆疊集合的識別碼。
- OperationId：堆疊集合作業的識別碼。

如需詳細資訊，請參閱Variables。

示例CloudFormationStackSet操作配置

下列範例顯示動作的動CloudFormationStackSet作組態。

自我管理權限模型的範例

下列範例顯示輸入的部署目標為AWS帳號 ID 的CloudFormationStackSet動作。

YAML

```
Name: CreateStackSet
ActionTypeId:
```

```
  Category: Deploy
  Owner: AWS
  Provider: CloudFormationStackSet
  Version: '1'
RunOrder: 1
Configuration:
  DeploymentTargets: '111111222222'
  FailureTolerancePercentage: '20'
  MaxConcurrentPercentage: '25'
  PermissionModel: SELF_MANAGED
  Regions: us-east-1
  StackSetName: my-stackset
  TemplatePath: 'SourceArtifact::template.json'
OutputArtifacts: []
InputArtifacts:
  - Name: SourceArtifact
Region: us-west-2
Namespace: DeployVariables
```

JSON

```
{
    "Name": "CreateStackSet",
    "ActionTypeId": {
        "Category": "Deploy",
        "Owner": "AWS",
        "Provider": "CloudFormationStackSet",
        "Version": "1"
    },
    "RunOrder": 1,
    "Configuration": {
        "DeploymentTargets": "111111222222",
        "FailureTolerancePercentage": "20",
        "MaxConcurrentPercentage": "25",
        "PermissionModel": "SELF_MANAGED",
        "Regions": "us-east-1",
        "StackSetName": "my-stackset",
        "TemplatePath": "SourceArtifact::template.json"
    },
    "OutputArtifacts": [],
    "InputArtifacts": [
        {
            "Name": "SourceArtifact"
```

```
        }
    ],
    "Region": "us-west-2",
    "Namespace": "DeployVariables"
}
```

服務管理權限模型的範例

下列範例顯示服務管理權限模型的CloudFormationStackSet動作，其中啟用堆疊保留功能的 auto 動部署至 Organ AWS izations 的選項。

YAML

```
Name: Deploy
ActionTypeId:
  Category: Deploy
  Owner: AWS
  Provider: CloudFormationStackSet
  Version: '1'
RunOrder: 1
Configuration:
  Capabilities: 'CAPABILITY_IAM,CAPABILITY_NAMED_IAM'
  OrganizationsAutoDeployment: EnabledWithStackRetention
  PermissionModel: SERVICE_MANAGED
  StackSetName: stacks-orgs
  TemplatePath: 'SourceArtifact::template.json'
OutputArtifacts: []
InputArtifacts:
  - Name: SourceArtifact
Region: eu-central-1
Namespace: DeployVariables
```

JSON

```
{
    "Name": "Deploy",
    "ActionTypeId": {
        "Category": "Deploy",
        "Owner": "AWS",
        "Provider": "CloudFormationStackSet",
        "Version": "1"
    },
```

```
    "RunOrder": 1,
    "Configuration": {
        "Capabilities": "CAPABILITY_IAM,CAPABILITY_NAMED_IAM",
        "OrganizationsAutoDeployment": "EnabledWithStackRetention",
        "PermissionModel": "SERVICE_MANAGED",
        "StackSetName": "stacks-orgs",
        "TemplatePath": "SourceArtifact::template.json"
    },
    "OutputArtifacts": [],
    "InputArtifacts": [
        {
            "Name": "SourceArtifact"
        }
    ],
    "Region": "eu-central-1",
    "Namespace": "DeployVariables"
}
```

CloudFormationStackInstances 動作

此動作會建立新執行個體，並將堆疊集部署到指定的執行個體。「堆疊執行個體」是針對區域內某個目標帳戶中的堆疊所做的參考。堆棧實例可以在沒有堆棧的情況下存在; 例如，如果堆棧創建不成功，堆棧實例顯示堆棧創建失敗的原因。堆疊執行個體僅與一個堆疊集相關聯。

初始建立堆疊集之後，您可以使用來新增堆疊執行個體CloudFormationStackInstances。在建立或更新堆疊集執行個體作業期間，可以在堆疊執行個體層級覆寫範本參數值。

每個堆疊集都有一個範本和一組範本參數。當您更新範本或範本參數時，會更新整個參數集的參數。然後，所有執行個體狀態都會設定為，OUTDATED直到變更部署至該執行個體為止。

若要覆寫特定執行個體上的參數值，例如，如果範本包含的參數值為 stageprod，您可以將該參數的值覆寫為beta或gamma。

主題

- 動作類型
- 組態參數
- Input artifacts (輸入成品)
- 輸出成品
- 輸出變數

- 動作組態範例

動作類型

- 類別：Deploy
- 擁有者：AWS
- 提供者：CloudFormationStackInstances
- 版本：1

組態參數

StackSetName

必要：是

要與堆疊集相關聯的名稱。此名稱在建立該名稱的區域中必須是唯一的。

名稱只能包含英數字元和連字號字元。它必須以字母字元開頭，且不得超過 128 個字元。

DeploymentTargets

必要：否

> **Note**
> 對於 SERVICE_MANAGED 權限模型，您可以為部署目標提供組織根 ID 或組織單位識別碼。對於 SELF_MANAGED 權限模型，您只能提供帳戶。

> **Note**
> 選取此參數時，您還必須選取「區域」(Region)。

應在其中建立/更新堆疊集執行個體的AWS帳戶或組織單位 ID 清單。

- 帳戶：

  您可以將帳戶提供為常值清單或檔案路徑：

  - 常值：以速記語法格式輸入參數account_ID,account_ID，如下列範例所示。

```
111111222222,333333444444
```

- 檔案路徑：包含應在其中建立/更新堆疊集實例的AWS帳戶清單的檔案位置，並以格式輸入。InputArtifactName::AccountsFileName如果您使用檔案路徑來指定帳戶或 OrganizationalUnitIds，檔案格式必須為 JSON，如下列範例所示。

  ```
  SourceArtifact::accounts.txt
  ```

  下列範例顯示的檔案內容accounts.txt：

  ```
  [
      "111111222222"
  ]
  ```

  下列範例顯示列出多個帳號accounts.txt時的檔案內容：

  ```
  [
      "111111222222","333333444444"
  ]
  ```

- OrganizationalUnitIds:

  > **Note**
  >
  > 此參數對於「服務 _ 託管」權限模型是可選的，不適用於 SELF_MANAGED 權限模型。如果您選取，請勿使用此選項OrganizationsAutoDeployment。

  要在其中更新關聯堆疊執行個體的AWS組織單位。

  您可以提供組織單位 ID 做為常值清單或檔案路徑。

  - 常值：輸入以逗號分隔的字串陣列，如下列範例所示。

    ```
    ou-examplerootid111-exampleouid111,ou-examplerootid222-exampleouid222
    ```

  - 檔案路徑：包含要 OrganizationalUnitIds 在其中建立或更新堆疊集實例的清單的檔案位置。如果您使用檔案路徑來指定帳戶或 OrganizationalUnitIds，檔案格式必須為 JSON，如下列範例所示。

以格式輸入檔案的路徑InputArtifactName::OrganizationalUnitIdsFileName。

```
SourceArtifact::OU-IDs.txt
```

下列範例顯示的檔案內容OU-IDs.txt：

```
[
    "ou-examplerootid111-exampleouid111","ou-examplerootid222-exampleouid222"
]
```

區域

必要：是

> **Note**
>
> 選取此參數時，您也必須選取DeploymentTargets。

建立或更新堆疊集執行個體的AWS區域清單。區域會依照輸入的順序進行更新。

以格式輸入有效「AWS區域」清單：Region1,Region2，如下列範例所示。

```
us-west-2,us-east-1
```

ParameterOverrides

必要：否

您要在選取的堆疊例證中覆寫的堆疊集合參數清單。覆寫的參數值會套用至指定帳戶和區域中的所有堆疊執行個體。

您可以將參數提供為常值清單或檔案路徑：

- 您可以使用下列速記語法格式輸入參數：.
  ParameterKey=string,ParameterValue=string,UsePreviousValue=boolean,ResolvedV
  ParameterKey=string,ParameterValue=string,UsePreviousValue=boolean,ResolvedV
  如需這些資料類型的詳細資訊，請參閱模板參數數據類型。

  下列範例會示範以 my-值命名BucketName的參數bucket。

```
ParameterKey=BucketName,ParameterValue=my-bucket
```

下列範例顯示具有多個參數的項目。

```
ParameterKey=BucketName,ParameterValue=my-bucket
ParameterKey=Asset1,ParameterValue=true
ParameterKey=Asset2,ParameterValue=true
```

- 您可以輸入包含以格式輸入的範本參數取代清單的檔案位置`InputArtifactName::ParameterOverridessFileName`，如下列範例所示。

```
SourceArtifact::parameter-overrides.txt
```

下列範例顯示的檔案內容`parameter-overrides.txt`。

```
[
    {
        "ParameterKey": "KeyName",
        "ParameterValue": "true"
    },
    {
        "ParameterKey": "KeyName",
        "ParameterValue": "true"
    }
]
```

FailureTolerancePercentage

必要：否

在AWS CloudFormation停止該區域的作業之前，此堆疊作業可能會失敗的每個區域的帳戶百分比。如果「區域」中的作業已停止，則AWS CloudFormation不會在後續「區域」中嘗試該作業。根據指定的百分比計算帳戶數目時，AWS CloudFormation 會無條件捨去到下一個整數。

MaxConcurrentPercentage

必要：否

一次執行此作業的帳戶百分比上限。根據指定的百分比計算帳戶數目時，AWS CloudFormation 會無條件捨去到下一個整數。如果向下舍入會導致零，則將數字AWS CloudFormation設置為一。雖然您指定了最大值，但對於大型部署，由於服務節流，並行處理的實際帳戶數量可能會降低。

RegionConcurrencyType

必要：否

您可以設定區域並行部署參數，指定堆疊集應AWS 區域依序或 parallel 部署。如果指定區域 parallel 部署多AWS 區域個堆疊，可以加快整體部署時間。

- 並行：只要區域的部署失敗不超過指定的故障容忍，就會同時執行堆疊集合部署。
- 循序：只要區域的部署失敗不超過指定的故障容忍，就會一次執行一個堆疊集部署。順序部署是預設選項。

ConcurrencyMode

必要：否

並行模式可讓您選擇並行層級在堆疊集合作業期間的行為方式，無論是具有嚴格或軟式失敗容錯。嚴格容錯能力會降低部署速度，由於每次故障會使並行值減少，因此堆疊集操作會發生故障。軟性容錯能力會優先考慮部署速度，同時仍然利用 AWS CloudFormation 安全功能。

- STRICT_FAILURE_TOLERANCE：此選項會動態降低並行層次，以確保失敗的帳戶數目永遠不會超過特定的失敗容錯。這是預設行為。
- SOFT_FAILURE_TOLERANCE：此選項將失敗容忍與實際並行分離。這可讓堆疊集合作業在設定的並行層級執行，而不論失敗次數為何。

Input artifacts (輸入成品)

CloudFormationStackInstances可以包含列出部署目標和參數的人工因素。

- 人工因素數目：0 to 2
- 描述：作為輸入，堆疊集動作可選擇性地接受人工因素以達到下列目的：
  - 提供要使用的參數檔案。(請參閱 ParameterOverrides 參數。)
  - 提供要使用的目標帳戶檔案。(請參閱 DeploymentTargets 參數。)

輸出成品

- 人工因素數目：0

- 描述：輸出人工因素不適用於此動作類型。

輸出變數

設定時，此動作會產生變數，供管道中的下游動作的動作組態所參考。您可以設定動作的命名空間，讓這些變數可供下游動作的組態使用。

- StackSetId：堆疊集合的識別碼。
- OperationId：堆疊集合作業的識別碼。

如需詳細資訊，請參閱Variables。

動作組態範例

下列範例顯示動作的動CloudFormationStackInstances作組態。

自我管理權限模型的範例

下列範例顯示輸入的部署目標為 AWS 帳戶 ID 的CloudFormationStackInstances動作111111222222。

YAML

```
Name: my-instances
ActionTypeId:
  Category: Deploy
  Owner: AWS
  Provider: CloudFormationStackInstances
  Version: '1'
RunOrder: 2
Configuration:
  DeploymentTargets: '111111222222'
  Regions: 'us-east-1,us-east-2,us-west-1,us-west-2'
  StackSetName: my-stackset
OutputArtifacts: []
InputArtifacts:
  - Name: SourceArtifact
Region: us-west-2
```

JSON

```
{
    "Name": "my-instances",
    "ActionTypeId": {
        "Category": "Deploy",
        "Owner": "AWS",
        "Provider": "CloudFormationStackInstances",
        "Version": "1"
    },
    "RunOrder": 2,
    "Configuration": {
        "DeploymentTargets": "111111222222",
        "Regions": "us-east-1,us-east-2,us-west-1,us-west-2",
        "StackSetName": "my-stackset"
    },
    "OutputArtifacts": [],
    "InputArtifacts": [
        {
            "Name": "SourceArtifact"
        }
    ],
    "Region": "us-west-2"
}
```

服務管理權限模型的範例

下列範例顯示服務管理權限模型的CloudFormationStackInstances動作，其中部署目標是組 Organ AWS izations 組織單位 ID `ou-1111-1example`。

YAML

```
Name: Instances
ActionTypeId:
  Category: Deploy
  Owner: AWS
  Provider: CloudFormationStackInstances
  Version: '1'
RunOrder: 2
Configuration:
  DeploymentTargets: ou-1111-1example
  Regions: us-east-1
```

```
  StackSetName: my-stackset
OutputArtifacts: []
InputArtifacts:
  - Name: SourceArtifact
Region: eu-central-1
```

JSON

```
{
    "Name": "Instances",
    "ActionTypeId": {
        "Category": "Deploy",
        "Owner": "AWS",
        "Provider": "CloudFormationStackInstances",
        "Version": "1"
    },
    "RunOrder": 2,
    "Configuration": {
        "DeploymentTargets": "ou-1111-1example",
        "Regions": "us-east-1",
        "StackSetName": "my-stackset"
    },
    "OutputArtifacts": [],
    "InputArtifacts": [
        {
            "Name": "SourceArtifact"
        }
    ],
    "Region": "eu-central-1"
}
```

堆疊集作業的權限模型

由於跨多個帳戶AWS CloudFormation StackSets 執行作業，因此您必須先在這些帳戶中定義必要的權限，才能建立堆疊集。您可以透過自我管理的權限或服務管理的權限來定義權限。

使用自我管理許可，您可以建立兩個所需的 IAM 角色 StackSets -一個管理員角色，例如 AWSCloudFormationStackSetAdministrationRole 在您定義堆疊集的帳戶中，以及執行角色，例如部署堆疊集執行個體的每個帳戶 AWSCloudFormationStackSetExecutionRole 中的執行角色。使用此許可模型， StackSets 可以部署到使用者有權建立 IAM 角色的任何AWS帳戶。如需詳細資訊，請參閱AWS CloudFormation使用指南中的授與自我管理權限。

> **Note**
>
> 由於跨多個帳戶AWS CloudFormation StackSets 執行作業，因此您必須先在這些帳戶中定義必要的權限，才能建立堆疊集。

透過服務管理的權限，您可以將堆疊執行個體部署到由 Organ AWS izations 管理的帳戶。使用此許可模型，您不必建立必要的 IAM 角色，因為 StackSets 會代表您建立 IAM 角色。使用此模型，您也可以對 future 新增至組織的帳戶啟用自動部署。請參閱《AWS CloudFormation使用指南》中的「啟用 Organ AWS izations 信任的存取」。

模板參數數據類型

在堆棧集操作中使用的模板參數包括以下數據類型。如需詳細資訊，請參閱DescribeStackSet。

ParameterKey

- 描述：與參數相關聯的鍵。如果您沒有為特定參數指定索引鍵和值，則AWS CloudFormation會使用範本中指定的預設值。
- 範例：

```
"ParameterKey=BucketName,ParameterValue=my-bucket"
```

ParameterValue

- 描述：與參數相關聯的輸入值。
- 範例：

```
"ParameterKey=BucketName,ParameterValue=my-bucket"
```

UsePreviousValue

- 在堆疊更新期間，請使用堆疊用於指定參數鍵的現有參數值。如果您指定`true`，請勿指定參數值。
- 範例：

```
"ParameterKey=Asset1,UsePreviousValue=true"
```

每個堆疊集都有一個範本和一組範本參數。當您更新範本或範本參數時，會更新整個參數集的參數。然後，所有執行個體狀態都會設為「過時」，直到變更部署到該執行個體為止。

若要覆寫特定執行個體上的參數值，例如，如果範本包含的參數值為 `stageprod`，您可以將該參數的值覆寫為`beta`或`gamma`。

另請參閱

以下相關資源可協助您使用此動作。

- 參數類型 — AWS CloudFormation使用者指南中的本參考章節提供範 CloudFormation 本參數的更多說明和範例。
- 最佳做法 — 如需部署堆疊集的最佳作法的詳細資訊，請參閱《AWS CloudFormation使用者指南》https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/stacksets-bestpractices.html中的〈〉。
- AWS CloudFormationAPI 參考 — 您可以參考 AWS CloudFormationAPI 參考中的下列 CloudFormation 動作，以取得有關堆疊集操作中使用之參數的詳細資訊：
  - 此動CreateStackSet作會建立堆疊集。
  - 此UpdateStackSet動作會更新指定帳戶和區域中的堆疊集和相關聯的堆疊執行個體。即使透過更新堆疊集建立的堆疊集合作業失敗 (完全或部分、低於或高於指定的失敗容許)，堆疊集也會以這些變更進行更新。指定堆疊集的後續 CreateStackInstances 呼叫會使用更新的堆疊集。
  - 此CreateStackInstances動作會針對自我管理的權限模型上所有指定帳戶內的所有指定區域，或在服務管理權限模型上的所有指定部署目標內建立堆疊執行個體。您可以覆寫由此動作建立之執行個體的參數。如果實例已存在，則使用相同的輸入參數進行 CreateStackInstances 調 UpdateStackInstances 用。使用此動作建立執行個體時，不會變更其他堆疊執行個體的狀態。
  - 此UpdateStackInstances動作會使堆疊執行個體在自我管理的權限模型上，或是服務管理權限模型上所有指定帳戶內所有指定區域的堆疊設定為最新狀態。您可以覆寫由此動作更新之執行個體的參數。當您使用此動作更新執行個體的子集時，不會變更其他堆疊執行個體的狀態。
  - 該DescribeStackSetOperation動作返回指定堆棧集操作的描述。
  - 該DescribeStackSet操作返回指定堆棧集的描述。

AWS CodeBuild

可讓您執行建置和測試做為您管道的一部分。當您執行 CodeBuild 建置或測試動作時，建置規格中指定的命令會在 CodeBuild 容器內執行。指定為 CodeBuild 動作輸入成品的所有成品，都可以在執行命令的容器內使用。 CodeBuild 可以提供構建或測試操作。如需詳細資訊，請參閱 AWS CodeBuild 使用者指南。

當您使用主控台中的 CodePipeline 精靈建立組建專案時，組 CodeBuild 建專案會顯示來源提供者為 CodePipeline。當您在 CodeBuild主控台中建立組建專案時，您無法指定 CodePipeline 為來源提供者，但是將建置動作新增至管線會調整 CodeBuild 主控台中的來源。如需詳細資訊，請參閱《AWS CodeBuild API 參考》 中的 ProjectSource。

主題

動作類型

- 類別：`Build` 或 `Test`
- 擁有者：`AWS`
- 提供者：`CodeBuild`
- 版本：`1`

組態參數

ProjectName

必要：是

`ProjectName`是中建置專案的名稱 CodeBuild。

PrimarySource

必要：有條件

`PrimarySource`參數的值必須是動作的其中一個輸入加工品的名稱。 CodeBuild 查找構建規範文件，並在包含此成品的解壓縮版本的目錄中運行 build spec 命令。

如果針對 CodeBuild 動作指定了多個輸入成品，則此參數為必要。動作只有一個來源成品時，則 `PrimarySource` 成品會預設為該成品。

BatchEnabled

必要：否

`BatchEnabled`參數的 Boolean 值允許動作在相同的組建執行中執行多個組建。

啟用此選項時，即可使用此`CombineArtifacts`選項。

如需啟用批次建置的管道範例，請參閱[與批次建置CodePipeline 整合 CodeBuild 和批次建置](#)。

CombineArtifacts

必要：否

`CombineArtifacts`參數的 Boolean 值會將批次組建中的所有建置加工品合併到建置動作的單一成品檔案中。

若要使用此選項，必須啟用`BatchEnabled`參數。

EnvironmentVariables

必要：否

此參數的值用於為管線中的 CodeBuild 動作設定環境變數。`EnvironmentVariables` 參數的值採用環境變數物件的 JSON 陣列格式。請參閱 [動作宣告 (CodeBuild 範例)](#) 中的範例參數。

每個物件都有三個部分，而且全都是字串：

- `name`：環境變數的名稱或索引鍵。
- `value`：環境變數的值。使用`PARAMETER_STORE`或`SECRETS_MANAGER`類型時，此值必須是您已經儲存在AWS Systems Manager 參數存放區中的參數名稱，或是您已經儲存在AWS Secrets Manager 中的機密名稱。

> **Note**
>
> 我們強烈建議使用環境變數來儲存敏感值，尤其是AWS憑證。使用 CodeBuild 主控台或 AWS CLI 時，環境變數會以純文字顯示。對於敏感值，建議您改用 `SECRETS_MANAGER` 類型。

- `type`：(選擇性) 環境變數的類型。有效值為 `PARAMETER_STORE`、`SECRETS_MANAGER` 或 `PLAINTEXT`。未指定時，則將預設為 `PLAINTEXT`。

> **Note**
>
> 當您`type`針對環境變數組態輸入`name`、`value`、和時，特別是當環境變數包含 CodePipeline 輸出變數語法時，請勿超過組態值欄位 1000 個字元的限制。如果超過此限制，系統就會傳回驗證錯誤。

如需詳細資訊，請參閱 EnvironmentVariable中的，請AWS CodeBuild參閱中的。如需具有可解析為 GitHub 分支名稱之環境變數的範例 CodeBuild 動作，請參閱範例：搭配 CodeBuild 環境 BranchName變數使用變數。

Input artifacts (輸入成品)

- 人工因素數目：`1 to 5`
- 描述： CodeBuild 尋找組建規格檔案，並從主要來源加工品的目錄執行建置規格命令。為動作指定多個輸入來源時，必須使用中的 CodeBuild 動作組態參數來設定`PrimarySource`此人工因素 CodePipeline。

  每個輸入成品都會擷取到自己的目錄，其位置會存放在環境變數中。主要來源成品的目錄可透過 `$CODEBUILD_SRC_DIR` 使用。所有其他輸入成品的目錄可透過 `$CODEBUILD_SRC_DIR_yourInputArtifactName` 使用。

> **Note**
>
> CodeBuild 專案中設定的成品會成為管線中 CodeBuild 動作所使用的輸入成品。

輸出成品

- 人工因素數目：`0 to 5`
- 描述：這些可用於使 CodeBuild 建構規格檔案中定義的加工品可供管線中的後續動作使用。僅定義一個輸出成品時，可以直接在建置規格檔案的 `artifacts` 區段下定義此成品。指定多個輸出成品時，所有參照的成品都必須在建置規格檔案中定義為次要成品。中的輸出加工品名稱 CodePipeline 必須與建構規格檔案中的人工因素識別元相符。

> **Note**
>
> 在 CodeBuild 專案中設定的成品會成為管線動作中的 CodePipeline 輸入成品。

如果針對批次建置選取`CombineArtifacts`參數，則輸出人工因素位置會包含在相同執行中執行之多個組建的合併成品。

輸出變數

此動作將建置過程中匯出的所有環境變數產生為變數。如需如何匯出環境變數的詳細資訊，請參閱 AWS CodeBuildAPI 指南 EnvironmentVariable中的。

