保護應用程式

Astra Control Center

NetApp
November 21, 2023

This PDF was generated from https://docs.netapp.com/zh-tw/astra-control-center-2204/use/protection-overview.html on November 21, 2023. Always check docs.netapp.com for the latest.

目錄

保護應用程式 1
保護總覽 1
利用快照與備份來保護應用程式 1
還原應用程式 4
複製及移轉應用程式 6
管理應用程式執行掛勾 7

保護應用程式

保護總覽

您可以使用Astra Control Center為應用程式建立備份、複製、快照及保護原則。備份應用程式有助於您的服務和相關資料盡可能可用；在災難案例中、從備份還原可確保應用程式及其相關資料的完整還原、並將中斷時間降至最低。備份、複製和快照有助於防範勒索軟體、意外資料遺失和環境災難等常見威脅。 "瞭解Astra Control Center中可用的資料保護類型、以及使用時間"。

應用程式保護工作流程

您可以使用下列範例工作流程、開始保護應用程式。

[一] 備份所有應用程式

為了確保應用程式立即受到保護、 "建立所有應用程式的手動備份"。

[二] 為每個應用程式設定保護原則

若要自動化未來的備份與快照、 "為每個應用程式設定保護原則"。舉例來說、您可以從每週備份和每日快照開始著手、兩個快照均保留一個月。強烈建議使用保護原則來自動化備份與快照、而不要手動備份與快照。

[三] 選用：調整保護原則

隨著應用程式及其使用模式的改變、請視需要調整保護原則、以提供最佳保護。

[四] 發生災難時、請還原您的應用程式

如果發生資料遺失、您可以透過進行恢復 "還原最新的備份" 每個應用程式的第一名。然後您可以還原最新的快照（如果有）。

利用快照與備份來保護應用程式

使用自動保護原則或臨機操作、拍攝快照和備份資料、保護您的應用程式。您可以使用Astra UI或 "Astra Control API" 保護應用程式。

> 如果您使用Helm來部署應用程式、Astra Control Center需要Helm版本3。完全支援使用Helm 3部署的應用程式管理與複製（或從Helm 2升級至Helm 3）。不支援以Helm 2部署的應用程式。

> 當您在OpenShift叢集上建立裝載應用程式的專案時、專案（或Kubernetes命名空間）會被指派安全性轉換唯一碼。若要啟用Astra Control Center來保護應用程式、並將應用程式移至OpenShift中的其他叢集或專案、您必須新增原則、讓應用程式以任何唯一識別碼的形式執行。例如、下列OpenShift CLI命令會將適當的原則授予WordPress應用程式。
>
> 「occ new－project worm新聞」（occ new－project wormPress）「ocadm policy add－scc to－group anyuid system:serviceaccounts：wormPress」（ocadm policy add－scc對使用者權限-z預設值-n wormPress）

設定保護原則

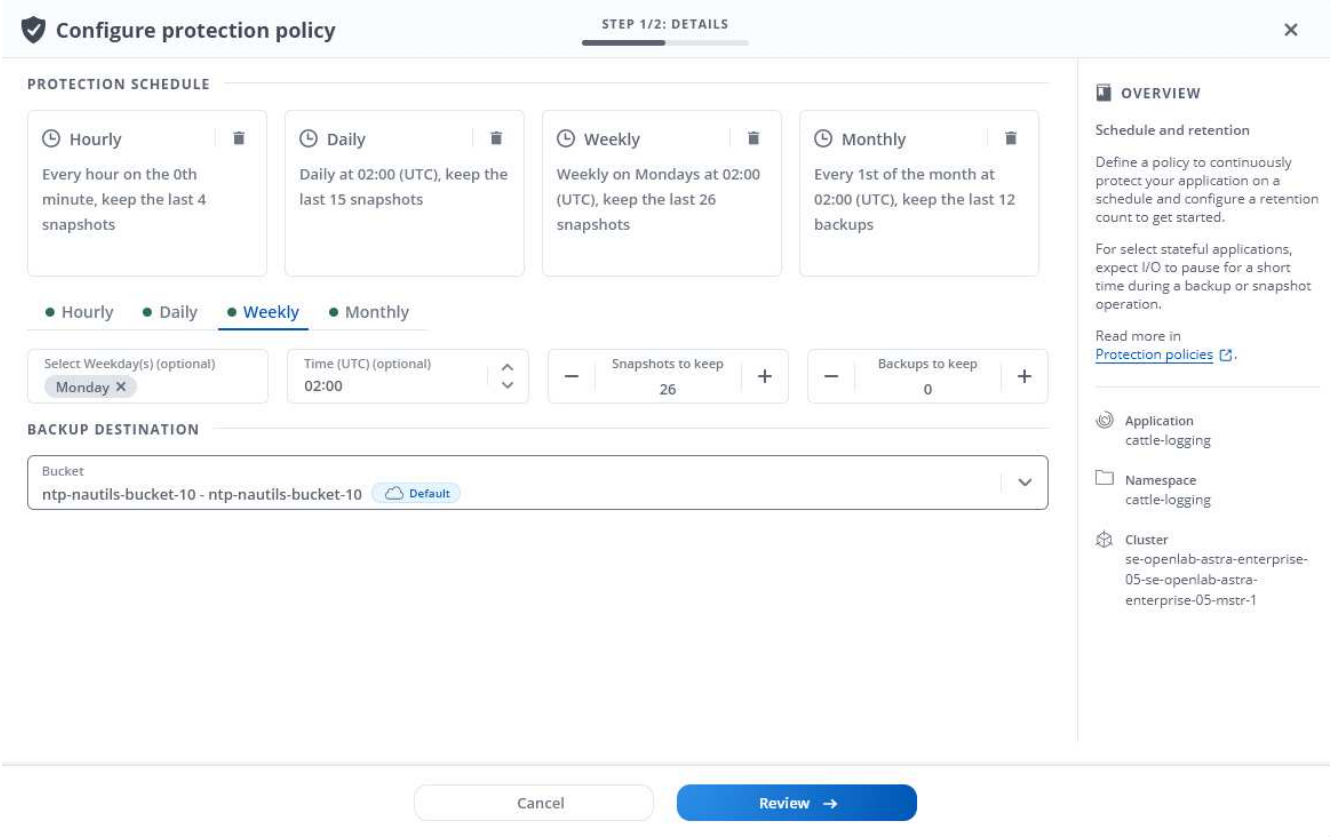
保護原則可在已定義的排程中建立快照、備份或兩者、以保護應用程式。您可以選擇每小時、每天、每週和每月建立快照和備份、也可以指定要保留的複本數量。例如、保護原則可能會建立每週備份和每日快照、並將備份和快照保留一個月。建立快照和備份的頻率、以及保留快照的時間長短、取決於組織的需求。