如需有關在中使用 CodeBuild 環境變數的詳細資訊 CodePipeline，請參閱中的範例CodeBuild 動作輸出變數。如需可在中使用的環境變數清單 CodeBuild，請參閱《使AWS CodeBuild用指南》中的建置環境中的環境變數。

動作宣告 (CodeBuild 範例)

YAML

```
Name: Build
Actions:
  - Name: PackageExport
    ActionTypeId:
      Category: Build
      Owner: AWS
      Provider: CodeBuild
      Version: '1'
    RunOrder: 1
    Configuration:
      BatchEnabled: 'true'
      CombineArtifacts: 'true'
      ProjectName: my-build-project
      PrimarySource: MyApplicationSource1
      EnvironmentVariables:
 '[{"name":"TEST_VARIABLE","value":"TEST_VALUE","type":"PLAINTEXT"},
{"name":"ParamStoreTest","value":"PARAMETER_NAME","type":"PARAMETER_STORE"}]'
    OutputArtifacts:
      - Name: MyPipeline-BuildArtifact
```

```
    InputArtifacts:
      - Name: MyApplicationSource1
      - Name: MyApplicationSource2
```

JSON

```
{
    "Name": "Build",
    "Actions": [
        {
            "Name": "PackageExport",
            "ActionTypeId": {
                "Category": "Build",
                "Owner": "AWS",
                "Provider": "CodeBuild",
                "Version": "1"
            },
            "RunOrder": 1,
            "Configuration": {
                "BatchEnabled": "true",
                "CombineArtifacts": "true",
                "ProjectName": "my-build-project",
                "PrimarySource": "MyApplicationSource1",
                "EnvironmentVariables": "[{\"name\":\"TEST_VARIABLE\",\"value\":
\"TEST_VALUE\",\"type\":\"PLAINTEXT\"},{\"name\":\"ParamStoreTest\",\"value\":
\"PARAMETER_NAME\",\"type\":\"PARAMETER_STORE\"}]"
            },
            "OutputArtifacts": [
                {
                    "Name": "MyPipeline-BuildArtifact"
                }
            ],
            "InputArtifacts": [
                {
                    "Name": "MyApplicationSource1"
                },
                {
                    "Name": "MyApplicationSource2"
                }
            ]
        }
    ]
```

```
}
```

另請參閱

以下相關資源可協助您使用此動作。

- AWS CodeBuild使用者指南 — 如需具有 CodeBuild 動作的範例管道，請參閱 CodePipeline 搭配使用 CodeBuild 來測試程式碼和執行組建。有關具有多個輸入和輸出成 CodeBuild 品的專案範例，請參閱CodePipeline整合與 CodeBuild 多個輸入來源及輸出人工因素範例和多個輸入來源和輸出人工因素範例。
- 教程：創建一個管道，用於構建和測試您的 Android 應用程序 AWS Device Farm— 本教程提供了一個示例構建規範文件和示例應用程序，用於創建具有構建和測試 Android 應用程序的 GitHub 源代碼的 CodeBuild 管道AWS Device Farm。
- 建構的規格參考 CodeBuild — 此參考主題提供瞭解 CodeBuild 建構規格檔案的定義與範例。如需可在中使用的環境變數清單 CodeBuild，請參閱《使AWS CodeBuild用指南》中的建置環境中的環境變數。

CodeCommit

在配置的 CodeCommit 存儲庫和分支上進行新的提交時啟動管道。

如果您使用控制台來建立或編輯管道，請 CodePipeline 建立一個 CodeCommit CloudWatch 事件規則，以便在存放庫中發生變更時啟動管道。

您必須先建立存 CodeCommit 放庫，然後才能透過 CodeCommit 動作連接管道。

偵測到程式碼變更之後，您有下列選擇將程式碼傳遞給後續動作：

- 預設 — 設定 CodeCommit 來源動作以輸出含有提交淺層副本的 ZIP 檔案。
- 完整複製 — 設定來源動作，將 Git URL 參考輸出至儲存庫，以供後續動作使用。

  目前，Git URL 參考只能由下游 CodeBuild 動作用來複製存放庫和相關聯的 Git 中繼資料。嘗試將 Git URL 參考傳遞給非CodeBuild 動作會導致錯誤。

主題

- 動作類型

動作類型

- 類別：`Source`
- 擁有者：`AWS`
- 提供者：`CodeCommit`
- 版本：`1`

組態參數

RepositoryName

必要：是

要偵測來源變更的儲存庫名稱。

BranchName

必要：是

要偵測來源變更的分支名稱。

PollForSourceChanges

必要：否

`PollForSourceChanges`控制是否 CodePipeline 輪詢 CodeCommit存放庫中的來源變更。我們建議您改用 E CloudWatch vents 來偵測來源變更。如需有關配置 CloudWatch 事件的詳細資訊，請參閱移轉輪詢管線 (CodeCommit 來源) (CLI)或移轉輪詢管線 (CodeCommit 來源) (AWS CloudFormation範本)。

> **Important**
>
> 如果您打算配置 CloudWatch 事件規則，則必須將其設定`PollForSourceChanges`為以`false`避免重複的管線執行。

此參數的有效值：

- `true`：如果設置，則 CodePipeline 輪詢您的存儲庫以進行源更改。

  > **Note**
  >
  > 如果省略`PollForSourceChanges`，則 CodePipeline 預設會輪詢儲存庫以進行來源變更。此行為同於包含 `PollForSourceChanges` 且設定為 `true`。

- `false`：如果設定，則 CodePipeline 不會輪詢您的儲存庫是否有來源變更。如果您想要設定 CloudWatch 事件規則以偵測來源變更，請使用此設定。

OutputArtifactFormat

必要：否

輸出成品格式。值可以是`CODEBUILD_CLONE_REF`或`CODE_ZIP`。如果未指定，預設值為 `CODE_ZIP`。

> **Important**
>
> `CODEBUILD_CLONE_REF` 選項只能由 CodeBuild 下游動作使用。
> 如果您選擇此選項，則需要將`codecommit:GitPull`權限新增至您的 CodeBuild 服務角色，如中所示新增 CodeCommit來源動作的 CodeBuild GitClone 權限。您還需要將`codecommit:GetRepository`權限添加到您的 CodePipeline服務角色，如中所示將許可新增至 CodePipeline 服務角色。如需說明如何使用「完整複製」選項的教學課程，請參閱教學課程：搭 CodeCommit 配管線來源使用完整複製。

Input artifacts (輸入成品)

- 人工因素數目：`0`
- 描述：輸入成品不適用於此動作類型。

輸出成品

- 人工因素數目：1
- 描述：此動作的輸出成品是 ZIP 檔案，其中包含在指定為管道執行來源修訂的遞交上所設定的儲存庫和分支的內容。從存放庫產生的成品是 CodeCommit 動作的輸出成品。中的原始程式碼提交 ID 會顯示 CodePipeline 為觸發管線執行的來源修訂。

輸出變數

設定時，此動作會產生變數，供管道中的下游動作的動作組態所參考。即使此動作沒有命名空間，此動作產生的變數仍可視為輸出變數。您可以設定動作的命名空間，讓這些變數可供下游動作的組態使用。

如需詳細資訊，請參閱 Variables。

CommitId

觸發管線執行的 CodeCommit 提交 ID。遞交 ID 是遞交的完整 SHA。

CommitMessage

與觸發管道執行的遞交相關聯的描述訊息 (如果有的話)。

RepositoryName

執行觸發管線之提交的 CodeCommit 儲存庫名稱。

BranchName

進行來源變更之 CodeCommit 儲存庫的分支名稱。

AuthorDate

遞交的撰寫日期 (時間戳記格式)。

如需 Git 中的作者和遞交者之間差異的詳細信息，請參閱 Scott Chacon 和 Ben Straub 共同撰寫的 Pro Git 中的檢視提交的歷史記錄。

CommitterDate

遞交的遞交日期 (時間戳記格式)。

如需 Git 中的作者和遞交者之間差異的詳細信息，請參閱 Scott Chacon 和 Ben Straub 共同撰寫的 Pro Git 中的檢視提交的歷史記錄。

動作組態範例

預設輸出人工因素格式的範例

YAML

```
Actions:
  - OutputArtifacts:
      - Name: Artifact_MyWebsiteStack
    InputArtifacts: []
    Name: source
    Configuration:
      RepositoryName: MyWebsite
      BranchName: main
      PollForSourceChanges: 'false'
    RunOrder: 1
    ActionTypeId:
      Version: '1'
      Provider: CodeCommit
      Category: Source
      Owner: AWS
  Name: Source
```

JSON

```
{
    "Actions": [
        {
            "OutputArtifacts": [
                {
                    "Name": "Artifact_MyWebsiteStack"
                }
            ],
            "InputArtifacts": [],
            "Name": "source",
            "Configuration": {
                "RepositoryName": "MyWebsite",
                "BranchName": "main",
                "PollForSourceChanges": "false"
            },
            "RunOrder": 1,
            "ActionTypeId": {
                "Version": "1",
```

```
                "Provider": "CodeCommit",
                "Category": "Source",
                "Owner": "AWS"
            }
        }
    ],
    "Name": "Source"
},
```

完整複製輸出加工品格式的範例

YAML

```
name: Source
actionTypeId:
  category: Source
  owner: AWS
  provider: CodeCommit
  version: '1'
runOrder: 1
configuration:
  BranchName: main
  OutputArtifactFormat: CODEBUILD_CLONE_REF
  PollForSourceChanges: 'false'
  RepositoryName: MyWebsite
outputArtifacts:
  - name: SourceArtifact
inputArtifacts: []
region: us-west-2
namespace: SourceVariables
```

JSON

```
{
    "name": "Source",
    "actionTypeId": {
        "category": "Source",
        "owner": "AWS",
        "provider": "CodeCommit",
        "version": "1"
    },
    "runOrder": 1,
```

```
    "configuration": {
        "BranchName": "main",
        "OutputArtifactFormat": "CODEBUILD_CLONE_REF",
        "PollForSourceChanges": "false",
        "RepositoryName": "MyWebsite"
    },
    "outputArtifacts": [
        {
            "name": "SourceArtifact"
        }
    ],
    "inputArtifacts": [],
    "region": "us-west-2",
    "namespace": "SourceVariables"
}
```

另請參閱

以下相關資源可協助您使用此動作。

- 教學：建立範本管道 (CodeCommit 儲存庫)— 本教學課程提供範例應用程式規格檔案，以及範例 CodeDeploy 應用程式和部署群組。使用本教學建立管道，其中包含部署到 Amazon EC2 執行個體的 CodeCommit 來源。

AWS CodeDeploy

您可以使用AWS CodeDeploy操作將應用程序代碼部署到您的部署隊列。您的部署隊列可以由 Amazon EC2 執行個體、現場部署執行個體或兩者。

> **Note**
>
> 本參考主題介紹 CodeDeploy 的部署操作 CodePipeline ，其中部署平台是 Amazon EC2。有關 Amazon 彈性容器服務以在 CodePipeline 中部署藍/綠色部署操作的參考信息，請參閱Amazon Elastic Coner Coner Coner Coner Coner ConCodeDeploy。

主題

- 動作類型

動作類型

- 類別：`Deploy`
- 擁有者：`AWS`
- 提供者：`CodeDeploy`
- 版本：`1`

組態參數

ApplicationName

必要 是

您在 CodeDeploy 中創建的應用程序的名稱。

DeploymentGroupName

必要 是

您在 CodeDeploy 中創建的部署組。

Input artifacts (輸入成品)

- 成品數量： 1
- 描述：所以此 AppSpec 文件 CodeDeploy 用於確定：
  - 從 Amazon S3 或 GitHub 的應用程式修訂中安裝至執行個體的項目。
  - 為回應部署生命週期事件而執行的生命週期事件勾點。

  如需有關的詳細資訊 AppSpec 文件中，請參閱CodeDeploy AppSpec 檔案參考。

輸出成品

- 成品數量： 0
- 描述：輸出成品不適用於此動作類型。

動作宣告

YAML

```
Name: Deploy
Actions:
  - Name: Deploy
    ActionTypeId:
      Category: Deploy
      Owner: AWS
      Provider: CodeDeploy
      Version: '1'
    RunOrder: 1
    Configuration:
      ApplicationName: my-application
      DeploymentGroupName: my-deployment-group
    OutputArtifacts: []
    InputArtifacts:
      - Name: SourceArtifact
    Region: us-west-2
    Namespace: DeployVariables
```

JSON

```
{
    "Name": "Deploy",
    "Actions": [
        {
            "Name": "Deploy",
            "ActionTypeId": {
                "Category": "Deploy",
                "Owner": "AWS",
                "Provider": "CodeDeploy",
                "Version": "1"
            },
            "RunOrder": 1,
```

```
            "Configuration": {
                "ApplicationName": "my-application",
                "DeploymentGroupName": "my-deployment-group"
            },
            "OutputArtifacts": [],
            "InputArtifacts": [
                {
                    "Name": "SourceArtifact"
                }
            ],
            "Region": "us-west-2",
            "Namespace": "DeployVariables"
        }
    ]
},
```

另請參閱

以下相關資源可協助您使用此動作。

- 教學：建立簡易管道 (S3 儲存貯體)— 本教程將引導您完成源存儲桶、EC2 實例和 CodeDeploy 資源部署範例應用程式。然後，您可以使用 CodeDeploy 部署操作，該操作將 S3 存儲桶中維護的代碼部署到您的 Amazon EC2 實例。
- 教學：建立範本管道 (CodeCommit 儲存庫)— 此教學課程會逐步解說如何創建 CodeCommit 源存儲庫、EC2 實例和 CodeDeploy 資源部署範例應用程式。然後，您可以使用 CodeDeploy 部署操作，該操作從 CodeCommit 存儲庫設定至 Amazon EC2 執行個體。
- CodeDeploy AppSpec 檔案參考— 此參考章節位於AWS CodeDeploy使用者指南提供參考信息和示例 CodeDeploy AppSpec 檔案。

CodeStarSourceConnection 適用於 Bitbucket 雲端 GitHub、GitHub 企業伺服器、 GitLab .com 和 GitLab 自我管理動作

在第三方原始程式碼儲存庫上進行新的提交時，啟動管道。當手動執行管線或從來源提供者傳送 webhook 事件時，來源動作會擷取程式碼變更。

您可以將管道中的動作設定為使用允許您使用觸發程序啟動管道的 Git 組態。若要設定管線觸發器組態以篩選觸發器，請參閱中的詳細資訊篩選程式碼推送或提取要求的觸發程序。

> **Note**
>
> 亞太區域 (香港)、亞太區域 (海德拉巴)、亞太區域 (雅加達)、亞太區域 (墨爾本)、亞太區域 (大阪)、非洲 (開普敦)、中東 (巴林)、中東 (阿聯酋)、歐洲 (西班牙)、歐洲 (蘇黎世)、以色列 (特拉維夫) 或 AWS GovCloud (美國西部) 區域不提供此功能。若要參考其他可用動作，請參閱產品與服務整合 CodePipeline。如需在歐洲 (米蘭) 區域進行此動作的考量，請參閱中的附註CodeStarSourceConnection 適用於 Bitbucket 雲端 GitHub、 GitHub 企業伺服器、GitLab .com 和 GitLab 自我管理動作。

連線可以為您的 AWS 資源與下列第三方儲存庫建立關聯：

- Bitbucket 雲端 (透過主控台中的 Bitbucket 提供者選項或 CodePipeline CLI 中的`Bitbucket`提供者)

  > **Note**
  >
  > 您可以建立連至 Bitbucket Cloud 儲存庫的連線。不支援安裝式 Bitbucket 供應商類型，例如 Bitbucket 伺服器。

-

  > **Note**
  >
  > 如果您使用 Bitbucket 工作區，則必須具有管理員存取權才能建立連線。

- GitHub 和 GitHub 企業雲（通過控制台中的 GitHub （版本 2）提供程序選項或 CodePipeline CLI 中的`GitHub`提供程序）

  > **Note**
  >
  > 如果您的存放庫位於 GitHub 組織中，您必須是組織擁有者才能建立連線。如果您使用的是不在組織中的存放庫，您必須是存放庫擁有者。

- GitHub 企業伺服器 (透過主控台中的「GitHub 企業伺服器提供者」選項或 CodePipeline CLI 中的`GitHub Enterprise Server`提供者)
- GitLab.com (透過主控台中的GitLab提供者選項或 CodePipeline CLI 中的`GitLab`提供者)

> **Note**
>
> 您可以建立與具有 Owner 角色之儲存庫的連線 GitLab，然後連線可與具有諸如資源的存放庫一起使用 CodePipeline。如果是群組中的儲存庫，您不需要為群組擁有者。

- GitLab (企業版或社群版) 的自我管理安裝 (透過主控台中的GitLab 自我管理提供者選項或 CodePipeline CLI 中的`GitLabSelfManaged`提供者)

> **Note**
>
> 每個連接都支持您使用該提供程序的所有存儲庫。您只需要為每個提供者類型建立新連線。

連線可讓您的管道透過第三方供應商的安裝應用程式偵測來源變更。例如，Webhook 可用來訂閱 GitHub 事件類型，而且可以安裝在組織、存放庫或應用 GitHub 程式上。您的連接會在您的 GitHub 應用程序上安裝一個訂閱 GitHub 推送類型事件的存儲庫 webhook。

偵測到程式碼變更之後，您有下列選擇將程式碼傳遞給後續動作：

- 默認值：與其他現有 CodePipeline 源操作一樣，`CodeStarSourceConnection`可以輸出帶有提交的淺層副本的 ZIP 文件。
- 完全克隆：也`CodeStarSourceConnection`可以配置為輸出對存儲庫的 URL 引用以進行後續操作。

  目前，Git URL 參考只能由下游 CodeBuild 動作用來複製存放庫和相關聯的 Git 中繼資料。嘗試將 Git URL 參考傳遞給非CodeBuild 動作會導致錯誤。

CodePipeline 建立連線時，會提示您將連AWS接器安裝應用程式新增至第三方帳戶。您必須先建立第三方提供者帳戶和存放庫，才能透過`CodeStarSourceConnection`動作進行連線。

> **Note**
>
> 若要使用AWS CodeStar連線所需的權限，建立原則或將原則附加至您的角色，請參閱連線權限參考。視建立 CodePipeline 服務角色的時間而定，您可能需要更新其權限以支援AWS CodeStar連線。如需說明，請參閱將許可新增至 CodePipeline 服務角色。

> **Note**
>
> 若要使用歐洲區域 (米蘭) AWS 區域，您必須：
>
> 1. 安裝區域特定的應用程式
> 2. 啟用區域
>
> 此區域特定的應用程式支援歐洲 (米蘭) 區域中的連線。它會在第三方供應商網站上發佈，並且它會與支援其他區域連線的現有應用程式分開。透過安裝此應用程式，您授權第三方提供商僅使用該區域服務來共用您的資料，並且您可以透過解除安裝該應用程式來隨時撤銷許可。
> 除非您啟用區域，否則服務不會處理或儲存您的資料。啟用此區域，即表示您授予我們服務許可來處理和儲存您的資料。
> 即使未啟用該區域，如果仍已安裝區域特定的應用程式，第三方供應商仍然可以使用我們的服務來共用您的資料，因此請確保在停用該區域後解除安裝該應用程式。如需詳細資訊，請參閱啟用區域。

主題

動作類型

- 類別：`Source`
- 擁有者：`AWS`
- 提供者：`CodeStarSourceConnection`
- 版本：`1`

組態參數

ConnectionArn

必要：是

針對來源提供者設定和驗證的連線 ARN。

FullRepositoryId

必要：是

要偵測來源變更的儲存庫的擁有者和名稱。

範例：`some-user/my-repo`

> **Important**
>
> 您必須為FullRepositoryId值維持正確的大小寫。例如，如果您的使用者名稱是some-user且存放庫名稱為My-Repo，則建議的值FullRepositoryId為some-user/My-Repo。

BranchName

必要：是

要偵測來源變更的分支名稱。

OutputArtifactFormat

必要：否

指定輸出成品格式。可以是 CODEBUILD_CLONE_REF 或 CODE_ZIP。如果未指定，預設值為 CODE_ZIP。

> **Important**
>
> CODEBUILD_CLONE_REF 選項只能由 CodeBuild 下游動作使用。
> 如果您選擇此選項，則需要更新 CodeBuild 專案服務角色的權限，如中所示新增連線至 Bitbucket GitHub、 GitHub 企業伺服器或 GitLab .com 的 CodeBuild GitClone 權限。如需說明如何使用「完整複製」選項的教學課程，請參閱教學課程：搭 GitHub 配管線來源使用完整複製。

DetectChanges

必要：否

當在配置的存儲庫和分支上進行新的提交時，控制項會自動啟動管道。如果未指定，則默認值為`true`，默認情況下不顯示字段。此參數的有效值：

- `true`：在新的提交時 CodePipeline 自動啟動管道。
- `false`: CodePipeline 不會在新的提交時啟動管道。

Input artifacts (輸入成品)

- 人工因素數目：`0`
- 描述：輸入成品不適用於此動作類型。

輸出成品

- 人工因素數目：`1`
- 描述：從儲存庫產生的成品是 `CodeStarSourceConnection` 動作的輸出成品。中的原始程式碼提交 ID 會顯示 CodePipeline 為觸發管線執行的來源修訂。您可以在下列項目中設定此動作的輸出成品：
  - ZIP 檔案，其中包含在指定為管道執行來源修訂的遞交上所設定的儲存庫和分支的內容。
  - JSON 檔案，其中包含儲存庫的 URL 參考，讓下游動作可以直接執行 Git 命令。

> **Important**
>
> 此選項只能由 CodeBuild 下游動作使用。
> 如果您選擇此選項，則需要更新 CodeBuild 專案服務角色的權限，如中所示疑難排 CodePipeline。如需說明如何使用「完整複製」選項的教學課程，請參閱教學課程：搭 GitHub 配管線來源使用完整複製。

輸出變數

設定時，此動作會產生變數，供管道中的下游動作的動作組態所參考。即使此動作沒有命名空間，此動作產生的變數仍可視為輸出變數。您可以設定動作的命名空間，讓這些變數可供下游動作的組態使用。

如需詳細資訊，請參閱 Variables。

AuthorDate

遞交的撰寫日期 (時間戳記格式)。

BranchName

進行來源變更的 儲存庫分支的名稱。

CommitId

觸發管道執行的 遞交 ID。

CommitMessage

與觸發管道執行的遞交相關聯的描述訊息 (如果有的話)。

ConnectionArn

針對來源提供者設定和驗證的連線 ARN。

FullRepositoryName

進行遞交以觸發管道的 儲存庫的名稱。

動作宣告

在下列範例中，針對與 ARN `arn:aws:codestar-connections:region:account-id:connection/connection-id` 連線，輸出人工因素會設定`CODE_ZIP`為的預設 ZIP 格式。

YAML

```
Name: Source
Actions:
  - InputArtifacts: []
    ActionTypeId:
      Version: '1'
      Owner: AWS
      Category: Source
      Provider: CodeStarSourceConnection
    OutputArtifacts:
      - Name: SourceArtifact
    RunOrder: 1
    Configuration:
```

```
      ConnectionArn: "arn:aws:codestar-connections:region:account-
id:connection/connection-id"
      FullRepositoryId: "some-user/my-repo"
      BranchName: "main"
      OutputArtifactFormat: "CODE_ZIP"
    Name: ApplicationSource
```

JSON

```
{
    "Name": "Source",
    "Actions": [
        {
            "InputArtifacts": [],
            "ActionTypeId": {
                "Version": "1",
                "Owner": "AWS",
                "Category": "Source",
                "Provider": "CodeStarSourceConnection"
            },
            "OutputArtifacts": [
                {
                    "Name": "SourceArtifact"
                }
            ],
            "RunOrder": 1,
            "Configuration": {
                "ConnectionArn": "arn:aws:codestar-connections:region:account-
id:connection/connection-id",
                "FullRepositoryId": "some-user/my-repo",
                "BranchName": "main",
                "OutputArtifactFormat": "CODE_ZIP"
            },
            "Name": "ApplicationSource"
        }
    ]
},
```

安裝安裝應用程式並建立連線

第一次使用控制台向第三方存儲庫添加新連接時，必須授權存儲庫的 CodePipeline 訪問權限。您可以選擇或建立安裝應用程式，以協助連線至您用來建立第三方程式碼儲存庫的帳戶。

使用AWS CLI或AWS CloudFormation範本時，您必須提供已經通過安裝交握之連線的連線 ARN。否則不會觸發管道。

> **Note**
>
> 對於CodeStarSourceConnection來源動作，您不需要設定 webhook 或預設為輪詢。連線動作會為您管理來源變更偵測。

另請參閱

以下相關資源可協助您使用此動作。

- AWS::CodeStarConnections::Connection—「AWS CodeStar 連線」資源的AWS CloudFormation範本參考提供範AWS CloudFormation本中連線的參數和範例。
- AWS CodeStar連線 API 參考 — AWS CodeStar 連線 API 參考提供可用連線動作的參考資訊。
- 若要檢視使用連線支援的來源動作建立管線的步驟，請參閱下列內容：
  - 若是 Bitbucket 雲端，請使用主控台中的「Bitbucket」選項或 CLI 中的CodestarSourceConnection動作。請參閱 比特桶雲連接。
  - 對於 GitHub 和 GitHub 企業雲，請使用主控台中的GitHub提供者選項或 CLI 中的CodestarSourceConnection動作。請參閱 GitHub 連接。
  - 對於 GitHub 企業伺服器，請使用主控台中的「GitHub 企業伺服器提供者」選項或 CLI 中的CodestarSourceConnection動作。請參閱 GitHub 企業伺服器連線。
  - 對於 GitLab .com，請使用主控台中的GitLab提供者選項，或使用 CLI 中GitLab提供者的CodestarSourceConnection動作。請參閱 GitLab.com 連線。
- 若要檢視使用 Bitbucket 來源和 CodeBuild 動作建立管線的入門教學課程，請參閱開始使用連線。
- 如需說明如何連線至 GitHub 存放庫，以及如何搭配下游 CodeBuild 動作使用「完整複製」選項的教學課程，請參閱教學課程：搭 GitHub 配管線來源使用完整複製。

AWS Device Farm

在管道中，您可以設定用AWS Device Farm來在裝置上執行和測試應用程式的測試動作。Device Farm 使用裝置的測試集區和測試架構來測試特定裝置上的應用程式。如需 [Device Farm 服器陣列] 動作所支援之測試架構類型的相關資訊，請參閱使用 AWS Device Farm 中的測試類型。

主題

動作類型

- 類別：Test
- 擁有者：AWS
- 提供者：DeviceFarm
- 版本：1

組態參數

AppType

必要：是

您正在測試的作業系統和應用程式類型。以下是有效值的清單：

- iOS
- Android
- Web

ProjectId

必要：是

Device Farm 專案 ID。

若要尋找您的專案 ID，請在 Device Farm 主控台中選擇您的專案。在瀏覽器中，複製新專案的 URL。URL 包含專案 ID。專案 ID 是之後 URL 中的值projects/。在下列範例中，專案 ID 為eec4905f-98f8-40aa-9afc-4c1cfexample。

```
https://<region-URL>/devicefarm/home?region=us-west-2#/projects/
eec4905f-98f8-40aa-9afc-4c1cfexample/runs
```

应用

必要：是

輸入成品中應用程式檔案的名稱和位置。例如：`s3-ios-test-1.ipa`

TestSpec

條件式：是

測試規格定義檔案在輸入成品中的位置。這是自定義模式測試所必需的。

DevicePoolArn

必要：是

Device Farm 裝置集區 ARN。

若要取得專案的可用裝置集區 ARN (包括常用裝置的 ARN)，請使用 AWS CLI 輸入下列命令：

```
aws devicefarm list-device-pools --arn arn:aws:devicefarm:us-
west-2:account_ID:project:project_ID
```

TestType

必要：是

指定測試支援的測試架構。以下是有效值的清單`TestType`：

- 爪哇胡尼特
- Appium_ 爪哇測試
- APPIUM 節點
- 蘋果紅寶石
- 蟒蛇
- 阿皮亚姆网站 JA_JUnit
- 蘋果網站測試
- 应用网节点
- 蘋果紅寶石
- 蟒蛇網
- 內建模糊
- INSTRUMENTATION

- XCTEST
- XCTEST_UI

> **Note**
>
> WEB_PERFORMANCE_PROFILE、REMOTE_ACCESS_RECORD和中 CodePipeline的動作不支援下列測試類型REMOTE_ACCESS_REPLAY。

如需 Device Farm 測試類型的相關資訊，請參閱使用 AWS Device Farm 中的測試

RadioBluetoothEnabled

必要：否

Boolean 值，指出是否在測試開始時啟用藍牙。

RecordAppPerformanceData

必要：否

Boolean 值，指出測試期間是否記錄裝置效能資料，例如 CPU、FPS 和記憶體效能。

RecordVideo

必要：否

Boolean 值，指出是否在測試期間錄製視訊。

RadioWifiEnabled

必要：否

布林值，指出是否在測試開始時啟用 Wi-Fi。

RadioNfcEnabled

必要：否

布林值，指出是否在測試開始時啟用 NFC。

RadioGpsEnabled

必要：否

布林值，指出是否在測試開始時啟用 GPS。

測試

必要：否

來源位置中測試定義檔案的名稱和路徑。路徑為相對於您測試輸入成品根的相對路徑。

FuzzEventCount

必要：否

模糊測試要執行的使用者介面事件數目，介於 1 到 10,000 之間。

FuzzEventThrottle

必要：否

模糊測試在執行下一個使用者介面事件之前等待的毫秒數 (介於 1 到 1,000 之間)。

FuzzRandomizerSeed

必要：否

模糊測試的種子，用於隨機化使用者介面事件。在後續的模糊測試中使用相同的數字會產生相同的事件序列。

CustomHostMachineArtifacts

必要：否

主機上將儲存自訂加工品的位置。

CustomDeviceArtifacts

必要：否

裝置上將儲存自訂成品的位置。

UnmeteredDevicesOnly

必要：否

Boolean 值，指出在此步驟中執行測試時，是否只使用未計量裝置。

JobTimeoutMinutes

必要：否

在超時之前，每個設備將執行測試運行的分鐘數。

緯度

必要：否

裝置的緯度，以地理座標系度表示。

經度

必要：否

裝置的經度，以地理座標系度表示。

Input artifacts (輸入成品)

- 人工因素數目：1
- 描述：要提供給測試動作的一組成品。Device Farm 會尋找要使用的建置應用程式和測試定義。

輸出成品

- 成品數量：0
- 描述：輸出人工因素不適用於此動作類型。

動作宣告

YAML

```
Name: Test
Actions:
  - Name: TestDeviceFarm
    ActionTypeId: null
    category: Test
    owner: AWS
    provider: DeviceFarm
    version: '1'
RunOrder: 1
Configuration:
  App: s3-ios-test-1.ipa
  AppType: iOS
  DevicePoolArn: >-
    arn:aws:devicefarm:us-west-2::devicepool:0EXAMPLE-d7d7-48a5-ba5c-b33d66efa1f5
```

```
  ProjectId: eec4905f-98f8-40aa-9afc-4c1cfEXAMPLE
  TestType: APPIUM_PYTHON
  TestSpec: example-spec.yml
OutputArtifacts: []
InputArtifacts:
  - Name: SourceArtifact
Region: us-west-2
```

JSON

```
{
    "Name": "Test",
    "Actions": [
        {
            "Name": "TestDeviceFarm",
            "ActionTypeId": null,
            "category": "Test",
            "owner": "AWS",
            "provider": "DeviceFarm",
            "version": "1"
        }
    ],
    "RunOrder": 1,
    "Configuration": {
        "App": "s3-ios-test-1.ipa",
        "AppType": "iOS",
        "DevicePoolArn": "arn:aws:devicefarm:us-west-2::devicepool:0EXAMPLE-
d7d7-48a5-ba5c-b33d66efa1f5",
        "ProjectId": "eec4905f-98f8-40aa-9afc-4c1cfEXAMPLE",
        "TestType": "APPIUM_PYTHON",
        "TestSpec": "example-spec.yml"
    },
    "OutputArtifacts": [],
    "InputArtifacts": [
        {
            "Name": "SourceArtifact"
        }
    ],
    "Region": "us-west-2"
},
```

另請參閱

以下相關資源可協助您使用此動作。

- 在裝置伺服器陣列中使用測試類型 — 裝置伺服器陣列開發人員指南中的這個參考章節提供有關裝置伺服器 Device Farm 列支援的 Android、iOS 和 Web 應用程式測試架構的詳細說明。
- Device Farm 中的動作 — 裝 Dev ice Farm API 參考中的 API 呼叫和參數可協助您使用 Device Farm 專案。
- 教程：創建一個管道，用於構建和測試您的 Android 應用程序 AWS Device Farm— 本教程提供了一個示例構建規範文件和示例應用程序，用於創建具有 GitHub 源代碼的管道，該管道可使用和 Device Farm 構建 CodeBuild 和測試 Android 應用程序。
- 教學課程：建立測試 iOS 應用程式的管道 AWS Device Farm— 本教學提供範例應用程式，可使用 Amazon S3 來源建立管道，以測試使用裝置伺服器陣列測試建置的 iOS 應用程式。

AWS Lambda

可讓您在管道中以動作的形式執行 Lambda 函數。使用事件物件作為此函數的輸入，函數具有存取動作組態、輸入成品位置、輸出成品位置，以及其他存取成品所需的資訊。如需傳遞至 Lambda 叫用函數的範例事件，請參閱JSON 事件範例。Lambda 函數實作時，必須呼叫 `PutJobSuccessResult API` 或 `PutJobFailureResult API`。否則，執行此動作會停止回應，直到動作逾時為止。如果您指定動作的輸出成品，則必須將它們上傳到 S3 儲存貯體，做為函數實作的一部分。

> **Important**
>
> 請勿記錄 CodePipeline 傳送至 Lambda 的 JSON 事件，因為這可能會導致使用者認證記錄在 CloudWatch 記錄中。該 CodePipeline 角色使用 JSON 事件將臨時登入資料傳遞給`artifactCredentials`欄位中的 Lambda。如需範例事件，請參閱JSON 事件範例。

動作類型

- 類別：`Invoke`
- 擁有者：`AWS`
- 提供者：`Lambda`
- 版本：`1`

組態參數

FunctionName

必要：是

`FunctionName`是在 Lambda 中創建的函數的名稱。

UserParameters

必要：否

可由 Lambda 函數當做輸入處理的字串。

Input artifacts (輸入成品)

- 成品數量：`0 to 5`
- 描述：要提供給 Lambda 函數使用的一組成品。

輸出成品

- 成品數量：`0 to 5`
- 說明：Lambda 函數產生為輸出的一組成品。

輸出變數

此動作會產生 PutJobSuccessResult API 要求`outputVariables`區段中包含的所有索引鍵值組做為變數。

如需中變數的詳細資訊 CodePipeline，請參閱「Variables」。

動作組態範例

YAML

```
Name: Lambda
Actions:
  - Name: Lambda
    ActionTypeId:
      Category: Invoke
```

```
      Owner: AWS
      Provider: Lambda
      Version: '1'
    RunOrder: 1
    Configuration:
      FunctionName: myLambdaFunction
      UserParameters: 'http://192.0.2.4'
    OutputArtifacts: []
    InputArtifacts: []
    Region: us-west-2
```

JSON

```
{
    "Name": "Lambda",
    "Actions": [
        {
            "Name": "Lambda",
            "ActionTypeId": {
                "Category": "Invoke",
                "Owner": "AWS",
                "Provider": "Lambda",
                "Version": "1"
            },
            "RunOrder": 1,
            "Configuration": {
                "FunctionName": "myLambdaFunction",
                "UserParameters": "http://192.0.2.4"
            },
            "OutputArtifacts": [],
            "InputArtifacts": [],
            "Region": "us-west-2"
        }
    ]
},
```

JSON 事件範例

Lambda 動作會傳送 JSON 事件，其中包含工作 ID、管線動作組態、輸入和輸出成品位置，以及成品的任何加密資訊。工作工作者會存取這些詳細資料以完成 Lambda 動作。如需詳細資訊，請參閱 任務詳細資訊。以下為範例 事件。

```
{
    "CodePipeline.job": {
        "id": "11111111-abcd-1111-abcd-111111abcdef",
        "accountId": "111111111111",
        "data": {
            "actionConfiguration": {
                "configuration": {
                    "FunctionName": "MyLambdaFunction",
                    "UserParameters": "input_parameter"
                }
            },
            "inputArtifacts": [
                {
                    "location": {
                        "s3Location": {
                            "bucketName": "bucket_name",
                            "objectKey": "filename"
                        },
                        "type": "S3"
                    },
                    "revision": null,
                    "name": "ArtifactName"
                }
            ],
            "outputArtifacts": [],
            "artifactCredentials": {
                "secretAccessKey": "secret_key",
                "sessionToken": "session_token",
                "accessKeyId": "access_key_ID"
            },
            "continuationToken": "token_ID",
            "encryptionKey": {
              "id": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
              "type": "KMS"
            }
        }
    }
}
```

JSON 事件會針對中的 Lambda 動作提供下列工作詳細資訊 CodePipeline：

- `id`：唯一系統產生的任務 ID。

- accountId：與工作相關聯的AWS帳戶 ID。
- data：任務工作者完成任務所需的其他資訊。
  - actionConfiguration：Lambda 動作的動作參數。如需定義，請參閱 組態參數 。
  - inputArtifacts：提供給動作的成品。
    - location：成品存放區位置。
      - s3Location：動作的輸入成品位置資訊。
        - bucketName：動作的管道成品存放區名稱 (例如，名為 codepipeline-us-east-2-1234567890 的 Amazon S3 儲存貯體)。
        - objectKey：應用程式的名稱 (例如，CodePipelineDemoApplication.zip)。
      - type：位置中成品的類型。目前，S3 是唯一有效的成品類型。
    - revision：成品的修訂 ID。視物件類型而定，這可以是提交 ID (GitHub) 或修訂 ID (Amazon 簡單儲存服務)。如需詳細資訊，請參閱「ArtifactRevision」。
    - name：要處理的成品名稱，例如 MyApp。
  - outputArtifacts：動作的輸出。
    - location：成品存放區位置。
      - s3Location：動作的輸出成品位置資訊。
        - bucketName：動作的管道成品存放區名稱 (例如，名為 codepipeline-us-east-2-1234567890 的 Amazon S3 儲存貯體)。
        - objectKey：應用程式的名稱 (例如，CodePipelineDemoApplication.zip)。
      - type：位置中成品的類型。目前，S3 是唯一有效的成品類型。
    - revision：成品的修訂 ID。視物件類型而定，這可以是提交 ID (GitHub) 或修訂 ID (Amazon 簡單儲存服務)。如需詳細資訊，請參閱「ArtifactRevision」。
    - name：成品輸出的名稱，例如 MyApp。
  - artifactCredentials：用於存取 Amazon S3 儲存貯體中輸入和輸出成品的工AWS作階段登入資料。這些登入資料是 AWS Security Token Service (AWS STS) 發出的臨時登入資料。
    - secretAccessKey：工作階段的私密存取金鑰。
    - sessionToken：工作階段的字符。
    - accessKeyId：工作階段的私密存取金鑰。
  - continuationToken：動作產生的字符。未來動作會使用此字符來識別動作的執行中執行個體。動作完成時，不應提供接續字符。

- `encryptionKey`：用來加密成品存放區中資料的加密金鑰，例如 AWS KMS 金鑰。若尚未定義，則使用 Amazon 簡單儲存服務的預設金鑰。
  - `id`：用來識別金鑰的 ID。如果是 AWS KMS 金鑰，您可以使用金鑰 ID、金鑰 ARN 或別名 ARN。
  - `type`：加密金鑰的類型，例如 AWS KMS。

另請參閱

以下相關資源可協助您使用此動作。

- AWS CloudFormation使用者指南 — 如需管線的 Lambda 動作和AWS CloudFormation成品的詳細資訊，請參閱將參數覆寫函數與 CodePipeline 管道搭配使用、自動化部署 Lambda 應用程式和AWS CloudFormation成品。
- 在 CodePipeline 中於管道中呼叫 AWS Lambda 函數— 此程序提供範例 Lambda 函數，並示範如何使用主控台建立具有 Lambda 叫用動作的管道。

Snyk 動作結構參考

S nyk 動作可 CodePipeline 自動偵測並修正開放原始碼中的安全性弱點。您可以將 Snyk 與第三方存儲庫（例如 GitHub 或 Bitbucket Cloud）中的應用程序源代碼一起使用，或與容器應用程序的映像一起使用。您的處理行動將掃描並報告您設定的弱點等級和警示。

> **Note**

主題

動作類型識別碼

- 類別：`Invoke`

- 擁有者：ThirdParty
- 提供者：Snyk
- 版本：1

範例：

```
{
    "Category": "Invoke",
    "Owner": "ThirdParty",
    "Provider": "Snyk",
    "Version": "1"
},
```

Input artifacts (輸入成品)

- 人工因素數目：1
- 描述：組成呼叫動作之輸入成品的檔案。

輸出成品

- 人工因素數目：1
- 描述：組成呼叫動作之輸出成品的檔案。

另請參閱

以下相關資源可協助您使用此動作。

- 如需有關在中使用 Snyk 動作的詳細資訊 CodePipeline，請參閱使用 Snyk 自動掃描中 CodePipeline 的弱點。

AWS Step Functions

AWS CodePipeline 動作會執行下列動作：

- 從您的管道啟動 AWS Step Functions 狀態機器執行。

- 透過動作組態中的屬性或位於管道成品中做為輸入傳遞的檔案，為狀態機器提供初始狀態。
- 選擇性地設定執行 ID 字首，以識別源自動作的執行。
- 支援標準和快速狀態機器。

> **Note**
>
> 此功能不在亞太區域 (香港) 和歐洲 (米蘭) 區域中提供。如需引用其他可行的動作，請參產品與服務整合 CodePipeline。

動作類型

- 類別：`Invoke`
- 擁有者：`AWS`
- 提供者：`StepFunctions`
- 版本：`1`

組態參數

StateMachineArn

：必要 是

要叫用之狀態機器的 Amazon Resource Name (ARN)。

ExecutionNamePrefix

：必要 否

依預設，動作執行 ID 會用作為狀態機執行名稱。如果有提供字首，會與連字號用於動作執行 ID 的前綴，並一起用作為狀態機執行名稱。

```
myPrefix-1624a1d1-3699-43f0-8e1e-6bafd7fde791
```

對於快速狀態機器，名稱應該只包含 0-9、A-Z、a-z、- 和 _。

InputType

：必要 否

- 常值(default): 如果指定，則輸入欄位會直接傳遞到狀態機器輸入。

  的項目範例輸入字段時間常值處於選中狀態：

  ```
  {"action": "test"}
  ```

- FilePath：輸入成品中的檔案內容會由輸入欄位會用作為狀態機器執行的輸入。當 InputType 設定為 FilePath 時，需要輸入成品。

  的項目範例輸入字段時間FilePath處於選中狀態：

  ```
  assets/input.json
  ```

Input

：必要 有條件

- 常值：時機InputType已設定為常值（默認），則此欄為選用欄位。

  如果有提供此項，input (輸入) 欄位會直接用作為狀態機器執行的輸入。否則，會使用空的 JSON 物件 `{}` 叫用狀態機器。

- FilePath：時機InputType已設定為FilePath，則此欄位為必填。

  當 InputType 設定為 FilePath 時，也需要輸入成品。

  在指定的輸入成品中的檔案內容會用作為狀態機器執行的輸入。

Input artifacts (輸入成品)

- 成品數量： `0 to 1`
- 描述：如果InputType已設定為FilePath，則此成品是必要的，並且會用來為狀態機器執行提供輸入。

輸出成品

- 成品數量： `0 to 1`
- 描述：
  - 標準狀態機器：如果有提供此項，則會使用狀態機器的輸出填入輸出成品。這是從`output`屬性Step Functions DescribeExecution API響應後狀態機器執行成功完成。

- 快速狀態機器：不支援.

輸出變數

此動作會產生輸出變數，並且可供管道中下游動作的動作組態進行參考。

如需詳細資訊，請參閱 Variables。

StateMachineArn

狀態機器的 ARN。

ExecutionArn

執行狀態機器的 ARN。僅限標準狀態機器。

動作組態範例

預設輸入的範例

YAML

```
Name: ActionName
ActionTypeId:
  Category: Invoke
  Owner: AWS
  Version: 1
  Provider: StepFunctions
OutputArtifacts:
  - Name: myOutputArtifact
Configuration:
  StateMachineArn: arn:aws:states:us-east-1:111122223333:stateMachine:HelloWorld-
StateMachine
  ExecutionNamePrefix: my-prefix
```

JSON

```
{
    "Name": "ActionName",
    "ActionTypeId": {
        "Category": "Invoke",
```

```
        "Owner": "AWS",
        "Version": 1,
        "Provider": "StepFunctions"
    },
    "OutputArtifacts": [
        {
            "Name": "myOutputArtifact"
        }
    ],
    "Configuration": {
        "StateMachineArn": "arn:aws:states:us-
east-1:111122223333:stateMachine:HelloWorld-StateMachine",
        "ExecutionNamePrefix": "my-prefix"
    }
}
```

常值輸入的範例

YAML

```
Name: ActionName
ActionTypeId:
  Category: Invoke
  Owner: AWS
  Version: 1
  Provider: StepFunctions
OutputArtifacts:
  - Name: myOutputArtifact
Configuration:
  StateMachineArn: arn:aws:states:us-east-1:111122223333:stateMachine:HelloWorld-
StateMachine
  ExecutionNamePrefix: my-prefix
  Input: '{"action": "test"}'
```

JSON

```
{
    "Name": "ActionName",
    "ActionTypeId": {
        "Category": "Invoke",
        "Owner": "AWS",
        "Version": 1,
```

```
        "Provider": "StepFunctions"
    },
    "OutputArtifacts": [
        {
            "Name": "myOutputArtifact"
        }
    ],
    "Configuration": {
        "StateMachineArn": "arn:aws:states:us-
east-1:111122223333:stateMachine:HelloWorld-StateMachine",
        "ExecutionNamePrefix": "my-prefix",
        "Input": "{\"action\": \"test\"}"
    }
}
```

輸入檔案的範例

YAML

```
Name: ActionName
InputArtifacts:
  - Name: myInputArtifact
ActionTypeId:
  Category: Invoke
  Owner: AWS
  Version: 1
  Provider: StepFunctions
OutputArtifacts:
  - Name: myOutputArtifact
Configuration:
  StateMachineArn: 'arn:aws:states:us-east-1:111122223333:stateMachine:HelloWorld-
StateMachine'
  ExecutionNamePrefix: my-prefix
  InputType: FilePath
  Input: assets/input.json
```

JSON

```
{
    "Name": "ActionName",
    "InputArtifacts": [
        {
```

```
            "Name": "myInputArtifact"
        }
    ],
    "ActionTypeId": {
        "Category": "Invoke",
        "Owner": "AWS",
        "Version": 1,
        "Provider": "StepFunctions"
    },
    "OutputArtifacts": [
        {
            "Name": "myOutputArtifact"
        }
    ],
    "Configuration": {
        "StateMachineArn": "arn:aws:states:us-
east-1:111122223333:stateMachine:HelloWorld-StateMachine",
        "ExecutionNamePrefix": "my-prefix",
        "InputType": "FilePath",
        "Input": "assets/input.json"
    }
}
```

Behavior (行為)

在發佈期間， CodePipeline 會使用如動作組態中指定的輸入來執行狀態機器。

當 InputType 設定為 Literal (常值) 時，Input (輸入) 動作組態欄位的內容會用作為狀態機器的輸入。當沒有提供常值輸入時，狀態機器執行會使用空的 JSON 物件 {}。如需在不輸入的情況下執行狀態機器執行的詳細資訊，請參閱 Step Functions StartExecution API。