步驟

1. 選取*應用程式*、然後選取應用程式名稱。
2. 選擇*資料保護*。
3. 選取*設定保護原則*。
4. 選擇每小時、每天、每週和每月保留的快照和備份數量、以定義保護排程。

   您可以同時定義每小時、每日、每週及每月排程。在您設定保留層級之前、排程不會變成作用中。

   下列範例設定四種保護排程：每小時、每日、每週及每月提供快照與備份。



5. 選擇* Review *。
6. 選取*設定保護原則。*

結果

Astra Control Center使用您定義的排程和保留原則、建立並保留快照和備份、以實作資料保護原則。

建立快照

您可以隨時建立隨需快照。

步驟

1. 選擇*應用程式*。
2. 在所需應用程式*「Actions」（動作）欄的「Options」（選項）功能表中、選取*「Snapshot」（快照）*。
3. 自訂快照名稱、然後選取* Review *。
4. 檢閱快照摘要、然後選取* Snapshot *。

結果

快照程序隨即開始。當「資料保護>*快照*」頁面*「動作*」欄中的狀態為*可用*時、快照就會成功。

建立備份

您也可以隨時備份應用程式。

> Astra Control Center中的S3鏟斗未報告可用容量。在備份或複製由Astra Control Center管理的應用程式之前、請先查看ONTAP 資訊庫（英文）或StorageGRID 資訊庫（英文）管理系統中的庫位資訊。

步驟

1. 選擇*應用程式*。
2. 在所需應用程式*「Actions」（動作）欄的「Options」（選項）功能表中、選取「* Backup *」。
3. 自訂備份名稱。
4. 選擇是否要從現有的快照備份應用程式。如果選取此選項、您可以從現有快照清單中進行選擇。
5. 從儲存貯體區清單中選取、以選擇備份目的地。
6. 選擇* Review *。
7. 檢閱備份摘要、然後選取*備份*。

結果

Astra Control Center會建立應用程式的備份。

> 如果您的網路中斷或異常緩慢、備份作業可能會逾時。這會導致備份失敗。

> 無法停止執行中的備份。如果您需要刪除備份、請等到備份完成後再使用中的指示 [刪除備份]。若要刪除失敗的備份、 "使用Astra Control API"。

> 資料保護作業（複製、備份、還原）及後續持續調整磁碟區大小之後、UI中會顯示新的磁碟區大小、延遲最多20分鐘。資料保護作業只需幾分鐘就能成功完成、您可以使用儲存後端的管理軟體來確認磁碟區大小的變更。

檢視快照與備份

您可以從「資料保護」索引標籤檢視應用程式的快照與備份。

步驟

1. 選取*應用程式*、然後選取應用程式名稱。
2. 選擇*資料保護*。

   快照預設會顯示。

3. 選取*備份*以查看備份清單。

刪除快照

刪除不再需要的排程或隨需快照。

步驟

1. 選取*應用程式*、然後選取應用程式名稱。
2. 選擇*資料保護*。
3. 在所需快照*「Actions」（動作）欄的「Options」（選項）功能表中、選取*「Delete snapshot」（刪除快照）*。
4. 輸入「DELETE」一詞以確認刪除、然後選取*「Yes、Delete snapshot （是、刪除快照）」。