當 InputType 設定為 FilePath 時，動作會將輸入成品解壓縮，並使用在 Input (輸入) 動作組態欄位中指定的檔案內容做為狀態機器的輸入。指定 FilePath 時，Input (輸入) 欄位為必要欄位，且必須存在輸入成品；否則，動作會失敗。

成功啟動執行後，行為將分歧為兩種狀態機類型，「標準」和「快速」。

標準狀態機器

如果標準狀態機器執行啟動成功， CodePipeline 輪詢`DescribeExecution`API，直到執行達到終端狀態為止。如果執行成功完成，表示動作成功；否則會失敗。

如果已設定輸出成品，該成品將包含狀態機器的傳回值。這是從`output`屬性Step Functions DescribeExecution API響應後狀態機器執行成功完成。請注意，此 API 上有強制執行輸出長度限制。

錯誤處理

- 如果該動作無法啟動狀態機器執行，則動作執行會失敗。
- 如果狀態機器執行無法達到終端狀態 CodePipeline Step Functions 動作達到其超時 (預設為 7 天)，則動作執行會失敗。儘管發生此類失敗，狀態機器可能會繼續執行。如需有關 Step Functions 中狀態機器執行超時的詳細資訊，請參標準與快速工作流程。

> **Note**
>
> 您可以為具有動作的帳戶要求增加叫用動作逾時的配額。不過，增加配額會套用至該帳戶所有區域中此類型的所有動作。

- 如果狀態機器執行達到 FAILED、TIMED_OUT 或 ABORTED 的終端狀態，則動作執行會失敗。

快速狀態機器

如果快速狀態機器執行啟動成功，叫用動作執行會成功完成。

針對快速狀態機器設定的動作考量：

- 您無法指定輸出成品。
- 該動作不會等待狀態機器執行完成。
- 在 CodePipeline 中開始動作執行後，即使狀態機器執行失敗，動作執行也會成功。

錯誤處理

- 如果 CodePipeline 無法啟動狀態機器執行，則動作執行會失敗。否則，該動作會立即成功。操作在 CodePipeline 而無論狀態機器執行需要多長時間才能完成或其結果如何。

另請參閱

以下相關資源可協助您使用此動作。

- AWS Step Functions開發人員指南— 有關狀態機、執行和狀態機輸入的信息，請參閱AWS Step Functions開發人員指南。

- 教學課程：使用AWS Step Functions叫用管道中的動作— 本教學讓您開始使用範例標準狀態機器，並展示如何使用控制台通過添加 Step Functions 調用動作來更新管道。

整合模型參考

有許多針對第三方服務的預先構建集成，可幫助將現有的客戶工具構建到管道發布流程中。合作夥伴或協力廠商服務供應商會使用整合模型來實作動作類型，以便在中使用 CodePipeline。

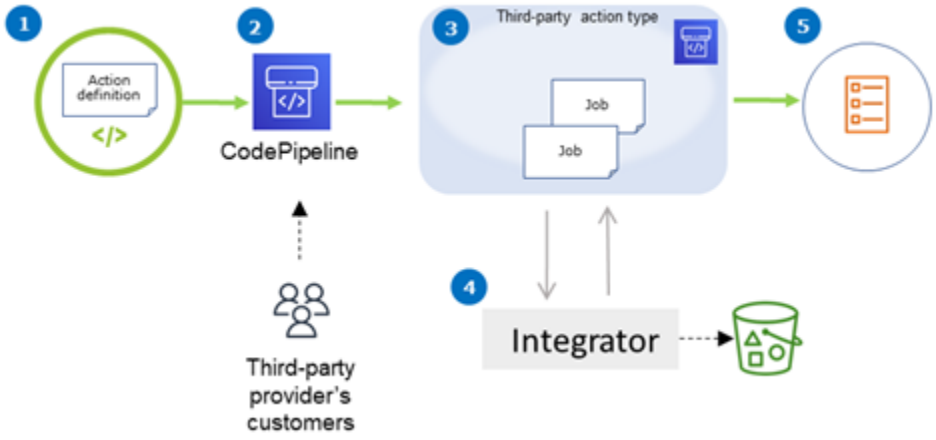
當您計劃或使用中受支援整合模型管理的動作類型時，請使用此參考 CodePipeline。

若要將您的第三方動作類型認證為合作夥伴整合 CodePipeline，請參閱合作AWS夥伴網路 (APN)。此資訊是「AWS CLI參考」的補充。

主題

協力廠商動作類型如何與整合經銷商合作

您可以將第三方動作類型新增至客戶管道，以完成客戶資源的工作。整合經銷商管理工作請求，並使用執行動作 CodePipeline。下圖顯示為客戶建立的第三方動作類型，以便在其管道中使用。客戶設定動作後，動作會執行並建立由整合商的動作引擎處理的工作請求。



圖表顯示以下步驟：

1. 動作定義已註冊並在中提供 CodePipeline。第三方動作可供第三方供應商的客戶。

2. 供應商的客戶選擇並設定中的動作 CodePipeline。
3. 動作執行並將工作排入佇列中 CodePipeline。當工作準備就緒時 CodePipeline，它會傳送工作要求。
4. 整合者 (協力廠商輪詢 API 或 Lambda 函數的工作工作者) 會接收工作要求、傳回確認，然後處理動作的成品。
5. 整合者傳回成功/失敗輸出 (工作者使用成功/失敗 API 或 Lambda 函數傳送成功/失敗輸出)，其中包含工作結果和延續權杖。

如需有關可用來請求、檢視和更新動作類型之步驟的資訊，請參閱使用動作類型。

概念

本節針對第三方動作類型使用下列條款：

動作類型

可重複的程序，可在執行相同持續交付工作負載的管道中重複使用。動作類型由`Owner`、`CategoryProvider`、和識別`Version`。例如：

```
{

    "Category": "Deploy",
    "Owner": "AWS",
    "Provider": "CodeDeploy",
    "Version": "1"
},
```

相同類型的所有動作共用相同的實作。

動作

動作類型的單一執行個體，這是在管線階段內發生的其中一個離散程序。這通常包括執行此動作之管道的特定使用者值。

動作定義

動作類型的結構描述，可定義配置動作和輸入/輸出成品所需的內容。

動作執行

已執行的工作集合，以判斷客戶管線上的動作是否成功。

動作執行引擎

動作執行組態的屬性，可定義動作類型所使用的整合類型。有效值為 `JobWorker` 和 `Lambda`。

整合

描述整合者執行以實作動作類型的軟體。 CodePipeline 支援與兩個支援的動作引擎`JobWorker`和相對應的兩種整合類型`Lambda`。

整合商

擁有動作類型實行的人員。

任務

與管道和客戶上下文一起執行集成的一部分工作。動作執行由一或多個任務組成。

Job 工人

處理客戶輸入並執行工作的服務。

支援的整合模型

CodePipeline 有兩種整合模式：

- Lambda 整合模型：此整合模型是使用 CodePipeline. Lambda 整合模型會在動作執行時使用 Lambda 函數來處理工作請求。
- Job 者整合模型：工作者整合模型是先前用於第三方整合的模型。工作背景工作者整合模型使用設定為在動作執行時連絡 CodePipeline API 以處理工作要求的工作工作者。

為了進行比較，下表描述了兩種模型的功能：

	Lambda 合模型	Job 人, 集成, 模型
Description (描述)	整合經銷商會將整合寫入為 Lambda 函數， CodePipeline 只要有可用於動作的工作，就會呼叫此函數。Lambda 函數不會輪詢可用的工作，而是等待直到收到下一個工作請求為止。	整合經銷商會以工作者的身分撰寫整合，並持續輪詢客戶管道上的可用工作。然後，工作者會執行工作，並使用 CodePipeline API 將 CodePipeline 工作結果提交回。

	Lambda 合模型	Job 人, 集成, 模型
基礎建	AWS Lambda	將工作工作者程式碼部署到整合商的基礎設施，例如 Amazon EC2 執行個體。
發展努力	整合只包含商業邏輯。	除了包含商務邏輯之外，整合還需要與 CodePipeline API 互動。
行動努力	較少的操作工作，因為基礎設施只是AWS資源。	更高的操作工作量，因為工作者需要其獨立硬件。
Job 執行時間上限	如果整合需要主動執行超過 15 分鐘，則無法使用此模型。此動作適用於需要啟動程序 (例如，根據客戶的程式碼人工因素啟動建置)，並在完成後傳回結果的整合經銷商。我們不建議整合經銷商繼續等待組建完成。相反，返回一個延續。CodePipeline如果從整合者的程式碼接收到延續，以檢查工作直到工作完成為止，則會在另外 30 秒內建立新工作。	使用此模型可以持續很長的運行作業（小時/天）。

Lambda 整合模型

支援的 Lambda 整合模型包括建立 Lambda 函數，以及定義第三方動作類型的輸出。

更新您的 Lambda 函數以處理來源的輸入 CodePipeline

您可以建立新 Lambda 函數。您可以將商業邏輯新增至 Lambda 函數，只要管道上有可用的動作類型工作，就會執行該函數。例如，考慮到客戶和管道的上下文，您可能希望為客戶在服務中啟動構建。

使用下列參數來更新 Lambda 函數以處理來源的輸入 CodePipeline。

格式:

- `jobId`:

- 系統產生的唯一名稱。
- 類型：字串
- 模式:[0-9-f] {8}-[0-9-F] {4}-[0-9-F] {4}-[0-9-f] {4}-[0-9-F] {12}

- accountId:
  - 執行工作時要使用的客AWS戶帳戶 ID。
  - 類型：字串
  - 圖案:[0-9] {12}
- data:
  - 整合用來完成工作之工作的其他資訊。
  - 包含以下內容的地圖：
    - actionConfiguration:
      - 動作的組態資料。動作設定欄位是鍵值對的對映，供您的客戶輸入值。當您設定動作時，這些鍵由動作類型定義檔案中的關鍵參數決定。在此範例中，值由在Username和Password欄位中指定資訊的動作使用者決定。
      - 類型：字串到字串到字串的字串，可選擇存

        範例：

        ```
        "configuration": {
            "Username": "MyUser",
            "Password": "MyPassword"
        },
        ```
    - encryptionKey:
      - 代表用來加密成品存放區資訊，例如AWS KMS金鑰。
      - 內容：資料類型的類型encryptionKey，可選擇性地顯示
    - inputArtifacts:
      - 要處理的成品名稱，例如測試或建立成品。
      - 內容：數據類型的列表Artifact，可選擇性地顯示
    - outputArtifacts:
      - 動作輸出的資訊。
      - 內容：數據類型的列表Artifact，可選擇性地顯示

- actionCredentials:
  - 表示AWS工作階段認證物件。這些認證是由發行的臨時登入資料AWS STS。它們可用來存取 S3 儲存貯體中的輸入和輸出成品，用於將管道的成品存放在中 CodePipeline。

    這些認證也具有與動作類型定義檔案中指定的原則陳述式範本相同的權限。
  - 內容：資料類型的類型AWSSessionCredentials，可選擇性地顯示
- actionExecutionId:
  - 動作執行的外部識別碼。
  - 類型：字串
- continuationToken:
  - 工作以非同步方式繼續工作所需的系統產生的 Token (例如部署 ID)。
  - 類型：字串，可選擇存在

數據類型：

- encryptionKey:
  - id:
    - 用來識別金鑰的 ID。如果是 AWS KMS 金鑰，您可以使用金鑰 ID、金鑰 ARN 或別名 ARN。
    - 類型：字串
  - type:
    - 加密金鑰的加密金鑰，例如AWS KMS金鑰。
    - 類型：字串
    - 有效值：KMS
- Artifact:
  - name:
    - 工件的名稱。
    - 類型：字串，可選擇存在
  - revision:
    - 成品的修訂 ID。視物件類型而定，這可能是提交 ID (GitHub) 或修訂 ID (Amazon S3)。
    - 類型：字串，可選擇存在
  - location:
    - 成品的成品。

    - 內容：資料類型的類型ArtifactLocation，可選擇性地顯示
- ArtifactLocation:
  - type:
    - 位置中的人工因素類型。
    - 類型：字串，可選擇存在
    - 有效值：S3
  - s3Location:
    - 包含修訂版本的 S3 儲存貯體。
    - 內容：資料類型的類型S3Location，可選擇性地顯示
- S3Location:
  - bucketName:
    - S3 儲存貯體的名稱。
    - 類型：字串
  - objectKey:
    - S3 儲存貯體中的物件，可唯一識別儲存貯體中的物件。
    - 類型：字串
- AWSSessionCredentials:
  - accessKeyId:
    - 工作階段的存取金鑰。
    - 類型：字串
  - secretAccessKey:
    - 工作階段的私密存取金鑰。
    - 類型：字串
  - sessionToken:
    - 工作階段的權杖。
    - 類型：字串

範例：


```
{
    "jobId": "01234567-abcd-abcd-abcd-012345678910",
```

```
    "accountId": "012345678910",
    "data": {
        "actionConfiguration": {
            "key1": "value1",
            "key2": "value2"
        },
        "encryptionKey": {
            "id": "123-abc",
            "type": "KMS"
        },
        "inputArtifacts": [
            {
                "name": "input-art-name",
                "location": {
                    "type": "S3",
                    "s3Location": {
                        "bucketName": "inputBucket",
                        "objectKey": "inputKey"
                    }
                }
            }
        ],
        "outputArtifacts": [
            {
                "name": "output-art-name",
                "location": {
                    "type": "S3",
                    "s3Location": {
                        "bucketName": "outputBucket",
                        "objectKey": "outputKey"
                    }
                }
            }
        ],
        "actionExecutionId": "actionExecutionId",
        "actionCredentials": {
            "accessKeyId": "access-id",
            "secretAccessKey": "secret-id",
            "sessionToken": "session-id"
        },
        "continuationToken": "continueId-xxyyzz"
    }
}
```

將 Lambda 函數的成品 CodePipeline

整合商的工作背景工作者資源必須在成功、失敗或繼續案例中傳回有效承載。

格式:

- `result`：工作的結果。
  - 必要
  - 有效值（不區分大小寫）：
    - `Success`：表示工作成功並終止。
    - `Continue`：表示工作成功且必須繼續，例如，如果為相同動作執行重新呼叫工作 Worker。
    - `Fail`：表示任務已失敗，是終端。
- `failureType`：要與失敗工作相關聯的失敗類型。

  夥伴動作的`failureType`類別描述執行工作時遇到的失敗類型。整合經銷商會在將工作失敗結果傳回至時，設定類型以及失敗訊息 CodePipeline。
  - 選用。如果結果為，則需要`Fail`。
  - 如果`result`是`Success`或，則必須為空`Continue`
  - 有效值：
    - ConfigurationError
    - JobFailed
    - PermissionsError
    - RevisionOutOfSync
    - RevisionUnavailable
    - SystemUnavailable
- `continuation`: 繼續狀態要傳遞給當前動作執行中的下一個作業。
  - 選用。如果結果為，則需要`Continue`。
  - 如果`result`是`Success`或，則必須為 null`Fail`。
  - 屬性：
    - `State`：要傳遞的狀態的散列。
- `status`：動作執行的狀態。
  - 選用。
  - 屬性：

  - ExternalExecutionId：要與工作關聯的選擇性外部執行 ID 或認可 ID。
  - Summary：發生了什麼的可選摘要。在失敗情況下，這會成為使用者看到的失敗訊息。
- outputVariables：要傳遞給下一個動作執行的一組鍵/值對。
  - 選用。
  - 如果result是Continue或，則必須為 nullFail。

範例：

```
{
    "result": "success",
    "failureType": null,
    "continuation": null,
    "status": {
        "externalExecutionId": "my-commit-id-123",
        "summary": "everything is dandy"
    },
    "outputVariables": {
        "FirstOne": "Nice",
        "SecondOne": "Nicest",
        ...
    }
}
```

使用繼續令牌等待異步進程的結果

令continuation牌是有效負載的一部分和 Lambda 函數的結果。這是一種將工作狀態傳遞給 CodePipeline 並指出工作需要繼續的方法。例如，整合經銷商在其資源上為客戶啟動組建之後，不會等待組建完成，而是透過傳回result ascontinue 並將 CodePipeline 組建的唯一 ID 作 CodePipeline 為continuation Token，表示不會有終端結果。

> **Note**
>
> Lambda 函數最多只能執行 15 分鐘。如果工作需要執行更長的時間，您可以使用延續權杖。

該 CodePipeline 團隊在 30 秒後調用整合者，並在其有效負載中使用相同的continuation令牌，以便它可以檢查它是否完成。如果構建完成，集成器返回終端成功/失敗結果，否則繼續。

提供 CodePipeline 在執行時間叫用整合商 Lambda 函數的權限

您可以將權限新增至整合商 Lambda 函數，以便為 CodePipeline 服務提供使用服 CodePipeline 務主體叫用該函數的許可：`codepipeline.amazonaws.com`。您可以使用AWS CloudFormation或命令列來新增權限。如需範例，請參閱 使用動作類型。

Job 人, 集成, 模型

設計完高階工作流程後，您可以建立工作 Worker。雖然協力廠商動作的細節決定了工作者需要什麼，但是大多數協力廠商動作的工作者都包含下列功能：

- 輪詢 CodePipeline 使用的工作`PollForThirdPartyJobs`。
- 確認工作並將結果傳回至 CodePipeline 使用`AcknowledgeThirdPartyJob`、`PutThirdPartyJobSuccessResult`和`PutThirdPartyJobFailureResult`。
- 從管道擷取成品和/或將成品放入 Amazon S3 儲存貯體。若要從 Amazon S3 儲存貯體下載成品，您必須建立一個使用簽名版本 4 簽署 (Sig V4) 的 Amazon S3 用戶端。簽名 V4 是必需的AWS KMS.

  若要將成品上傳到 Amazon S3 儲存貯體，您還必須將 Amazon S3`PutObject` 請求設定為透過 AWS Key Management Service (AWS KMS) 使用加密。 AWS KMS用途AWS KMS keys。為了知道是否使用AWS 受管金鑰或客戶管理的金鑰來上傳人工因素，您的工作人員必須查看工作資料並檢查加密金鑰屬性。如果已設定屬性，您應在設定時使用該客戶管理的金鑰 IDAWS KMS。如果索引鍵屬性為 null，您可以使用AWS 受管金鑰. CodePipeline 除非另有配置，AWS 受管金鑰否則會使用。

  如需示範如何在 Java 或 .NET 中建立AWS KMS參數的範例，請參閱使用AWS開發套件AWS Key Management Service在 Amazon S3 中指定參數。如需有關的 Amazon S3 儲存貯體的詳細資訊 CodePipeline，請參閱CodePipeline 概念 。

選擇和設定任務工作者的許可管理策略

若要為中的第三方動作開發工作者 CodePipeline，您需要整合使用者和權限管理的策略。

最簡單的策略是建立具有AWS Identity and Access Management (IAM) 執行個體角色的 Amazon EC2 執行個體，以新增任務工作者所需的基礎設施，讓您輕鬆擴展整合所需的資源。您可以使用內建的整合AWS來簡化工作者與 CodePipeline.

進一步了解 Amazon EC2，並判斷它是否是您整合的正確選擇。如需詳細資訊，請參閱 Amazon EC2-虛擬伺服器託管。如需有關設定 Amazon EC2 執行個體的詳細資訊，請參閱 Amazon EC2 Linux 執行個體入門。

另一個需要考慮的策略是使用與 IAM 聯合身分識別來整合您現有的身分識別提供者系統和資源。如果您已經擁有公司身分識別提供者，或已設定為支援使用 Web 身分提供者的使用者，則此策略非常有用。聯合身分可讓您授予AWS資源的安全存取權 CodePipeline，包括無需建立或管理 IAM 使用者。您可以使用功能和政策以符合密碼安全要求和輪換登入資料。您可以使用範例應用程式做為您自己設計的範本。如需資訊，請參閱管理聯合。

若要提供存取權，請新增許可到您的使用者、群組或角色：

- AWS IAM Identity Center 中的使用者和群組：

  建立許可集合。請遵循《AWS IAM Identity Center 使用者指南》的建立許可集合中的指示。
- 透過身分提供者在 IAM 中管理的使用者：

  建立聯合身分的角色。請遵循《IAM 使用者指南》的為第三方身分提供者 (聯合) 建立角色中的指示。
- IAM 使用者：
  - 建立您的使用者可擔任的角色。請遵循《IAM 使用者指南》的為 IAM 使用者建立角色中的指示。
  - (不建議) 將政策直接連接至使用者，或將使用者新增至使用者群組。請遵循《IAM 使用者指南》的新增許可到使用者 (主控台)中的指示。

以下是您可能建立以搭配協力廠商工作者使用的範例原則。此政策僅做為範例使用，並以現狀提供。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codepipeline:PollForThirdPartyJobs",
        "codepipeline:AcknowledgeThirdPartyJob",
        "codepipeline:GetThirdPartyJobDetails",
        "codepipeline:PutThirdPartyJobSuccessResult",
        "codepipeline:PutThirdPartyJobFailureResult"
      ],
      "Resource": [
        "arn:aws:codepipeline:us-east-2::actionType:ThirdParty/Build/Provider/1/"
```

```
        ]
      }
    ]
}
```

映像定義檔案參考

本節僅供參考。如需有關利用來源建立管道或部署容器動作的詳細資訊，請參閱 在中建立管線 CodePipeline。

AWS CodePipeline容器動作 (例如 Amazon ECR 來源動作或 Amazon ECS 部署動作) 的任務工作程式使用定義檔案將映像 URI 和容器名稱對應到任務定義。每個定義檔案都是一種 JSON 格式的檔案，由動作提供者使用，如下所示：

- 亞馬遜 ECS 標準部署需要`imagedefinitions.json`檔案做為輸入以部署動作。
- Amazon ECS 藍/綠部署`imageDetail.json`檔案做為輸入以部署動作。
  - Amazon ECR 來源動作產生 `imageDetail.json` 檔案，提供做為來源動作的輸出。

主題

適用於 Amazon ECS 標準部署動作的 imagedefinitions.json 檔案

映像定義文件為 JSON 檔案，說明 Amazon ECS 容器名稱、映像及標籤。若您部署容器式應用程式，必須產生映像定義檔案 CodePipeline 使用 Amazon ECS 容器和影像識別功能的工作者，以便從映像儲存庫擷取，例如 Amazon ECR。

> **Note**
>
> 此檔案的預設檔案名稱為 `imagedefinitions.json`。如果您選擇使用不同的檔案名稱，必須在建立管道部署階段時提供該名稱。

根據下列考量建立 `imagedefinitions.json` 檔案：

- 檔案必須使用 UTF-8 編碼。
- 映像定義檔案的檔案大小上限為 100 KB。
- 您必須建立 檔案做為來源或建置成品，以做為部署動作的輸入成品。換句話說，請確保檔案上傳到您的原始碼位置，例如 CodeCommit 存儲庫，或作為內置輸出工件生成。

imagedefinitions.json 檔案提供容器名稱和映像 URI。它必須以下列索引鍵/值組建構。

索引鍵	數值
name	*container_name*
imageUri	*imageUri*

此處為 JSON 結構，該容器名稱為 sample-app，映像 URI 為 ecs-repo 且標籤為 latest：

```
[
  {
    "name": "sample-app",
    "imageUri": "11111EXAMPLE.dkr.ecr.us-west-2.amazonaws.com/ecs-repo:latest"
  }
]
```

您也可以建構此檔案以列出多個容器映像組。

JSON 結構：

```
[
  {
    "name": "simple-app",
    "imageUri": "httpd:2.4"
  },
  {
    "name": "simple-app-1",
    "imageUri": "mysql"
  },
  {
    "name": "simple-app-2",
    "imageUri": "java1.8"
  }
]
```

在您建立管道前，請使用下列步驟來設定 imagedefinitions.json 檔案。

1. 在管道中規劃容器式應用程式部署時，請適時規劃原始碼階段與建置階段。
2. 選擇下列其中一項：

a. 若您的管道建立管道以便略過建置階段，您必須手動建立 JSON 檔案並上傳到您的原始碼儲存庫，讓原始碼動作可提供成品。使用文字編輯器建立檔案，並為檔案命名或使用預設的 `imagedefinitions.json` 檔案名稱。推送映像定義檔案到您的原始碼儲存庫。

> **Note**
>
> 若您的原始碼儲存庫為 Amazon S3 儲存貯體，請記得壓縮 JSON 檔案。

b. 若您的管道有建置階段，請新增命令到您的建置規格檔案；此檔案會在建置階段將映像定義檔案輸出至來源碼儲存庫。以下範例使用 printf 命令來建立 `imagedefinitions.json` 檔案。在 `buildspec.yml` 檔案的 `post_build` 部分列出此命令：

```
printf '[{"name":"container_name","imageUri":"image_URI"}]' >
imagedefinitions.json
```

您必須納入映像定義檔案做為 `buildspec.yml` 檔案的輸出成品。

3. 當您在主控台中建立管道部署的所有頁面建立管道精靈，影像檔案名稱，輸入影像定義檔案名稱。

對於一個 step-by-step 建立使用 Amazon ECS 做為部署提供者的管道的教學，請參閱教學課程：使用 持續部署 CodePipeline。

適用於 Amazon ECS 藍色/綠色部署動作的 imageDetail.json 檔案

同時`imageDetail.json`文件為 JSON 檔案，說明您的 Amazon ECS 映像 URI。如果您是以容器式應用程式進行藍/綠部署，您必須產生`imageDetail.json`文件提供 Amazon ECS 和 CodeDeploy具有影像識別功能的工作者，以便從映像儲存庫擷取，例如 Amazon ECR。

> **Note**
>
> 此檔案的名稱必須是 `imageDetail.json`。

如需動作及其參數的描述，請參閱Amazon Elastic Coner Coner Coner Coner Coner ConCodeDeploy。

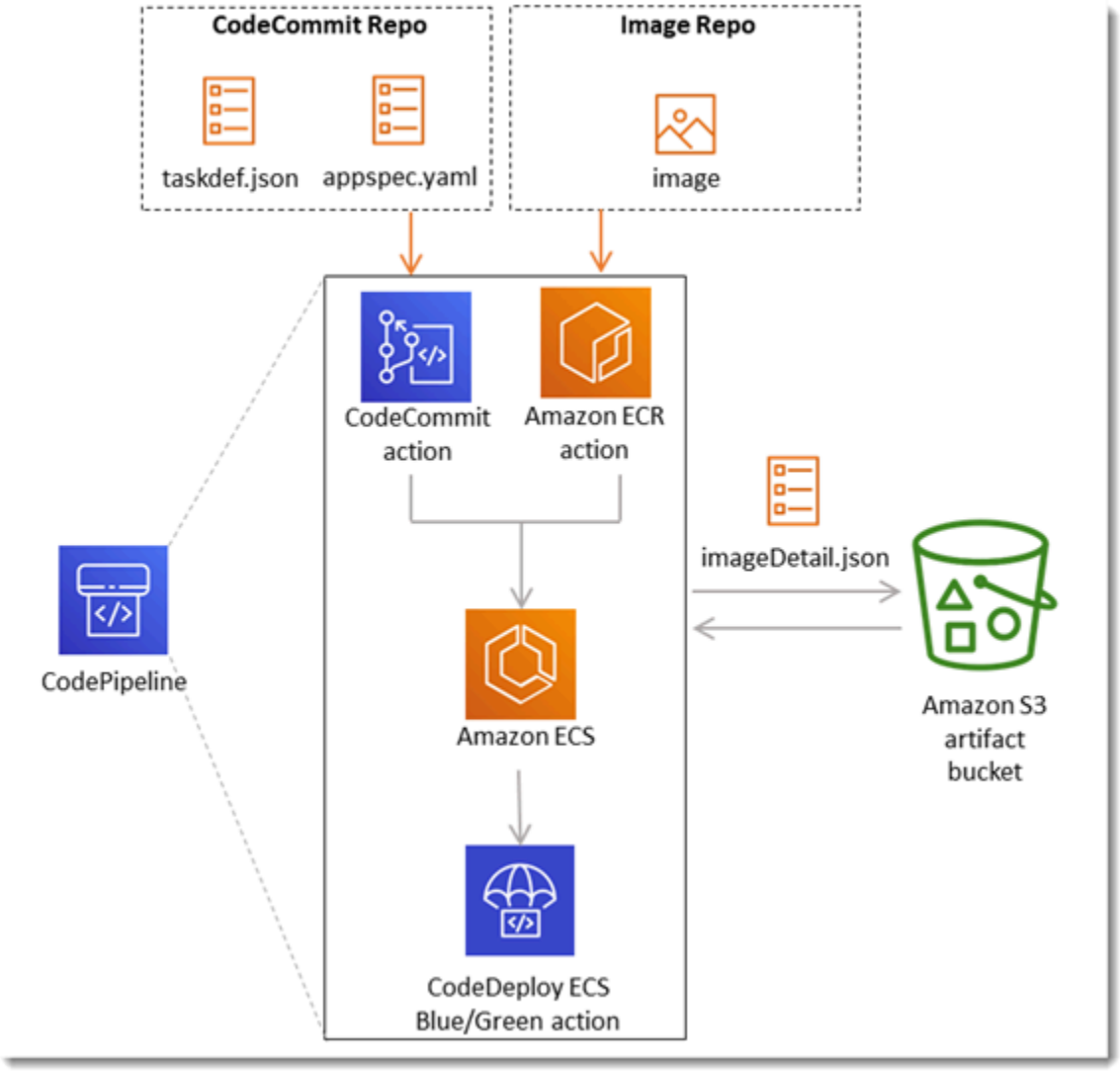
您必須建立 `imageDetail.json` 檔案做為來源或建置成品，以做為部署動作的輸入成品。您可以使用下列其中一種方法在管道中提供 `imageDetail.json` 檔案：

- 包括`imageDetail.json`檔案位於您的來源位置，以便在管道中提供此檔案做為 Amazon ECS 藍/綠部署動作的輸入。

> **Note**
>
> 若您的原始碼儲存庫為 Amazon S3 儲存貯體，請記得壓縮 JSON 檔案。

- 亞馬遜 ECR 來源動作會自動產生`imageDetail.json`檔案做為下一個動作的輸入成品。

> **Note**
>
> 由於 Amazon ECR 來源動作會建立此檔案，因此具有 Amazon ECR 來源動作的管道不需要手動提供`imageDetail.json`file.
> 如需建立包含 Amazon ECR 來源階段之管道的教學，請參閱教學課程：使用 Amazon ECR 來源和 ECS 到部署建立管道 CodeDeploy 。



`imageDetail.json` 檔案提供映像 URI。它必須以下列索引鍵/值組建構。

索引鍵	數值
ImageURI	*`image_URI`*

imageDetail.json

以下是 JSON 結構，其中影像 URI 是 `ACCOUNTID.dkr.ecr.us-west-2.amazonaws.com/dk-image-repo@sha256:example3`：

```
{
```

```
"ImageURI": "ACCOUNTID.dkr.ecr.us-west-2.amazonaws.com/dk-image-
repo@sha256:example3"
}
```

imageDetail.json (generated by ECR)

同時imageDetail.json每當有變更推送到映像儲存庫時，Amazon ECR 來源動作就會自動產生檔案。所以此imageDetail.jsonAmazon ECR 來源動作產生的將提供做為管道中從來源動作到下一個動作的輸出成品。

以下是 JSON 結構，其中儲存庫名稱為 dk-image-repo、影像 URI 為 ecs-repo，而映像標籤為 latest：

```
{
    "ImageSizeInBytes": "44728918",
    "ImageDigest":
 "sha256:EXAMPLE1122334455667788990bfea42ea2d3b8a1ee8329ba7e68694950afd3",
    "Version": "1.0",
    "ImagePushedAt": "Mon Jan 21 20:04:00 UTC 2019",
    "RegistryId": "EXAMPLE12233",
    "RepositoryName": "dk-image-repo",
    "ImageURI": "ACCOUNTID.dkr.ecr.us-west-2.amazonaws.com/dk-image-
repo@sha256:example3",
    "ImageTags": [
        "latest"
    ]
}
```

所以此imageDetail.json檔案將 URI 和容器名稱映射到 Amazon ECS 任務定義，如下所示：

- ImageSizeInBytes：儲存庫中的映像大小 (以位元組為單位) 。
- ImageDigest：sha256 映像資訊清單的摘要。
- Version：映像的版本。
- ImagePushedAt：最新映像推送到儲存庫的日期和時間。
- RegistryId：該AWS與內含儲存庫之登錄檔相關聯的帳戶 ID
- RepositoryName：推送映像的 Amazon ECR 儲存庫的 Amazon ECR 儲存庫的名稱
- ImageURI：映像的 URI。
- ImageTags：用於映像的標籤。

在您建立管道前，請使用下列步驟來設定 imageDetail.json 檔案。

1. 在管道中規劃容器式應用程式藍/綠部署時，請適時規劃原始碼階段與建置階段。
2. 選擇下列其中一項：

   a. 若您的管道已略過建置階段，您必須手動建立 JSON 檔案並上傳到您的原始碼儲存庫 CodeCommit，因此來源動作可以提供工件。使用文字編輯器建立檔案，並為檔案命名或使用預設的 imageDetail.json 檔案名稱。推送 imageDetail.json 檔案到您的原始碼儲存庫。

   b. 如果您的管道有建置階段，請執行以下操作：

      i. 新增命令到您的建置規格檔案，此檔案會在建置階段將映像定義檔案輸出至原始碼儲存庫。以下範例使用 printf 命令來建立 imageDetail.json 檔案。在 buildspec.yml 檔案的 post_build 區段列出此命令：

      ```
      printf '{"ImageURI":"image_URI"}' > imageDetail.json
      ```

      您必須在 buildspec.yml 檔案包含 imageDetail.json 檔案以做為輸出成品。

      ii. 在 buildspec.yml 檔案中新增 imageDetail.json 為成品檔案。

      ```
      artifacts:
        files:
          - imageDetail.json
      ```

Variables

本節僅供參考。如需有關建立變數的資訊，請參閱使用變數。

變數可讓您使用管線執行或動作執行時決定的值來設定管線動作。

某些動作提供者會產生一組已定義的變數。您可以從該動作提供者的預設變數索引鍵中選擇，例如遞交 ID。

> **Important**
>
> 傳遞秘密參數時，請勿直接輸入值。此值渲染為純文字，因此為可讀取。出於安全原因，請勿使用帶有密碼的純文本。我們強烈建議您使用 AWS Secrets Manager 來儲存機密。

若要查看 step-by-step 使用變數的範例：

- 如需在管線執行時傳送的管線層級變數的教學課程，請參閱。教學課程：使用管線層級變數
- 如需具有 Lambda 動作的教學課程，該動作使用上游動作 (CodeCommit) 中的變數並產生輸出變數，請參閱教學課程：使用變數搭配 Lambda 呼叫動作。
- 如需具有參考上游 AWS CloudFormation 動作之堆疊輸出變數之 CloudFormation 動作的自學課程，請參閱教學：建立透過 AWS CloudFormation 部署動作使用變數的管道。
- 如需含有訊息文字的手動核准動作範例，其中參考解析為 CodeCommit 提交 ID 和提交訊息的輸出變數，請參閱範例：在手動核准中使用變數。
- 如需具有可解析為 GitHub分支名稱之環境變數的範例 CodeBuild 動作，請參閱範例：搭配 CodeBuild 環境 BranchName變數使用變數。
- CodeBuild 動作會以變數形式產生所有已匯出為組建一部分的環境變數。如需詳細資訊，請參閱 CodeBuild 動作輸出變數。

變數限制

如需限制資訊，請參閱 AWS CodePipeline 中的配額。

> **Note**
>
> 當您在動作組態欄位中輸入變數語法時，請勿超過組態欄位的 1000 個字元限制。如果超過此限制，系統就會傳回驗證錯誤。

主題

概念

本節列出與變數和命名空間相關的主要術語和概念。

Variables

變數是鍵值組，可用來動態設定管道中的動作。目前有三種方式可以使用這些變數：

- 每個管線執行開始時有一組隱含可用的變數。這組變數目前包括 `PipelineExecutionId` (目前管道執行的 ID)。
- 管線層級的變數是在管線執行時建立和解析配管時定義的。

  您可以在建立配管時指定管線層級變數，並且可以在執行配管時提供值。
- 某些動作類型在執行時產生一組變數。您可以檢查屬於 ListActionExecutionsAPI 一部分的`outputVariables`欄位，查看動作所產生的變數。如需各動作提供者的可用索引鍵名稱的清單，請參閱管道動作可用的變數。若要查看每個動作類型所產生的變數，請參閱 CodePipeline 動作結構參考。

若要在動作組態中參考這些變數，您必須使用具有正確命名空間的變數參考語法。

如需變數工作流程範例，請參閱設定變數 。

命名空間

為了確保可唯一參考變數，必須將變數指派到一個命名空間。將一組變數指派到命名空間之後，即可在動作組態中使用命名空間和變數索引鍵來參考變數，語法如下：

```
#{namespace.variable_key}
```

有三種類型的命名空間下，可以分配變量：

- CodePipeline 保留的命名空間

  這是指派給每個管道執行開始時可用的隱含變數集的命名空間。這個命名空間是 codepipeline。變數參考範例：

  ```
  #{codepipeline.PipelineExecutionId}
  ```

- 在管道級別的變量命名空間

  這是指派給管線層級變數的命名空間。管線層級所有變數的命名空間為variables。變數參考範例：

  ```
  #{variables.variable_name}
  ```

- 動作指派的命名空間

  這是您指派給動作的命名空間。由動作產生的所有變數都屬於這個命名空間。若要讓動作所產生的變數可供下游動作組態中使用，您必須使用命名空間來設定產生動作。命名空間在整個管道定義中必須是唯一的，且不能與任何成品名稱衝突。以下是使用命名空間 SourceVariables 設定的動作的變數參考範例。

  ```
  #{SourceVariables.VersionId}
  ```

變數的使用案例

以下是管線層級變數的一些最常見使用案例，可協助您判斷如何針對特定需求使用變數。

- 管道層級的變數適用於希望每次使用相同管道的 CodePipeline 客戶，但動作組態的輸入有些微差異。任何啟動管道的開發人員都會在管道啟動時在 UI 中新增變數值。使用此組態時，您只會傳遞該執行的參數。
- 使用管線層級變數，您可以將動態輸入傳遞至管線中的動作。您可以將參數化管線移轉至，CodePipeline 而不必維護相同管線的不同版本，或建立複雜的管道。
- 您可以使用管線層級變數來傳遞輸入參數，讓您在每次執行時重複使用管線，例如當您要指定要部署到生產環境的版本時，就不必複製管線。

- 您可以使用單一管道將資源部署到多個建置和部署環境。例如，對於具有 CodeCommit 存放庫的管道，可以使用在管線層級傳遞和 CodeDeploy參數來完成從指定的分支 CodeBuild 和目標部署環境進行部署。

設定變數

您可以在配管層級或配管結構中的動作層級配置變數。

在管線層級配置變數

您可以在配管層級新增一或多個變數。您可以在 CodePipeline 動作的組態中參照此值。您可以在建立配管時新增變數名稱、預設值和說明。變量在執行時解析。

> **Note**
>
> 如果未在管線層級為變數定義預設值，則會將該變數視為必要。您必須在啟動管線時指定所有必要變數的取代，否則管線執行將會失敗並出現驗證錯誤。

您可以使用配管結構中的 variable 屬性在管線層級提供變數。在下列範例中，`Variable1`變數的值為`Value1`。

```
      "variables": [
           {
               "name": "Variable1",
               "defaultValue": "Value1",
               "description": "description"
           }
       ]
```

如需管線 JSON 結構中的範例，請參閱在中建立管線 CodePipeline。

如需在管線執行時傳送的管線層級變數的教學課程，請參閱。教學課程：使用管線層級變數

請注意，不支援在任何類型的 Source 動作中使用管線層級變數。

> **Note**
>
> 如果variables命名空間已用於管線中的某些動作，您必須更新動作定義，並為衝突的動作選擇另一個命名空間。

在動作層級配置變數

您可以宣告動作的命名空間，以設定動作來產生變數。動作必須已經是產生變數的動作提供者之一。否則，可用的變數是管道層級的變數。

您可以透過以下方式宣告命名空間：

- 在主控台的 Edit action (編輯動作) 頁面上，在 Variable namespace (變數命名空間) 中輸入命名空間。
- 在 JSON 管道結構的 namespace 參數欄位中輸入命名空間。

在此範例中，您可以使用名稱將namespace參數加入至 CodeCommit 來源動作SourceVariables。這會設定動作來產生可供該動作提供者使用的變數，例如 CommitId。

```
{
    "name": "Source",
    "actions": [
        {
            "outputArtifacts": [
                {
                    "name": "SourceArtifact"
                }
            ],
            "name": "Source",
            "namespace": "SourceVariables",
            "configuration": {
                "RepositoryName": "MyRepo",
                "BranchName": "mainline",
                "PollForSourceChanges": "false"
            },
            "inputArtifacts": [],
            "region": "us-west-2",
            "actionTypeId": {
                "provider": "CodeCommit",
```

```
                "category": "Source",
                "version": "1",
                "owner": "AWS"
            },
            "runOrder": 1
        }
    ]
},
```

接下來，您將下游動作設定為使用先前動作所產生的變數。作法如下：

- 在主控台的 Edit action (編輯動作) 頁面上，在動作組態欄位中輸入變數語法 (針對下游動作)。
- 在 JSON 管道結構的動作組態欄位中輸入變數語法 (針對下游動作)

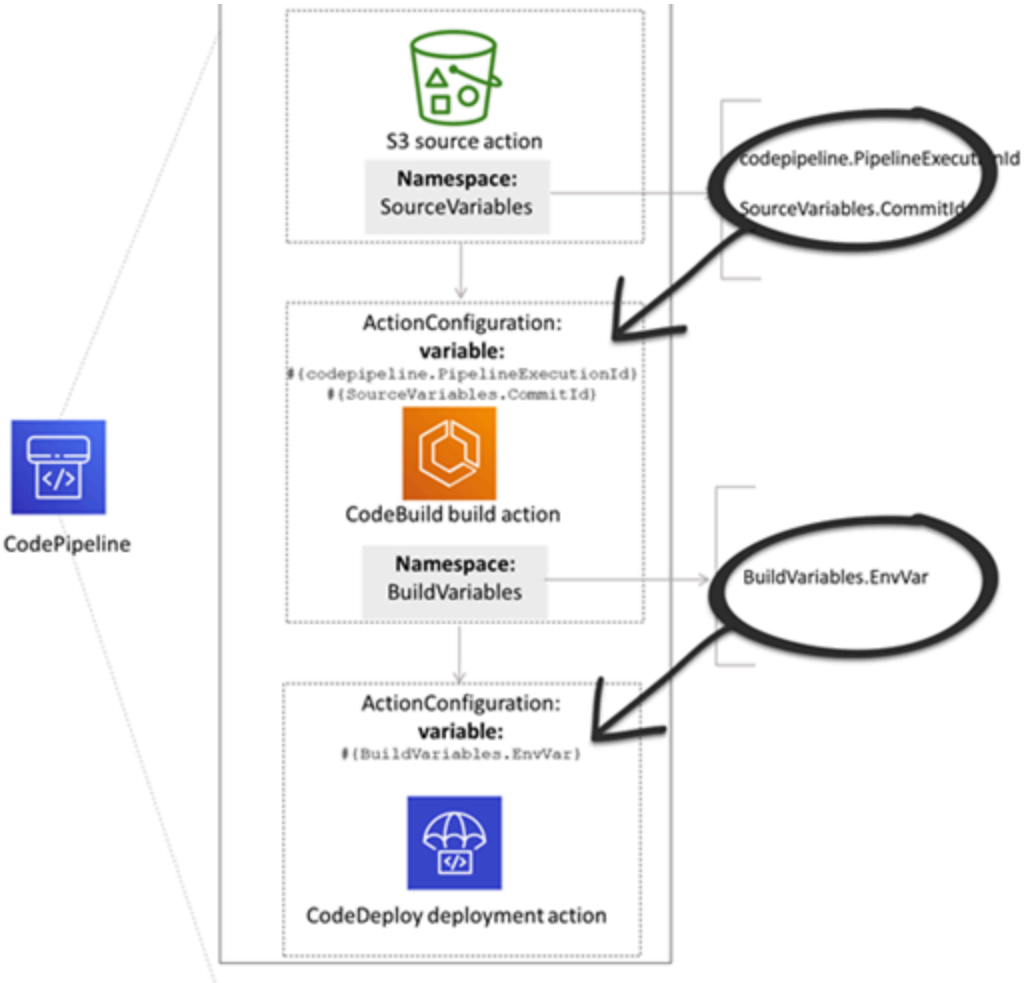
在此範例中，建置動作的組態欄位顯示動作執行時更新的環境變數。此範例以 `#{codepipeline.PipelineExecutionId}` 指定執行 ID 的命名空間和變數，以 **`#{SourceVariables.CommitId}`** 指定遞交 ID 的命名空間和變數。

```
{
    "name": "Build",
    "actions": [
        {
            "outputArtifacts": [
                {
                    "name": "BuildArtifact"
                }
            ],
            "name": "Build",
            "configuration": {
                "EnvironmentVariables": "[{\"name\":\"Release_ID\",\"value\":
\"#{codepipeline.PipelineExecutionId}\",\"type\":\"PLAINTEXT\"},{\"name\":\"Commit_ID
\",\"value\":\"#{SourceVariables.CommitId}\",\"type\":\"PLAINTEXT\"}]",
                "ProjectName": "env-var-test"
            },
            "inputArtifacts": [
                {
                    "name": "SourceArtifact"
                }
            ],
            "region": "us-west-2",
            "actionTypeId": {
                "provider": "CodeBuild",
```

```
                "category": "Build",
                "version": "1",
                "owner": "AWS"
            },
            "runOrder": 1
        }
    ]
},
```

變數解析

每次在管道執行中執行動作時，它產生的變數可用於產生動作之後保證發生的任何動作中。若要在取用動作中使用這些變數，您可以使用上一個範例所示的語法，將這些變數新增至取用動作的組態。在執行消費動作之前，請先 CodePipeline 解析組態中存在的所有變數參照，然後再啟動動作執行。



變數的規則

下列規則可協助您設定變數：

- 您可以透過新的動作屬性或編輯動作，指定動作的命名空間和變數。
- 當您使用管道建立精靈時，主控台會為精靈建立的每個動作產生命名空間。
- 如果未指定命名空間，則無法在任何動作組態中參考該動作產生的變數。
- 若要參考動作所產生的變數，參考動作必須在產生變數的動作之後發生。這意味著在比產生變數的動作更晚的階段，或在同一階段，但執行順序較高。

管道動作可用的變數

動作提供者決定哪些變數可以由動作產生。

如需管理變數的 step-by-step 程序，請參閱使用變數。

具有已定義變數鍵的動作

與您可以選擇的命名空間不同，下列動作使用無法編輯的變數索引鍵。例如，對於 Amazon S3 動作提供者，只有`ETag`和可`VersionId`變金鑰可用。

每個執行也有一組 CodePipeline產生的管線變數，其中包含執行相關資料，例如管線發行 ID。管道中的任何動作都可以取用這些變數。

主題

- CodePipeline執行 ID 變數
- Amazon ECR 動作輸出變量
- AWS CloudFormation StackSets 動作輸出變數
- CodeCommit 動作輸出變數
- CodeStarSourceConnection 動作輸出變數
- GitHub 動作輸出變數 (GitHub 動作版本 1)
- S3 動作輸出變數

CodePipeline執行 ID 變數

CodePipeline執行 ID 變數

供應商	變數索引鍵	範例值	範例變數語法
codepipeline	`PipelineExecutionId`	8abc75f0-fbf8-4f4c-bfEXAMPLE	`#{codepipeline.PipelineExecutionId}`

Amazon ECR 動作輸出變量

Amazon ECR 變量

變數索引鍵	範例值	範例變數語法
`ImageDigest`	sha256:EXAMPLE1122334455	`#{SourceVariables.ImageDigest}`
`ImageTag`	最新	`#{SourceVariables.ImageTag}`
`ImageURI`	11111EXAMPLE.dkr.ecr.us-west-2.amazonaws.com/ecs-repo:latest	`#{SourceVariables.ImageURI}`
`RegistryId`	EXAMPLE12233	`#{SourceVariables.RegistryId}`
`RepositoryName`	my-image-repo	`#{SourceVariables.RepositoryName}`

AWS CloudFormation StackSets 動作輸出變數

AWS CloudFormation StackSets 變數

變數索引鍵	範例值	範例變數語法
OperationId	例如	#{DeployVariables.OperationId}
StackSetId	我的堆棧：1111	#{DeployVariables.StackSetId}

CodeCommit 動作輸出變數

CodeCommit 變數

變數索引鍵	範例值	範例變數語法
AuthorDate	2019-10-29T03:32:21Z	#{SourceVariables.AuthorDate}
BranchName	開發	#{SourceVariables.BranchName}
CommitId	exampleb01f91b31	#{SourceVariables.CommitId}
CommitMessage	修正錯誤 (100 KB 大小上限)	#{SourceVariables.CommitMessage}
CommitterDate	2019-10-29T03:32:21Z	#{SourceVariables.CommitterDate}
RepositoryName	myCodeCommit回購	#{SourceVariables.RepositoryName}

CodeStarSourceConnection 動作輸出變數

CodeStarSourceConnection變量（比特桶雲 GitHub， GitHub企業存儲庫和 GitLab .com）

變數索引鍵	範例值	範例變數語法
AuthorDate	2019-10-29T03:32:21Z	#{SourceVariables.AuthorDate}
BranchName	開發	#{SourceVariables.BranchName}
CommitId	exampleb01f91b31	#{SourceVariables.CommitId}
CommitMessage	修正錯誤 (100 KB 大小上限)	#{SourceVariables.CommitMessage}
ConnectionArn	*arn: awn: #####:##:## ID: ##/## ID*	#{SourceVariables.ConnectionArn}
FullRepositoryName	用戶名/GitHubRepo	#{SourceVariables.FullRepositoryName}

GitHub 動作輸出變數 (GitHub 動作版本 1)

GitHub 變數 (GitHub 動作版本 1)

變數索引鍵	範例值	範例變數語法
AuthorDate	2019-10-29T03:32:21Z	#{SourceVariables.AuthorDate}
BranchName	主要	#{SourceVariables.BranchName}
CommitId	exampleb01f91b31	#{SourceVariables.CommitId}

變數索引鍵	範例值	範例變數語法
CommitMessage	修正錯誤 (100 KB 大小上限)	#{SourceVariables.CommitMessage}
CommitterDate	2019-10-29T03:32:21Z	#{SourceVariables.CommitterDate}
CommitUrl		#{SourceVariables.CommitUrl}
RepositoryName	myGitHub回購	#{SourceVariables.RepositoryName}

S3 動作輸出變數

S3 變數

變數索引鍵	範例值	範例變數語法
ETag	example28be1c3	#{SourceVariables.ETag}
VersionId	exampleta_IUQCv	#{SourceVariables.VersionId}

使用者設定之變數鍵的動作

對於 CodeBuild AWS CloudFormation、和 Lambda 動作，變數金鑰是由使用者設定的。

主題

- CloudFormation 動作輸出變數
- CodeBuild 動作輸出變數
- 動 Lambda 輸出變數

CloudFormation 動作輸出變數

AWS CloudFormation 變數

變數索引鍵	範例變數語法
對於 AWS CloudFormation 動作，變數是從堆疊範本`Outputs`區段中指定的任何值產生的。請注意，生成輸出的唯一 CloudFormation 動作模式是導致創建或更新堆棧的操作模式，例如堆棧創建，堆棧更新和更改集執行。產生變數的相應動作模式如下： • CREATE_UPDATE • CHANGE_SET_EXECUTE • CHANGE_SET_REPLACE • REPLACE_ON_FAILURE 如需這些動作模式的詳細資訊，請參閱AWS CloudFormation。如需說明如何在使用 AWS CloudFormation 輸出變數的管線中使用 AWS CloudFormation 部署動作建立管線的教學課程，請參閱教學：建立透過 AWS CloudFormation 部署動作使用變數的管道。	`#{DeployVariables.StackName}`

CodeBuild 動作輸出變數

CodeBuild 變數

變數索引鍵	範例變數語法
對於 CodeBuild 動作，變數是從匯出的環境變數產生的值產生的。透過在中編輯 CodeBuild動作 CodePipeline 或將 CodeBuild 環境變數新增至建構規格，來設定環境變數。 將指示添加到您的 CodeBuild 構建規範中，以在導出的變量部分下添加環境變量。請參閱《用戶指南》中的〈導出/導出變量〉。AWS CodeBuild	`#{BuildVariables.EnvVar}`

動 Lambda 輸出變數

Lambda 變量

變數索引鍵	範例變數語法
Lambda 動作會產生 PutJobSuccessResult API 要求`outputVariables` 區段中包含的所有索引鍵值組做為變數。 如需具有 Lambda 動作的教學課程，該動作使用上游動作 (CodeCommit) 中的變數並產生輸出變數，請參閱教學課程：使用變數搭配 Lambda 呼叫動作。	`#{TestVariables.testRunId}`

在語法中使用 glob 模式

當您指定用於管線人工因素或來源位置的檔案或路徑時，您可以根據動作類型指定人工因素。例如，對於 S3 動作，您可以指定 S3 物件金鑰。

對於觸發器，您可以指定篩選器。您可以使用 glob 模式來指定篩選器。範例如下。

當語法是「glob」，則使用有限模式語言與類似於正則表達式的語法來匹配路徑的字符串表示。例如：

- `*.java`指定代表以 .java 結尾的文件名的路徑
- `*.*`指定包含一個點的文件名
- `*.{java,class}`指定以 .java 或 .class 結尾的文件名
- `foo.?`指定檔案名稱以 foo. 和單一字元副檔名

以下規則用於解釋 glob 模式：

- 若要在目錄邊界中指定名稱元件的零個或多個字元，請使用*。
- 若要指定名稱元件跨越目錄邊界的零個或多個字元，請使用**。
- 若要指定名稱元件的一個字元，請使用?。
- 若要逸出要解譯為特殊字元的字元，請使用反斜線字元 (\)。
- 若要從一組字元中指定單一字元，請使用[ ]。
- 若要指定位於組建位置或來源存放庫位置根目錄中的單一檔案，請使用`my-file.jar`。
- 若要在子目錄中指定單一檔案，請使用`directory/my-file.jar`或`directory/subdirectory/my-file.jar`。
- 若要指定所有檔案，請使用"**/*"。**glob 模式指示匹配任意數量的子目錄。
- 若要指定名為的目錄中的所有檔案和目錄`directory`，請使用"directory/**/*"。**glob 模式指示匹配任意數量的子目錄。
- 若要指定名為的目錄中的所有檔案`directory`，但不指定其任何子目錄中的檔案，請使用"directory/*"。
- 括號內的*表示式?和\字元符合本身。如果 (-) 字元是括號內的第一個字元，或是 if 否定之後的第一個字元，就會!自行比對。
- 這些{ }字符是一組子模式，如果組中的任何子模式匹配，則該組匹配。該","字符用於分隔子模式。群組不能巢狀組合。

將輪詢管道更新至建議的變更偵測方法

如果您的管道使用輪詢來回應來源變更，您可以將其更新為使用建議的偵測方法。如需移轉指南，其中包含更新輪詢管線以使用建議的事件型變更偵測方法的指示，請參閱移轉輪詢管道以使用事件型變更偵測。

將 GitHub 版本 1 來源動作更新為 GitHub 版本 2 來源動作

在中AWS CodePipeline，有兩個受支援的 GitHub 來源動作版本：

- 建議：第 2 GitHub 版動作使用由CodeStarSourceConnection 適用於 Bitbucket 雲端 GitHub、 GitHub 企業伺服器、 GitLab .com 和 GitLab 自我管理動作資源支援的 Github 應用程式驗證。它會將AWS CodeStar連線應用程式安裝到您的 GitHub 組織中，以便您可以管理中的存取 GitHub。
- 不建議： GitHub 版本 1 操作使用 OAuth 令牌進行身份驗證， GitHub 並使用單獨的 webhook 來檢測更改。這不再是建議使用的方法。

> **Note**
>
> 亞太區域 (香港)、亞太區域 (海德拉巴)、亞太區域 (雅加達)、亞太區域 (墨爾本)、亞太區域 (大阪)、非洲 (開普敦)、中東 (巴林)、中東 (阿聯酋)、歐洲 (西班牙)、歐洲 (蘇黎世)、以色列 (特拉維夫) 或 AWS GovCloud (美國西部) 區域不提供轉機服務。若要參考其他可用動作，請參閱產品與服務整合 CodePipeline。如需在歐洲 (米蘭) 區域進行此動作的考量，請參閱中的附註CodeStarSourceConnection 適用於 Bitbucket 雲端 GitHub、 GitHub 企業伺服器、 GitLab .com 和 GitLab 自我管理動作。

使用 GitHub 版本 2 動作而非版本 1 動作有一些重要的優點： GitHub

- 使用連接， CodePipeline 不再需要 OAuth 應用程序或個人訪問令牌來訪問您的存儲庫。當您建立連線時，您會安裝一個 GitHub 應用程式，該應用程式會管理 GitHub 存放庫的驗證，並允許組織層級的權限。您必須將 OAuth 令牌作為用戶授權才能訪問存儲庫。如需有關基於 OAuth 的存取與應用程式 GitHub 存取不同的更多資訊，請參 GitHub 閱。https://docs.github.com/en/developers/apps/differences-between-github-apps-and-oauth-apps
- 當您在 CLI 中管理第 2 GitHub 版動作時 CloudFormation，或者您不再需要在 Secrets Manager 中將個人存取權杖儲存為密碼。您不再需要在動 CodePipeline 作配置中動態引用存儲的密碼。而是將連接 ARN 添加到您的操作配置中。如需動作組態範例，請參閱CodeStarSourceConnection 適用於 Bitbucket 雲端 GitHub、 GitHub 企業伺服器、 GitLab .com 和 GitLab 自我管理動作。
- 當您建立與中的第 2 GitHub 版動作搭配使用的連線資源時 CodePipeline，您可以使用相同的連線資源，將其他支援的服務 (例如 CodeGuru Reviewer) 與您的存放庫建立關聯。
- 在 Github 版本 2 中，您可以克隆存儲庫以在後續 CodeBuild 操作中訪問 git 元數據，而在 Github 版本 1 中，您只能下載源代碼。

- 系統管理員會為您組織的儲存庫安裝應用程式。您不再需要跟踪依賴於創建令牌的個人的 OAuth 令牌。

安裝到組織的所有應用程式都可以存取同一組儲存庫。若要變更可存取每個儲存庫的使用者，請修改每個連線的 IAM 政策。如需範例，請參閱範例：在指定存放庫中使用連線的範圍縮減原則。

您可以使用本主題中的步驟刪除 GitHub 版本 1 來源動作，並從主 CodePipeline 控台新增第 2 GitHub 版來源動作。

主題

步驟 1：取代您的版本 1 GitHub 動作

使用管線編輯頁面，將您的版本 1 GitHub 動作取代為第 2 版 GitHub 動作。

若要取代您的版本 1 GitHub 動作

1. 登入 CodePipeline 主控台。
2. 選擇您的管道，然後選擇 「編輯」。在來源階段中選擇「編輯階段」。會顯示一則訊息，建議您更新動作。
3. 在 [動作提供者] 中，選擇 [GitHub 版本 2]。
4. 執行以下任意一項：
   - 在 [連線] 下方，如果您尚未建立與提供者的連線，請選擇 [Connect 線至] GitHub。繼續執行「步驟 2：建立連線至 GitHub」。
   - 在 [連線] 下方，如果您已建立與提供者的連線，請選擇連線。繼續執行步驟 3：儲存連線的來源動作。

步驟 2：建立連線 GitHub

選擇建立連線之後，會顯示 [Connect 線至 GitHub] 頁面。

若要建立連線 GitHub

1. 在「GitHub 連線設定」 下，您的連線名稱會顯示在「連線名稱」中。

   在 [GitHub 應用程式] 下方，選擇應用程式安裝，或選擇 [安裝新的應用程式] 來建立

   > **Note**
   >
   > 您可以為您連至特定供應商的所有連線安裝一個應用程式。如果您已經安裝了該 GitHub 應用程序，請選擇它並跳過此步驟。

2. 如果 GitHub 顯示的授權頁面，請使用您的認證登入，然後選擇繼續。
3. 在應用程式安裝頁面上，會有訊息顯示AWS CodeStar應用程式正在嘗試連線到您的 GitHub 帳戶。

   > **Note**
   >
   > 您只能為每個 GitHub 帳戶安裝一次該應用程序。如果您先前已安裝應用程式，可以選擇 Configure (設定)，繼續前往應用程式安裝的修改頁面，或者您可以使用上一步按鈕返回主控台。

4. 在 [安裝 AWS CodeStar] 頁面上，選擇 [安裝]。
5. 在 [Connect 線至 GitHub] 頁面上，會顯示新安裝的連線 ID。選擇連線。

步驟 3：儲存您的 GitHub 來源動作

在 「編輯」動作頁面上完成更新，以儲存新的來源動作。

若要儲存 GitHub 來源動作

1. 在存放庫中，輸入第三方存放庫的名稱。在「分支」中，輸入您希望管線偵測來源變更的分支。

   > **Note**
   >
   > 在「存放庫」中，輸入`owner-name/repository-name`如下範例所示：
   >
   > ```
   > my-account/my-repository
   > ```

2. 在 [輸出成品格式] 中，選擇成品的格式。

   - 若要使用預設方法儲存 GitHub 動作的輸出成品，請選擇CodePipeline 預設值。此動作會從存 GitHub 放庫存取檔案，並將人工因素儲存在管線人工因素存放區中的 ZIP 檔案中。
   - 若要存放包含儲存庫 URL 參考的 JSON 檔案，以便下游動作可以直接執行 Git 命令，請選擇 Full clone (完整複製)。此選項只能由 CodeBuild 下游動作使用。

     如果您選擇此選項，則需要更新 CodeBuild 專案服務角色的權限，如中所示新增連線至 Bitbucket GitHub、 GitHub 企業伺服器或 GitLab .com 的 CodeBuild GitClone 權限。如需說明如何使用「完整複製」選項的教學課程，請參閱教學課程：搭 GitHub 配管線來源使用完整複製。

3. 在輸出人工因素中，您可以保留此動作的輸出成品名稱，例如`SourceArtifact`。選擇「完成」以關閉「編輯」動作頁面。
4. 選擇「完成」以關閉階段編輯頁面。選擇「儲存」以關閉配管編輯頁面。

AWS CodePipeline 中的配額

CodePipeline 擁有AWS帳戶在每個AWS區域中可以擁有的管道數量、階段、動作和 Webhook 數量的配額。以下配額適用每個區域，而且可以再提高。若要請求提高其上限，請使用支援中心主控台。

最多可能需要兩週時間來處理提高配額的請求。

資源	預設
動作逾時之前的時間長度 (這是可設定的逾時。 如需不可設定的逾時，請參閱下表)	AWS CloudFormation 部署動作：3 天 CodeDeploy 和 CodeDeploy ECS (藍/綠) 部署動作：5 天 AWS Lambda 呼叫動作：24 小時 **Note** 在動作執行時，請 CodePipeline 定期聯絡 Lambda 以取得狀態。Lambda 函數會回應動作執行成功、失敗或進行中的狀態。如果 Lambda 函數在 20 分鐘後沒有傳送回覆，則動作逾時。如果在 20 分鐘期間，Lambda 函數回覆動作仍在進行中，請 CodePipeline重新啟動 20 分鐘計時器，然後再試一次。如果 24 小時後仍未成功，請將 Lambda 叫用動作狀態 CodePipeline 設定為失敗。 Lambda 對於與 CodePipeline 動作逾時無關的 Lambda 函數有個別的逾時時間。 Amazon S3 部署動作：90 分鐘

資源	預設
	Note 如果上傳至 S3 在部署大型 ZIP 檔案期間逾時，動作會失敗，並顯示逾時錯誤。嘗試將 ZIP 文件分解為較小的文件。 手動核准作業帳號層級預設逾時：7 天 **Note** 您可以針對管道中的特定動作覆寫手動核准動作的預設逾時時間，最多可設定 86400 分鐘 (60 天)，最少值為 5 分鐘。如需詳細資訊，請參閱 CodePipeline API 參考ActionDeclaration中的。 設定後，此逾時會套用於動作。否則，會使用帳戶層級預設值。 所有其他動作：1 小時 **Note** Amazon ECS 部署動作逾時最多可設定一小時 (預設逾時)。

資源	預設
AWS 帳戶中每個區域總管道的數量上限	1000 **Note** 為輪詢或以事件為基礎的變更偵測而設定的管道將計入此配額。
每個AWS區域設定為輪詢來源變更的管線數目上限	300 **Note** 此配額是固定的，無法變更。如果達到輪詢管線的限制，您仍然可以設定使用事件型變更偵測的其他管道。如需詳細資訊，請參閱 來源動作和變更偵測方法。[1]
AWS 帳戶中每個區域的 Webhook 數目上限	300
AWS 帳戶中每個區域的自訂動作次數	50

[1]請根據您的來源提供商使用下列說明方式，更新您的輪詢管道以使用以事件為基礎的變更偵測：

- 若要更新來 CodeCommit 源動作，請參閱遷移輪詢管道 (CodeCommit 或 Amazon S3 來源) (主控台)。
- 若要更新 Amazon S3 來源動作，請參閱遷移輪詢管道 (CodeCommit 或 Amazon S3 來源) (主控台)。
- 若要更新來 GitHub 源動作，請參閱將輪詢管道遷移到 Webhook（GitHub 版本 1 源操作）（控制台）。

以下 AWS CodePipeline 中的配額適用於區域可用性、命名限制，以及允許的成品大小。這些配額是固定的，而且無法變更。

如需每個區域的 CodePipeline 服務端點清單，請參閱AWS一般參考中的AWS CodePipeline端點和配額。

如需結構需求的詳細資訊，請參閱 CodePipeline 配管結構參照。

AWS您可以在其中建立配管的區域	美國東部 (俄亥俄) 美國東部 (維吉尼亞北部) 美國西部 (加利佛尼亞北部) 美國西部 (奧勒岡) 加拿大 (中部) 歐洲 (法蘭克福) 歐洲 (蘇黎世) * 以色列 (特拉維夫) 歐洲 (愛爾蘭) 歐洲 (倫敦) 歐洲 (米蘭) * Europe (Paris) 歐洲 (西班牙) 歐洲 (斯德哥爾摩) 非洲 (開普敦) * 亞太區域 (香港) * 亞太區域 (海德拉巴) 亞太區域 (孟買) 亞太區域 (東京) 亞太區域 (首爾) 亞太區域 (大阪)

	亞太區域 (新加坡) 亞太區域 (雪梨) 亞太區域 (雅加達) 亞太區域 (墨爾本) 南美洲 (聖保羅) 中東 (巴林) * 中東 (阿拉伯聯合大公國) AWS GovCloud (美國西部) AWS GovCloud (美國東部)
動作名稱中允許的字元	動作名稱不能超過 100 個字元。允許的字元包含： 小寫字母 a 到 z (含)。 大寫字母 A 到 Z (含)。 數字 0 到 9，內含。 特殊字元 . (句點)、@ (@ 記號)、- (減號) 和 _ (底線)。 不允許任何其他字元 (例如空格)。

動作類型中允許的字元	動作類型名稱不能超過 25 個字元。允許的字元包含： 小寫字母 a 到 z (含)。 大寫字母 A 到 Z (含)。 數字 0 到 9 (含)。 特殊字元 . (句點)、@ (@ 記號)、- (減號) 和 _ (底線)。 不允許任何其他字元 (例如空格)。
人工因素名稱允許的字元	Artifact 名稱不能超過 100 個字元。允許的字元包含： 小寫字母 a 到 z (含)。 大寫字母 A 到 Z (含)。 數字 0 到 9，內含。 特殊字元 - (減號) 和 _ (底線)。 不允許任何其他字元 (例如空格)。
合作夥伴動作名稱中允許的字元	合作夥伴動作名稱必須遵循與 CodePipeline 中其他動作名稱相同的命名慣例和限制。尤其，它們不得超過 100 個字元。允許的字元包含： 小寫字母 a 到 z (含)。 大寫字母 A 到 Z (含)。 數字 0 到 9 (含)。 特殊字元 . (句點)、@ (@ 記號)、- (減號) 和 _ (底線)。 不允許任何其他字元 (例如空格)。

管道名稱中允許的字元	管道名稱不能超過 100 個字元。允許的字元包含： 小寫字母 a 到 z (含)。 大寫字母 A 到 Z (含)。 數字 0 到 9 (含)。 特殊字元 . (句點)、@ (@ 記號)、- (減號) 和 _ (底線)。 不允許任何其他字元 (例如空格)。
階段名稱中允許的字元	階段名稱不能超過 100 個字元。允許的字元包含： 小寫字母 a 到 z (含)。 大寫字母 A 到 Z (含)。 數字 0 到 9 (含)。 特殊字元 . (句點)、@ (@ 記號)、- (減號) 和 _ (底線)。 不允許任何其他字元 (例如空格)。
動作逾時之前的時間長度	CodeBuild 建立動作和測試動作：8 小時 自訂動作：24 小時 Step Functions 叫用動作：7 天
動作組態金鑰中的長度上限 (例如： CodeBuild 組態金鑰為 `ProjectName` 、`PrimarySource` 和 `EnvironmentVariables`)	50 個字元

動作配置值的最大長度（例如， CodeCommit 動作RepositoryName 配置中的配置值應小於 1000 個字符： `"RepositoryName": "my-repo-name-less-than-1000-characters" )`	1000 個字元
每個管道的動作數量上限	500
每個管線同時執行的管線數目上限 (QUEUEED|平行模式)	50
每個平行模式管線執行的最大並行動作執行數	5
Amazon S3 物件的最大檔案數	100,000
管道執行歷程記錄資訊保留的月份數上限	12
階段中平行動作的次數上限	50
階段中序列動作的次數上限	50
來源階段中成品的大小上限	存放在 Amazon S3 儲存貯體中的成品：7 GB 儲存在 CodeCommit 或儲存 GitHub 庫中的人工因素：1 GB 例外狀況：如果您使用 AWS Elastic Beanstalk 來部署應用程式，則成品大小上限一律為 512 MB。 例外狀況：如果您使用 AWS CloudFormation 來部署應用程式，則成品大小上限一律為 256 MB。 例外狀況：如果您使用 CodeDeployToECS 動作來部署應用程式，則成品大小上限一律為 3 MB。

管道部署 Amazon ECS 容器和映像中使用的影像定義 JSON 檔案大小上限	100 KB
AWS CloudFormation 動作的輸入成品大小上限	256 MB
`CodeDeployToECS` 動作的輸入成品大小上限	3 MB
可存放至 `ParameterOverrides` 屬性的 JSON 物件大小上限	對於作AWS CloudFormation為提供者的 CodePipeline 部署動作，該`ParameterOverrides` 屬性用於存儲 JSON 對象，該對象指定了AWS CloudFormation模板配置文件的值。能存放在 `ParameterOverrides` 屬性的 JSON 物件具有 1 KB 的最大大小限制。
階段中的動作次數	下限為 1，上限為 50
每個動作允許的成品數目	如需每個動作允許的輸入和輸出成品數目，請參閱 每個動作類型的輸入和輸出成品數目
管道中的階段數量	下限為 2，上限為 50
管道標籤	標籤會區分大小寫。每個資源的上限為 50。
管道標籤金鑰名稱	Unicode 字母、數字、空格，以及 UTF-8 與 1 之間允許字元的任何組合，長度為 128 個字元。允許的字元是 +、-、=、.、_、:、/、@ 標籤金鑰名稱必須是唯一的，而且每個金鑰只能有一個值。標籤不能： • 開始於AWS： • 只包含空格 • 以空格結尾 • 包含表情圖示或任何以下字元：?、^、*、[、\、~、!、#、$、%、&、*、(、)、>、<、|、"、'

管道標籤值	Unicode 字母、數字、空格，以及 UTF-8 與 1 之間允許字元的任何組合，長度為 256 個字元。允許的字元是 +、-、=、.、_、:、/、@ 金鑰只能有一個值，但多個金鑰可以有相同的值。標籤不能： • 開始於AWS： • 只包含空格 • 以空格結尾 • 包含表情圖示或任何以下字元：?、^、*、[、\、~、!、#、$、%、&、*、(、)、>、<、|、"、'
觸發	整個push和pull request組態的管線定義中最多有 50 個觸發程序。 每個推送觸發器和提取請求觸發器最多有三個篩選器。 Note 不允許同一事件類型陣列中的濾鏡重複項目。 您最多可以為每個事件類型（推送，拉取請求）添加 8 個 include 和 8 個排除模式，分支和文件路徑。 模式值中允許的字元包括所有字元類型。 對於包含和排除樣式，最大長度為 255 個字元。 對於標籤名稱，最大長度為 255 個字元。 triggers陣列的最大大小不應超過 200 KB

觸發濾波器	檔案路徑： • 圖案數量：您最多可以添加 8 個包含和 8 個排除模式。 • 圖樣大小：每個包含或排除圖樣的大小最多可以有 255 個字元。 分支機構: • 圖案數量：您最多可以添加 8 個包含和 8 個排除模式。 • 圖樣大小：每個包含或排除圖樣的大小最多可以有 255 個字元。 提取請求： 分支機構: • 圖案數量：您最多可以添加 8 個包含和 8 個排除模式。 • 圖樣大小：每個包含或排除圖樣的大小最多可以有 255 個字元。
名稱唯一性	在單一AWS帳戶中，您在「AWS區域」中建立的每個管線都必須具有唯一的名稱。您可以重複使用不同AWS區域中配管的名稱。 管道內的階段名稱必須是唯一的。 階段內的動作名稱必須是唯一的。

輸出變數和命名空間的配額	所有針對特定動作結合的輸出變數大小上限為 122880 位元。 特定動作以解決動作組態總大小上限為 100 KB。 輸出變量名稱有大小寫之分。 命名空間有大小寫之分。 允許的字元包含： • 小寫字母 a 到 z (含)。 • 大寫字母 A 到 Z (含)。 • 數字 0 到 9 (含)。 • 特殊字元 ^ (插入號)、@ (@ 記號)、- (減號)、_ (底線)、[(左括號)、] (右括號)、* (星號)、$ (貨幣符號)。 不允許任何其他字元 (例如空格)。
管線層級變數的配額	每個管線最多有 50 個管線層級變數。 管線層級變數的變數名稱必須是： • 長度上限為 128 個字元 • 小寫字母 a 到 z (含)。 • 大寫字母 A 到 Z (含)。 • 數字 0 到 9 (含)。 • 特殊字元 @\-_]+ 不允許任何其他字元 (例如空格)。 對於變數值，最大長度為 1000 個字元 對於變數值，則允許使用所有字元。 對於變數說明，最大長度為 200 個字元。

* 您必須先啟用此區域，才能使用它。

附錄 A：第 1 GitHub 版來源動作

本附錄提供中 GitHub 動作第 1 版的相關資訊 CodePipeline。

> **Note**
>
> 雖然我們不建議使用 GitHub 版本 1 動作，但具有第 1 GitHub 版動作的現有管道仍可繼續運作，而不會造成任何影響。對於具有 GitHub版本 1 操作的管道， CodePipeline 使用基於 OAuth 的 GitHub 令牌連接到存儲庫。相反地， GitHub 動作 (版本 2) 會使用連線資源將資 AWS 源與 GitHub 存放庫相關聯。連接資源使用基於應用程序的令牌進行連接。如需有關將管線更新為使用連線的建議 GitHub 動作的詳細資訊，請參閱將 GitHub 版本 1 來源動作更新為 GitHub 版本 2 來源動作。若要取得有關基於 OAuth 的存取 (與應用程式 GitHub 存取相比) 的更多資訊，請參 GitHub 閱。https://docs.github.com/en/developers/apps/differences-between-github-apps-and-oauth-apps

若要與整合 GitHub，請在您的管道中 CodePipeline 使用 GitHub OAuth 應用程式。 CodePipeline使用 webhook 透過第 1 GitHub 版來源動作管道管理變更偵測。

> **Note**
>
> 當您在中配置第 2 GitHub 版來源動作時 AWS CloudFormation，不會包含任何 GitHub 權杖資訊或新增 webhook 資源。您可以設定連線資源，如《 AWS CloudFormation 使用指南》AWS::CodeStarConnections::Connection中所示。

此參考包含第 1 GitHub 版動作的下列章節：

- 如需有關如何將 GitHub 版本 1 來源動作和 webhook 新增至管線的資訊，請參閱新增 GitHub 版本 1 來源動作。
- 如需第 1 GitHub 版來源動作的設定參數和範例 YAML/JSON 程式碼片段的相關資訊，請參閱。GitHub 版本 1 來源動作結構參考

主題

- 新增 GitHub 版本 1 來源動作
- GitHub 版本 1 來源動作結構參考

新增 GitHub 版本 1 來源動作

您可以將第 1 GitHub 版來源動作新 CodePipeline 增至：

- 使用主 CodePipeline 控台「建立管線精靈」(建立管道 (主控台)) 或 「編輯」動作頁面來選擇GitHub提供者選項。控制台創建一個 webhook，當源更改時啟動管道。
- 使用 CLI 新增動作的動作組態，並建立其他資源，如下所示：`GitHub`
  - 使用中的`GitHub`範例動作配置GitHub 版本 1 來源動作結構參考來建立動作，如中所示建立管道 (CLI)。
  - 停用定期檢查並手動建立變更偵測，因為變更偵測方法預設為透過輪詢來源來啟動管線。您可以將輪詢管線移轉至 Webhook 以進行第 1 GitHub 版動作。

GitHub 版本 1 來源動作結構參考

> **Note**
>
> 雖然我們不建議使用 GitHub 版本 1 動作，但具有第 1 GitHub 版動作的現有管道仍可繼續運作，而不會造成任何影響。對於具有 GitHub GitHub 版本 1 源動作的管道， CodePipeline 使用基於 OAuth 的令牌連接到存儲 GitHub 庫。相反地，新 GitHub 動作 (版本 2) 會使用連線資源將資 AWS 源與 GitHub 存放庫相關聯。連接資源使用基於應用程序的令牌進行連接。如需有關將管線更新為使用連線的建議 GitHub 動作的詳細資訊，請參閱將 GitHub 版本 1 來源動作更新為 GitHub 版本 2 來源動作。

在配置的 GitHub 存儲庫和分支上進行新的提交時觸發管道。

要與之整合 GitHub，請在管道中 CodePipeline 使用 OAuth 應用程式或個人存取權杖。如果您使用控制台來創建或編輯管道，請創 CodePipeline 建一個 GitHub webhook，以便在存儲庫中發生更改時啟動管道。

您必須先建立 GitHub 帳戶和存放庫，然後才能透過 GitHub 動作連接管道。

如果您想限制對存儲庫的訪問 CodePipeline 權限，請創建一個 GitHub 帳戶，並僅將該帳戶訪問權限授予您要與之集成的存儲庫 CodePipeline。當您設定為 CodePipeline 將 GitHub 存放庫用於管道中的來源階段時，請使用該帳戶。

如需詳細資訊，請參閱 GitHub 網站上的GitHub 開發人員文件。

主題

動作類型

- 類別：`Source`
- 擁有者：`ThirdParty`
- 提供者：`GitHub`
- 版本：`1`

組態參數

Owner

必要：是

擁有 GitHub存放庫的 GitHub 使用者或組織的名稱。

Repo

必要：是

要偵測來源變更的儲存庫名稱。

分支

必要：是

要偵測來源變更的分支名稱。

O AuthToken

必要：是

表示允許在 GitHub 存放庫上執 CodePipeline 行作業的 GitHub 驗證 Token。此項目永遠顯示為四個星號的遮罩。代表下列其中一個值：

- 當您使用主控台建立管道時，請 CodePipeline 使用 OAuth 權杖來註冊 GitHub 連線。
- 當您使用建立管道時，您可以在此欄位中傳遞您的 GitHub 個人存取權杖。 AWS CLI 用複製來源的個人訪問令牌替換星號（ **** ）。 GitHub當您執行 `get-pipeline` 以檢視動作組態時，會為此數值顯示四星號遮罩。
- 當您使用 AWS CloudFormation 範本建立管道時，必須先將權杖作為密碼儲存在中 AWS Secrets Manager。您可以將此欄位的值納入為 Secret Manager 中已儲存密碼的動態參照，例如`{{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}`。

有關 GitHub 範圍的更多信息，請參閱 GitHub 網站上的GitHub 開發人員 API 參考。

PollForSourceChanges

必要：否

`PollForSourceChanges`控制是否 CodePipeline 輪詢 GitHub存放庫中的來源變更。我們建議您改用 webhook 以偵測來源變更。如需有關設定 webhook 的詳細資訊，請參閱將輪詢管線移轉至 Webhook (第 1 GitHub 版來源動作) (CLI) 或更新推送事件的管道 (第 1 GitHub 版來源動作) (AWS CloudFormation範本)。

> **Important**
>
> 如果您想要設定 webhook，則必須將 `PollForSourceChanges` 設定為 `false`，以避免管道執行重複。

此參數的有效值：

- `True`：如果設置，則 CodePipeline 輪詢您的存儲庫以進行源更改。

> **Note**
>
> 如果省略`PollForSourceChanges`，則 CodePipeline 預設會輪詢儲存庫以進行來源變更。此行為同於 `PollForSourceChanges` 設定為 `true`。

- `False`：如果設定，則 CodePipeline 不會輪詢您的儲存庫是否有來源變更。如果您想要設定 webhook 以偵測來源變更，請使用此設定。

Input artifacts (輸入成品)

- 人工因素數目：`0`
- 描述：輸入成品不適用於此動作類型。

輸出成品

- 人工因素數目：`1`
- 描述：此動作的輸出成品是 ZIP 檔案，其中包含在指定為管道執行來源修訂的遞交上所設定的儲存庫和分支的內容。從存放庫產生的成品是 GitHub 動作的輸出成品。中的原始程式碼提交 ID 會顯示 CodePipeline 為觸發管線執行的來源修訂。

輸出變數

設定時，此動作會產生變數，供管道中的下游動作的動作組態所參考。即使此動作沒有命名空間，此動作產生的變數仍可視為輸出變數。您可以設定動作的命名空間，讓這些變數可供下游動作的組態使用。

如需中變數的更多資訊 CodePipeline，請參閱Variables。

CommitId

觸發管線執行的 GitHub 提交 ID。遞交 ID 是遞交的完整 SHA。

CommitMessage

與觸發管道執行的遞交相關聯的描述訊息 (如果有的話)。

CommitUrl

觸發管道的遞交的 URL 位址。

RepositoryName

執行觸發管線之提交的 GitHub 儲存庫名稱。

BranchName

進行來源變更之 GitHub 儲存庫的分支名稱。

AuthorDate

遞交的撰寫日期 (時間戳記格式)。

如需 Git 中的作者和遞交者之間差異的詳細信息，請參閱 Scott Chacon 和 Ben Straub 共同撰寫的 Pro Git 中的檢視提交的歷史記錄。

CommitterDate

遞交的遞交日期 (時間戳記格式)。

如需 Git 中的作者和遞交者之間差異的詳細信息，請參閱 Scott Chacon 和 Ben Straub 共同撰寫的 Pro Git 中的檢視提交的歷史記錄。

動作宣告 (GitHub 範例)

YAML

```
Name: Source
Actions:
  - InputArtifacts: []
    ActionTypeId:
      Version: '1'
      Owner: ThirdParty
      Category: Source
      Provider: GitHub
    OutputArtifacts:
      - Name: SourceArtifact
    RunOrder: 1
    Configuration:
      Owner: MyGitHubAccountName
      Repo: MyGitHubRepositoryName
      PollForSourceChanges: 'false'
      Branch: main
      OAuthToken: '{{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}'
    Name: ApplicationSource
```

JSON

```
{
    "Name": "Source",
    "Actions": [
```

```
        {
            "InputArtifacts": [],
            "ActionTypeId": {
                "Version": "1",
                "Owner": "ThirdParty",
                "Category": "Source",
                "Provider": "GitHub"
            },
            "OutputArtifacts": [
                {
                    "Name": "SourceArtifact"
                }
            ],
            "RunOrder": 1,
            "Configuration": {
                "Owner": "MyGitHubAccountName",
                "Repo": "MyGitHubRepositoryName",
                "PollForSourceChanges": "false",
                "Branch": "main",
                "OAuthToken":
 "{{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}"
            },
            "Name": "ApplicationSource"
        }
    ]
},
```

連線到 GitHub (OAuth)

第一次使用控制台將 GitHub 存儲庫添加到管道時，系統會要求您授權存儲庫的 CodePipeline 訪問權限。令牌需要以下 GitHub 範圍：

- `repo` 範圍，用於完全控制將成品從公有和私有儲存庫讀取和提取至管道。
- `admin:repo_hook` 範圍，用於完全控制儲存庫勾點。

使用 CLI 或 AWS CloudFormation 範本時，您必須為已在中建立的個人存取權杖提供值 GitHub。

另請參閱

以下相關資源可協助您使用此動作。

- 《AWS CloudFormation 使用者指南》的資源參考 AWS::CodePipeline::Webhook — 包括中資源的欄位定義、範例和片段 AWS CloudFormation。
- AWS CloudFormation 使用者指南 AWS::CodeStar::GitHub儲存庫的資源參考 — 包括中資源的欄位定義、範例和程式碼片段 AWS CloudFormation。
- 教程：創建一個管道，用於構建和測試您的 Android 應用程序 AWS Device Farm— 本教學課程提供範例建置規格檔案和範例應用程式，以建立具有 GitHub 來源的管道。它使用和構建和測試一個 Android 應用 CodeBuild 程序 AWS Device Farm。

AWS CodePipeline 使用者指南文件記錄

下表描述了《 CodePipeline 用戶指南》每個發行版本中的重要變更。如需有關此文件更新的通知，您可以訂閱 RSS 訂閱源。

- API 版本：2015-07-09
- 最新文件更新：2024 年 3 月 15 日

變更	描述	日期
受管理策略的更新	受 AWS 管理的策略`AWSCodePipeline_FullAccess` 已更新。請參閱的AWS 受管政策 AWS CodePipeline。	2024年3月15日
Support 手動核准動作的可設定逾時	為手動核准動作的新設定逾時欄位新增配額資訊。如需詳細資訊，請參閱 配額。	2024年2月15日
Support 按分支和文件路徑進行觸發過濾	增加了觸發器配置的 Support，允許過濾 V2 類型管道的提取請求狀態，分支和文件路徑。如需詳細資訊，請參閱篩選程式碼推送或提取要求上的觸發程序、觸發程序和篩選功能分支以啟動管道和配額。	2024年2月8日
Support 新的管線執行模式	已新 Support 平行和佇列管線執行模式的支援。如需詳細資訊，請參閱設定管線執行模式、以 QUEED 模式處理執行的方式、以平行模式處理執行的方式和配額。	2024年2月8日

用於檢視動作詳細資訊、檢閱手動核准動作和清單管道頁面的主控台頁面的更新	主控台更新記錄的新檢視詳細資料按鈕和對話方塊、新的手動核准對話方塊，以及清單管線頁面上最近執行的新資料行。如需詳細資訊，請參閱檢視管線 (主控台) 、檢視管線中的動作詳細資訊和管理管道中的核准動作。	2024 年 1 月 10 日
Support GitLab 自我管理	已新 Support 設定 AWS 資源連線以與 GitLab自我管理互動的支援。如需詳細資訊，請參閱 GitLab自我管理的連線。	2023 年 12 月 28 日
CloudFormationStackSet 和CloudFormationStackInstances 動作的更新	已針對CloudFormationStackSet 和CloudFormationStackInstances 動作加入ConcurrencyMode 參數。請參閱動作參考頁面。	2023 年 12 月 19 日
中 AWS Device Farm 動作參數的更新 CodePipeline	中 AWS Device Farm 動作的參數 CodePipeline 已更新。如需詳細資訊，請參閱AWS Device Farm 動作參考。	2023 年 12 月 18 日
為中的操作添加了詳細錯誤消息 AWS CloudFormation 的 Support CodePipeline	AWS CloudFormation 動作錯誤訊息現在可以顯示失敗資源的詳細資料。如需詳細資訊，請參閱AWS CloudFormation 動作參考。	2023 年 12 月 15 日
使用來源修訂版取代啟動管線的更新 CodePipeline	您現在可以使用指定的來源修訂版來啟動管線。如需詳細資訊，請參閱以來源修訂版本取代啟動管線。	2023 年 11 月 17 日

全新支援的地區	CodePipeline 現已在亞太區域 (海德拉巴)、亞太區域 (雅加達)、亞太區域 (墨爾本)、亞太區域 (大阪)、中東 (阿聯酋)、歐洲 (西班牙) 和以色列 (特拉維夫) 等區域推出。事件預留位置值區參考主題和AWS 服務端點主題已更新。	2023 年 11 月 13 日
Amazon 中事件字段的更新 EventBridge	現在，您可以在 Amazon 中查看更新的事件字段 EventBridge。如需詳細資訊，請參閱監視 CodePipeline事件。	2023 年 11 月 9 日
新管線類型 V2 管線、Git 標籤上的觸發程序以及管線變數的更新 CodePipeline	您現在可以在中選擇配管類型 CodePipeline。對於 V2 類型管道，您現在可以使用觸發器配置在 Git 標籤上啟動管道。對於 V2 類型管線，您也可以在管線層級使用變數來傳遞輸入參數以執行管線。如需詳細資訊，請參閱變數、教學課程：使用管線層級變數和教學課程：使用 Git 標籤啟動管線。如需有關配管類型的詳細資訊，請參閱配管類型。	2023 年 10 月 24 日
CodePipeline 允許重試失敗階段中的所有動作	對於中失敗的階段 CodePipeline，您可以在不重新執行管線的情況下重試該階段。您可以透過在階段中重試失敗的動作，或重試階段中的所有動作，從階段中的第一個動作開始。如需詳細資訊，請參閱重試階段中失敗的階段或失敗的動作。	2023 年 10 月 17 日

對 GitLab 團體的 Support	已新 Support 設定 AWS 資源連線以與 GitLab群組互動的支援。如需詳細資訊，請參閱GitLab 連線。	2023 年 9 月 15 日
CodePipeline 支援連線至 GitLab .com	您可以使用連線來設定要與 GitLab .com 互動的 AWS 資源。您也可以選擇完整複製選項，以便將 Git 命令和中繼資料用於下游動作。如需詳細資訊，請參閱GitLab 連線與CodeStarSourceConnection 動作結構參照主題。	2023 年 8 月 10 日
更新到操CloudFormationStackInstances 作	已針對CloudFormationStackInstances 動作新增RegionConcurrencyType 參數。請參閱動作的動CloudFormationStackInstances 作參考頁面。	2023 年 8 月 8 日
更新到操CloudFormationStackSet 作	已針對CloudFormationStackSet 動作新增RegionConcurrencyType 參數。請參閱動作的動CloudFormationStackSet 作參考頁面。	2023 年 7 月 24 日
受管理策略的更新	受 AWS 管理的策略AWSCodePipeline_FullAccess 已更新。請參閱的AWS 受管政策 AWS CodePipeline。	2023 年 6 月 21 日

輪詢管道的移轉程序更新	遷移 (更新) 輪詢管道以使用事件型變更偵測的程序已更新為使用 Amazon S3 儲存貯體啟用通知的管道步驟。 EventBridge如需詳細資訊，請參閱移轉輪詢管道以使用事件型變更偵測。	2023 年 6 月 12 日
受管理策略的更新	受 AWS 管理的策略，AWSCodePipeline_FullAccess 並AWSCodePipeline_ReadOnlyAccess 已使用其他權限進行更新。如需詳細資訊，請參閱AWS 受管理策略的更AWS CodePipeline 新。	2023 年 5 月 16 日
受管理策略的更新	受 AWS 管理的政策AWSCodePipelineFullAccess 和AWSCodePipelineReadOnlyAccess 已取代。使用AWSCodePipeline_FullAccess 和AWSCodePipeline_ReadOnlyAccess 策略。請參閱AWS 受管理策略的AWS CodePipeline 更新。	2022 年 11 月 17 日

使用的程序更新 CloudTrail	具有 S3 來源之管道的所有主控台程序、CLI 命令範例，以及範例程 AWS CloudFormation 式碼片段和範本都已更新，並在中選擇 [寫入] 選項，然後針對中的管理事件選取 false CloudTrail。請參閱啟動管線、教學課程：使用建立管線 AWS CloudFormation、編輯管道以使用推送事件和更新輪詢管線中的更新範例。	2022 年 4 月 27 日
與 Snyk 的新支持集成	您可以使用中 CodePipeline 的 Snyk invoke 動作來自動執行開放原始碼的安全性掃描。有關更多信息，請參閱 S nyk 操作參考和集成。	2021 年 6 月 10 日
新支援的地區歐洲 (米蘭)	CodePipeline 現已在歐洲 (米蘭) 推出。「限制」主題和AWS 服務 端點主題已更新。	2021 年 1 月 27 日
可以關閉具有連線的來源動作的變更偵測	您可以使用 CLI 或 SDK 更新`CodeStarSourceConnection` 來源動作，以關閉來源存放庫的自動變更偵測。CodeStarSourceConnection 動作結構參照主題已更新為`DetectChanges` 參數的描述。	2021 年 1 月 8 日

CodePipeline 現在支援 AWS CloudFormation StackSets 部署動作	新的教學課程「教學課程：建立用 AWS CloudFormation StackSets 作部署提供者的管道」，提供了使 AWS CloudFormation StackSets 用管道建立和更新堆疊集和堆疊執行個體的步驟。動AWS CloudFormation StackSets 作結構參照主題也已新增。	2020 年 12 月 30 日
新增支援的地區亞太區域 (香港)	CodePipeline 現已於亞太區域（香港）發售。「限制」主題和AWS 服務 端點主題已更新。	2020 年 12 月 22 日
檢視更新的 EventBridge 事件模式 CodePipeline	管道、階段和動作層級事件的更新事件模式和狀態已新增至監控 CodePipeline 事件。	2020 年 12 月 21 日
檢視輸入管線執行 CodePipeline	您可以使用主控台或 CLI 來檢視輸入執行。如需詳細資訊，請參閱檢視輸入執行 (主控台)和檢視輸入執行狀態 (CLI)。	2020 年 11 月 16 日
中的 CodeCommit 來源動作 CodePipeline 支援完整複製選項	當您使用 CodeCommit 來源動作時，您可以選擇完整複製選項，以便將 Git 命令和中繼資料用於下游 CodeBuild 動作。如需詳細資訊，請參閱CodeCommit 動作參考和教學課程：搭 CodeCommit 配管線來源使用完整複製。	2020 年 11 月 11 日

CodePipeline 支援 GitHub 與 GitHub 企業伺服器的連線	您可以使用連線來設定與 GitHub GitHub企業雲端和 GitHub 企業伺服器互動的 AWS 資源。您也可以選擇完整複製選項，以便將 Git 命令和中繼資料用於下游動作。如需詳細資訊，請參閱GitHub 連線、GitHub 企業伺服器連線和教學課程：搭 GitHub 配管線來源使用完整複製。如果您有具有 GitHub來源動作的現有管道，請參閱將 GitHub 版本 1 來源動作更新為 GitHub 版本 2 來源動作。	2020 年 9 月 30 日
此動 CodeBuild 作支援啟用批次建置 AWS CodePipeline	對於管線中的 CodeBuild 動作，您可以啟用批次建置，在單一執行中執行多個組建。如需詳細資訊，請參閱CodeBuild 動作結構參照和建立管線 (主控台)。	2020 年 7 月 30 日
AWS CodePipeline 現在支援 AWS AppConfig 部署動作	新的教學課程「教學課程：建立用 AWS AppConfig 作部署提供者的管線」，提供了在管線中部署組態檔時使用 AWS AppConfig 的步驟。動AWS AppConfig 作結構參照主題也已新增。	2020 年 6 月 25 日
AWS CodePipeline 現在支援 Amazon VPC AWS GovCloud (美國西部)	您現在可以 AWS CodePipeline 透過 AWS GovCloud (美國西部) 中的私有 Amazon VPC 端點直接連線到。如需詳細資訊，請參閱搭 CodePipeline 配 Amazon Virtual Private Cloud 使用。	2020 年 6 月 2 日

AWS CodePipeline 現在支持 AWS Step Functions 調用操作	您現在可以在中建立用作叫 CodePipeline 用 AWS Step Functions 動作提供者的管道。新的教學課程「教學課程：在管線中使用 AWS Step Functions 叫用動作」會提供從管線啟動狀態機器執行的步驟。當中也新增了 AWS Step Functions 動作結構參考主題。	2020 年 5 月 28 日
檢視、列出和更新連線	您可以在主控台中列出、刪除和更新連線。請參閱列出中的連線 CodePipeline。	2020 年 5 月 21 日
連線支援在 CLI 中標記連線資源	連線資源現在支援 AWS CLI 中進行標記。連接現在與 AWS CodeGuru. 請參閱連線的 IAM 許可參考。	2020 年 5 月 6 日
CodePipeline 目前可在 AWS GovCloud (美國西部)	您現在可以 CodePipeline 在 AWS GovCloud （美國西部）中使用。如需詳細資訊，請參閱 配額。	2020 年 4 月 8 日
配額主題會顯示可設定的 CodePipeline 服務配額	CodePipeline 配額主題已重新格式化。此文件說明哪些服務配額是可設定的，以及哪些配額是無法設定的。請參閱配額於 AWS CodePipeline.	2020 年 3 月 12 日
可設定 Amazon ECS 部署動作逾時	Amazon ECS 部署動作逾時最多可設定一小時 (預設逾時)。請參閱 AWS 中的配額CodePipeline。	2020 年 2 月 5 日

新主題說明如何停止管線執行	您可以在 CodePipeline 中停止管道執行。您可以指定允許在進行中的動作之後停止的執行完成，也可以指定立即停止執行並捨棄進行中的動作。請參閱如何停止管道執行和在 CodePipeline 中停止管道執行。	2020 年 1 月 21 日
CodePipeline 支援連線	您可以使用連線來設定 AWS 資源，以便與外部程式碼儲存庫互動。每個連接都是可供諸如連接 CodePipeline 到第三方存儲庫（例如 Bitbucket Cloud）之類的服務使用的資源。如需詳細資訊，請參閱中的使用連接 CodePipeline。	2019 年 12 月 18 日
更新的安全性、驗證和存取控制主題	的安全性、驗證和存取控制資訊 CodePipeline 已整理成新的「安全性」一章。如需詳細資訊，請參閱安全性。	2019 年 12 月 17 日
新主題說明如何在管道中使用變數	您現在可以為動作設定命名空間，在每次動作執行完成時產生變數。您可以設定下游動作來參考這些命名空間和變數。請參閱使用變數和變數。	2019 年 11 月 14 日

新主題說明管線執行的運作方式、階段在執行期間鎖定的原因，以及取代管線執行的時間	「歡迎使用」小節新增許多主題，說明管道執行的運作方式，包括執行期間鎖定階段的原因，以及接替管線執行時會發生什麼情況。這些主題包括概念清單、 DevOps工作流程範例，以及如何建立管線的建議。已新增下列主題：管線術語、DevOps 管線範例以及管線執行的運作方式。	2019 年 11 月 11 日
CodePipeline 支援通知規則	您現在可以使用通知規則，向使用者通知管道中有重要變更。如需詳細資訊，請參閱建立通知規則。	2019 年 11 月 5 日
CodeBuild 可用的環境變數 CodePipeline	您可以在管道的 CodeBuild 建置動作中設定 CodeBuild 環境變數。您可以使用主控台或 CLI 將 `EnvironmentVariables` 參數新增至管道結構。已更新建立管道 (主控台) 主題。CodeBuild 動作參考中的動作組態範例也已更新。	2019 年 10 月 14 日
新區域	CodePipeline 現已在歐洲 (斯德哥爾摩) 推出。「限制」主題和AWS 服務 端點主題已更新。	2019 年 9 月 5 日

為 Amazon S3 部署動作指定固定 ACL 和快取控制	您現在可以在中建立 Amazon S3 部署動作時指定固定 ACL 和快取控制選項 CodePipeline。下列主題已更新：建立管道 (主控台)、CodePipeline 管道結構參考和教學課程：建立使用 Amazon S3 做為部署供應商的管道。	2019 年 6 月 27 日
您現在可以將標籤新增至 AWS CodePipeline	您現在可以使用標記來追蹤和管理 AWS CodePipeline 資源，例如管線、自訂動作和 Webhook。已新增下列新主題：標記資源、使用標籤控制 CodePipeline 資源存取、在中標記管線 CodePipeline、在中標記自訂動作 CodePipeline，以及在中標記 webhook。CodePipeline下列主題已更新，顯示如何使用 CLI 標記資源：建立管線 (CLI)、建立自訂動作 (CLI) 和建立 GitHub來源的 webhook。	2019 年 5 月 15 日
您現在可以查看操作執行歷史記錄 AWS CodePipeline	您現在可以檢視在管道中過去執行的所有動作的詳細資訊。這些詳細資訊包括開始和結束時間、持續時間、動作執行 ID、狀態、輸入和輸出成品位置詳細資訊，以及外部資源詳細資訊。檢視管道詳細資訊和歷程記錄主題已更新，以反映此項支援。	2019 年 3 月 20 日

AWS CodePipeline 現在支援將應用程式發佈到 AWS Serverless Application Repository	您現在可以在中建立一個管道 CodePipeline ，將無伺服器應用程式發佈到 AWS Serverless Application Repository. 新的教學課程〈教學課程：將應用程式發佈到〉提供建立和設定管道的步驟 AWS Serverless Application Repository，以便持續將無伺服器應用程式傳遞至. AWS Serverless Application Repository	2019 年 3 月 8 日
AWS CodePipeline 現在支援主控台中的跨區域動作	您現在可以在 AWS CodePipeline 主控台中管理跨區域動作。「新增跨區域」動作已更新，其中包含新增、編輯或刪除管線中不同 AWS 區域中動作的步驟。建立管道、編輯管道和 CodePipeline 管道結構參考主題已更新。	2019 年 2 月 14 日
AWS CodePipeline 現在支援 Amazon S3 部署	您現在可以在中建立使 CodePipeline 用 Amazon S3 做為部署動作提供者的管道。新的教學課程：建立使用 Amazon S3 做為部署供應商的管道，提供使用將範例檔案部署到 Amazon S3 儲存貯體的步驟 CodePipeline。也已更新 CodePipeline 管道結構參考主題。	2019 年 1 月 16 日

AWS CodePipeline 現在支援 Alexa 技能套件部署	您現在可以使用 CodePipeline 和 Alexa 技能套件來持續部署 Alexa 技能。新教學課程：建立可部署 Amazon Alexa 技能的管道，其中包含建立登入資料的步驟，這些登入資料允許連線 AWS CodePipeline 至您的 Alexa S技能Kit 開發人員帳戶，然後建立管道以部署範例技能。已更新 CodePipeline 管道結構參考主題。	2018 年 12 月 19 日
AWS CodePipeline 現在支援採用的 Amazon VPC 端點 AWS PrivateLink	您現在可以 AWS CodePipeline 透過 VPC 中的私有端點直接連線，將所有流量保留在 VPC 和網路內。 AWS 如需詳細資訊，請參閱搭 CodePipeline 配 Amazon Virtual Private Cloud 使用。	2018 年 12 月 6 日
AWS CodePipeline 現在支援 Amazon ECR 來源動作和 ECS 到部署動作 CodeDeploy	您現在可以使用 CodePipeline 和 CodeDeploy 搭配 Amazon ECR 和 Amazon ECS 來持續部署以容器為基礎的應用程式。新的教學課程使用 Amazon ECR 來源和 ECS-to CodeDeploy 部署建立管道，其中包含使用主控台建立管道的步驟，將存放在映像儲存庫中的容器應用程式部署到具有流量路由的 Amazon ECS 叢集。 CodeDeploy 建立管道和 CodePipeline 管道結構參考主題已更新。	2018 年 11 月 27 日

AWS CodePipeline 現在支援管道中的跨區域動作	新主題「新增跨區域動作」包含使用 AWS CLI 或 AWS CloudFormation 新增與管線不同區域中的動作的步驟。建立管道、編輯管道和 CodePipeline 管道結構參考主題已更新。	2018 年 11 月 12 日
AWS CodePipeline 現在與 Service Catalog 整合	您現在可以將 Service Catalog 作為部署動作新增至管線。這可讓您設定管道，以便在來源儲存庫中進行變更時，將產品更新發佈至 Service Catalog。整合主題已更新，以反映此對 Service Catalog 的支援。教學課程區段中已加入兩個 Service Catalog AWS CodePipeline 教學課程。	2018 年 10 月 16 日
AWS CodePipeline 現在整合了 AWS Device Farm	您現在可以將 AWS Device Farm 測試動作新增至管道。這可讓您設定管道來測試行動應用程式。「整合」主題已更新，以反映此支援 AWS Device Farm。 AWS Device Farm 自學課程區段中已加入兩個AWS CodePipeline 自學課程。	2018 年 7 月 19 日

AWS CodePipeline 使用者指南更新通知現在可透過 RSS 取得	CodePipeline 使用者指南的 HTML 版本現在支援「文件更新歷程記錄」頁面中記錄的更新 RSS 摘要。RSS 摘要包含 2018 年 6 月 30 日以後所做的更新。先前發佈的更新仍可在 Documentation Update History (文件更新歷史記錄) 頁面中取得。使用頂部選單面板中的 RSS 按鈕來訂閱摘要。	2018 年 6 月 30 日

舊版更新

下表說明 2018 年 6 月 30 日及更早版本的「 CodePipeline 使用者指南」發行版本中的重要變更。

變更	描述	變更日期
使用 Webhook 來偵測管線中 GitHub 的來源變更	當您在控制台中創建或編輯管道時， CodePipeline 現在會創建一個 webhook 來檢測 GitHub 源存儲庫的更改，然後啟動管道。如需移轉管線的相關資訊，請參閱將 GitHub管線設定為使用 Webhook 進行變更偵測。如需詳細資訊，請參閱在 CodePipeline 中啟動管道執行。	2018 年 5 月 1 日
更新主題	在主控台中建立或編輯管道時， CodePipeline 現在會建立 Amazon E CloudWatch vents 規則和 AWS CloudTrail 追蹤，以偵測 Amazon S3 來源儲存貯體的變更，然後啟動管道。如需遷移管道的資訊，請參閱 來源動作和變更偵測方法。 教學：建立簡易管道 (S3 儲存貯體)已更新，顯示當您選取 Amazon S3 來源時，如何建立 Amazon CloudWatch 活動規則和追蹤。 在中建立管線 CodePipeline並且也在中編輯配管 CodePipeline已更新。 如需詳細資訊，請參閱 在中啟動管道 CodePipeline。	2018 年 3 月 22 日

變更	描述	變更日期
更新主題	CodePipeline 現已在歐洲 (巴黎) 推出。已更新 AWS CodePipeline 中的配額主題。	2018 年 2 月 21 日
更新主題	您現在可以使用 CodePipeline 和 Amazon ECS 來持續部署容器型應用程式。建立管道時，您可以選取 Amazon ECS 做為部署供應商。原始碼控制儲存庫中的程式碼變更會觸發管道以建立新的 Docker 映像、將其推送到容器登錄，然後將更新的映像部署到 Amazon ECS 服務。 主題和CodePipeline 配管結構參照已更新 產品與服務整合 CodePipeline在中建立管線 CodePipeline，以反映此對 Amazon ECS 的支援。	2017 年 12 月 12 日
更新主題	在主控台中建立或編輯管道時， CodePipeline 現在會建立 Amazon E CloudWatch vents 規則，以偵測 CodeCommit 儲存庫的變更，然後自動啟動管道。如需遷移現有管道的資訊，請參閱 來源動作和變更偵測方法。 教學：建立範本管道 (CodeCommit 儲存庫)已更新，顯示如何在您選取 CodeCommit 儲存庫和分支時建立 Amazon E CloudWatch vents 規則和角色。 在中建立管線 CodePipeline並且也在中編輯配管 CodePipeline已更新。 如需詳細資訊，請參閱 在中啟動管道 CodePipeline。	2017 年 10 月 11 日
新增與更新主題	CodePipeline 現在透過 Amazon CloudWatch 活動和亞馬遜簡單通知服務 (Amazon SNS) 為管道狀態變更通知提供內建支援。新增教學課程：設定 CloudWatch Events 規則以接收管道狀態變更的電子郵件通知這個新教學。如需詳細資訊，請參閱 監控 CodePipeline 事件。	2017 年 9 月 8 日

變更	描述	變更日期
新增與更新主題	您現在可以新增 CodePipeline 為 Amazon CloudWatch 活動動作的目標。Amazon E CloudWatch vents 規則可以設定為偵測來源變更，以便管道在這些變更發生後立即啟動，也可以將它們設定為執行排定的管道執行。已針對 PollForSourceChanges 來源動作組態選項新增資訊。如需詳細資訊，請參閱 在中啟動管道 CodePipeline。	2017 年 9 月 5 日
新 區域	CodePipeline 現已於亞太區域 (首爾) 及亞太區域 (孟買) 推出。更新 AWS CodePipeline 中的配額主題和區域與端點主題。	2017 年 7 月 27 日
新 區域	CodePipeline 目前已在美國西部 (加利佛尼亞北部)、加拿大 (中部) 和歐洲 (倫敦) 推出。更新 AWS CodePipeline 中的配額主題和區域與端點主題。	2017 年 6 月 29 日
更新主題	您現在可以檢視過去管道執行的詳細資訊，而不只是最新的執行。這些詳細資訊包含開始和結束時間、持續時間和執行 ID。最多可提供最新 12 個月期間之 100 個管道執行的詳細資訊。更新在中檢視管線和詳細資訊 CodePipeline、CodePipeline 權限參考和 AWS CodePipeline 中的配額主題，以反映此支援。	2017 年 6 月 22 日
更新主題	Nouvola 已新增至測試動作整合的可用動作清單。	2017 年 5 月 18 日
更新主題	在 AWS CodePipeline 精靈中，頁面步驟 4：Beta 版已重新命名「步驟 4：部署」。此步驟所建立的預設階段名稱已從 "Beta" 變更為 "Staging"。更新許多主題和螢幕擷取畫面，以反映這些變更。	2017 年 4 月 7 日

變更	描述	變更日期
更新主題	您現在可以將測試動 AWS CodeBuild 作新增至管線的任何階段。這使您可以更輕鬆地使 AWS CodeBuild 用對代碼運行單元測試。在此版本之前，您可以使用 AWS CodeBuild 僅作為構建操作的一部分運行單元測試。建置動作需要單位測試通常不會產生的建置輸出成品。 主題產品與服務整合 CodePipeline在中編輯配管 CodePipeline、和CodePipeline 配管結構參照已更新，以反映此支援 AWS CodeBuild。	2017 年 3 月 8 日
新增與更新主題	已重新整理目錄，以包含管道、動作和階段轉換的小節。已為 CodePipeline 自學課程加入新區段。為了獲得較佳的可用性，產品與服務整合 CodePipeline已分為較短的主題。 「授權與存取控制」新章節提供有關使用 AWS Identity and Access Management (IAM) 的全面資訊， CodePipeline 以及透過使用登入資料來協助安全存取資源。這些登入資料提供存取 AWS 資源所需的許可，例如從 Amazon S3 儲存貯體放置和擷取成品，以及將 AWS OpsWorks 堆疊整合到管道中。	2017 年 2 月 8 日
新 區域	CodePipeline 現已於亞太區域 (東京) 推出。更新 AWS CodePipeline 中的配額主題和區域與端點主題。	2016 年 12 月 14 日
新 區域	CodePipeline 現已在南美洲 (聖保羅) 推出。更新 AWS CodePipeline 中的配額主題和區域與端點主題。	2016 年 7 月 12 日

變更	描述	變更日期
更新主題	您現在可以將建置動 AWS CodeBuild 作新增至管線的任何階段。 AWS CodeBuild 是雲端中完全受控的建置服務，可編譯原始程式碼、執行單元測試，以及產生準備好部署的成品。您可以使用現有的組建專案，也可以在 CodePipeline 主控台中建立專案。接著，可以將建置專案的輸出部署為管道的一部分。 「驗證」和「存取控制」主題產品與服務整合 CodePipelineCodePipeline 配管結構參照已更新，以反映此支援 AWS CodeBuild。在中建立管線 CodePipeline 您現在可以使 CodePipeline 用 AWS CloudFormation 和 AWS 無伺服器應用程式模型來持續提供無伺服器應用程式。更新產品與服務整合 CodePipeline主題，以反映此支援。 產品與服務整合 CodePipeline已根據動作類型將產品重新組織為群組 AWS 和合作夥伴提供項目。	2016 年 12 月 1 日
新 區域	CodePipeline 現已在歐洲 (法蘭克福) 推出。更新 AWS CodePipeline 中的配額主題和區域與端點主題。	2016 年 11 月 16 日
更新主題	AWS CloudFormation 現在可以選取為管線中的部署提供者，讓您能夠在管線執行過程中對 AWS CloudFormation 堆疊和變更集採取動作。「驗證」和「存取控制」主題產品與服務整合 CodePipelineCodePipeline 配管結構參照已更新，以反映此支援 AWS CloudFormation。在中建立管線 CodePipeline	2016 年 11 月 3 日
新 區域	CodePipeline 亞太區域 (雪梨) 區域現已推出。更新 AWS CodePipeline 中的配額主題和區域與端點主題。	2016 年 10 月 26 日
新 區域	CodePipeline 現已於亞太區域 (新加坡) 推出。更新 AWS CodePipeline 中的配額主題和區域與端點主題。	2016 年 10 月 20 日
新 區域	CodePipeline 現已在美國東部 (俄亥俄) 區域推出。更新 AWS CodePipeline 中的配額主題和區域與端點主題。	2016 年 10 月 17 日

變更	描述	變更日期
更新主題	更新在中建立管線 CodePipeline，以反映在 Source provider (來源提供者) 和 Build provider (建置提供者) 清單中顯示自訂動作版本識別符的支援。	2016 年 9 月 22 日
更新主題	更新管理中的核准動作 CodePipeline小節來反映增強功能，讓核准動作檢閱者直接從電子郵件通知開啟 Approve or reject the revision (核准或拒絕修訂) 表單。	2016 年 9 月 14 日
新增與更新主題	這是一個新主題，說明如何檢視目前流經軟體發行管道之程式碼變更的詳細資料。檢閱手動核准動作或故障診斷管道中的失敗時，快速存取此資訊十分有用。 監控管道這個新小節提供集中位置，來放置所有與監控管道狀態和進度相關的主題。	2016 年 9 月 8 日
新增與更新主題	管理中的核准動作 CodePipeline這個新小節提供有關在管道中設定和使用手動核准動作的資訊。本節中的主題提供有關核准程序的概念性資訊、設定所需 IAM 許可、建立核准動作以及核准或拒絕核准動作的指示；以及在管道中達成核准動作時產生的 JSON 資料範例。	2016 年 7 月 6 日
新 區域	CodePipeline 歐洲 (愛爾蘭) 區域現已推出。更新 AWS CodePipeline 中的配額主題和區域與端點主題。	2016 年 6 月 23 日
新主題	新增重試階段中失敗的動作這個新主題，以說明如何重試階段中的失敗動作或一組平行失敗動作。	2016 年 6 月 22 日
更新主題	許多主題 (包括在中建立管線 CodePipeline驗證和存取控制和) 已更新CodePipeline 配管結構參照，以反映對設定管道以與中建立的自訂 Chef 說明書和應用程式一起部署程式碼的 AWS OpsWorks支援。產品與服務整合 CodePipeline CodePipeline 目前 AWS OpsWorks 僅在美國東部 (維吉尼亞北部) 區域 (us-east-1) 提供支援。	2016 年 6 月 2 日

變更	描述	變更日期
新增與更新主題	新增教學：建立範本管道 (CodeCommit 儲存庫) 這個新主題。本主題提供範例逐步解說，說明如何使用 CodeCommit 存放庫和分支做為管線中來源動作的來源位置。其他幾個主題已更新，以反映與此整合 CodeCommit，包括驗證與存取控制產品與服務整合 CodePipeline教學：建立四階段管道、、和疑難排 CodePipeline。	2016 年 4 月 18 日
新主題	新增在 CodePipeline 中於管道中呼叫 AWS Lambda 函數這個新主題。本主題包含將 Lambda AWS Lambda 函數新增至管道的範例函數和步驟。	2016 年 1 月 27 日
更新主題	「身分驗證與存取控制」中新增「資源型政策」章節。	2016 年 1 月 22 日
新主題	新增產品與服務整合 CodePipeline 這個新主題。與合作夥伴及其他合作夥伴整合的相關資訊 AWS 服務 已移至此主題。也已新增部落格和影片的連結。	2015 年 12 月 17 日
更新主題	與 Solano CI 整合的詳細資訊已新增至產品與服務整合 CodePipeline。	2015 年 11 月 17 日
更新主題	該 CodePipeline 插件詹金斯現在可以通過詹金斯插件管理器作為插件詹金斯庫的一部分。更新教學：建立四階段管道中有關安裝外掛程式的步驟。	2015 年 11 月 9 日
新 區域	CodePipeline 現已在美國西部 (奧勒岡) 區域推出。已更新 AWS CodePipeline 中的配額主題。在區域與端點中新增連結。	2015 年 10 月 22 日
新主題	新增為 Amazon S3 中存放的成品設定伺服器端加密 CodePipeline和在 CodePipeline 中建立使用來自另一個 AWS 帳戶資源的管道這兩個新主題。「身分驗證與存取控制」中新增範例 8：在管道中使用與另一個帳戶建立關聯的 AWS 資源章節。	2015 年 8 月 25 日

變更	描述	變更日期
更新主題	更新在 CodePipeline 中建立和新增自訂動作主題，以反映結構變更 (包含 `inputArtifactDetails` 和 `outputArtifactDetails`)。	2015 年 8 月 17 日
更新主題	本主疑難排 CodePipeline題已更新為疑難排解服務角色和 Elastic Beanstalk 問題的修訂步驟。	2015 年 8 月 11 日
更新主題	已更新「身分驗證與存取控制」主題，CodePipeline 的服務角色有最新變更。	2015 年 8 月 6 日
新主題	已新增疑難排 CodePipeline 主題。已針對 IAM 角色和詹金斯中教學：建立四階段管道新增更新的步驟。	2015 年 7 月 24 日
主題更新	教學：建立簡易管道 (S3 儲存貯體) 和教學：建立四階段管道中已新增下載範例檔案的更新步驟。	2015 年 7 月 22 日
主題更新	教學：建立簡易管道 (S3 儲存貯體) 中已新增下載範例檔案問題的暫時解決方法。	2015 年 7 月 17 日
主題更新	在 AWS CodePipeline 中的配額中新增連結，以指向可變更之限制的資訊。	2015 年 7 月 15 日
主題更新	已更新「身分驗證與存取控制」中的「受管政策」章節。	2015 年 7 月 10 日
初始公有版本	這是《 CodePipeline 使用者指南》的首次公開發行。	2015 年 7 月 9 日

AWS 詞彙表

如需最新的 AWS 術語，請參閱《AWS 詞彙表 參考》中的 AWS 詞彙表。

本文為英文版的機器翻譯版本，如內容有任何歧義或不一致之處，概以英文版為準。