結果

Astra Control Center會刪除快照。

刪除備份

刪除不再需要的排程或隨需備份。

> 無法停止執行中的備份。如果您需要刪除備份、請等到備份完成後再使用這些指示。若要刪除失敗的備份、 "使用Astra Control API"。

1. 選取*應用程式*、然後選取應用程式名稱。
2. 選擇*資料保護*。
3. 選擇*備份*。
4. 在所需備份*「Actions」（動作）欄的「**Options**」（選項）功能表中、選取「Delete backup*」（刪除備份*）。
5. 輸入「DELETE」一詞以確認刪除、然後選取*「Yes、Delete backup*（是、刪除備份*）」。

結果

Astra Control Center會刪除備份。

還原應用程式

Astra Control可以從快照或備份還原應用程式。將應用程式還原至同一個叢集時、從現有的快照還原速度會更快。您可以使用Astra Control UI或 "Astra Control API" 以還原應用程式。

關於這項工作

- 強烈建議您在還原應用程式之前、先擷取應用程式的快照或備份應用程式。這可讓您在還原失敗時、從快照或備份進行複製。
- 如果您使用Helm來部署應用程式、Astra Control Center需要Helm版本3。完全支援使用Helm 3部署的應用程式管理與複製（或從Helm 2升級至Helm 3）。不支援以Helm 2部署的應用程式。
- 如果還原至不同的叢集、請確定叢集使用相同的持續磁碟區存取模式（例如ReadWriteMany）。如果目的地持續磁碟區存取模式不同、還原作業將會失敗。
- 任何具有命名空間名稱/ ID或命名空間標籤限制的成員使用者、都可以將應用程式複製或還原到同一個叢集上的新命名空間、或是組織帳戶中的任何其他叢集。不過、相同的使用者無法存取新命名空間中的複製或還原應用程式。在複製或還原作業建立新命名空間之後、帳戶管理員/擁有者可以編輯成員使用者帳戶、並更新受影響使用者的角色限制、以便授予新命名空間的存取權。
- 當您在OpenShift叢集上建立裝載應用程式的專案時、專案（或Kubernetes命名空間）會被指派安全性轉換唯一碼。若要啟用Astra Control Center來保護應用程式、並將應用程式移至OpenShift中的其他叢集或專案、您必須新增原則、讓應用程式以任何唯一識別碼的形式執行。例如、下列OpenShift CLI命令會將適當的原則授予WordPress應用程式。

  「occ new－project worm新聞」（occ new－project wormPress）「ocadm policy add－scc to－group anyuid system:serviceaccounts：wormPress」（ocadm policy add－scc對使用者權限-z預設值-n wormPress）

步驟

1. 選取*應用程式*、然後選取應用程式名稱。
2. 選擇*資料保護*。
3. 如果要從快照還原、請選取* Snapshot*圖示。否則、請選取*備份*圖示以從備份還原。
4. 從您要還原之快照或備份的「動作」欄中的「選項」功能表中、選取「還原應用程式」。
5. 還原詳細資料：指定還原應用程式的詳細資料。預設會顯示目前的叢集和命名空間。保留這些值不變、即可將應用程式還原至舊版。如果您要還原至不同的叢集或命名空間、請變更這些值。
   - 輸入應用程式的名稱和命名空間。
   - 選擇應用程式的目的地叢集。
   - 選擇* Review *。

   > 如果還原至先前刪除的命名空間、則會在還原程序中建立名稱相同的新命名空間。任何在先前刪除命名空間中擁有管理應用程式權限的使用者、都必須手動還原新重新建立命名空間的權限。

6. 還原摘要：檢閱還原動作的詳細資料、輸入「還原」、然後選取*還原*。

結果

Astra Control Center會根據您提供的資訊來還原應用程式。如果您就地還原應用程式、則任何現有持續磁碟區的內容都會由還原應用程式的持續磁碟區內容取代。

> 在執行資料保護作業（複製、備份、還原）及後續持續調整磁碟區大小之後、新的磁碟區大小會在網路UI中顯示、延遲最多20分鐘。資料保護作業只需幾分鐘就能成功完成、您可以使用儲存後端的管理軟體來確認磁碟區大小的變更。

複製及移轉應用程式

複製現有的應用程式、在相同的Kubernetes叢集或其他叢集上建立複製的應用程式。當Astra Control Center複製應用程式時、會建立應用程式組態和持續儲存的複本。

如果您需要將應用程式和儲存設備從一個Kubernetes叢集移至另一個叢集、複製作業將有助於您。例如、您可能想要透過CI/CD傳輸途徑和Kubernetes命名空間來移動工作負載。您可以使用Astra UI或 "Astra Control API" 複製及移轉應用程式。

您需要的產品

若要將應用程式複製到不同的叢集、您需要預設的儲存區。當您新增第一個儲存區時、它會成為預設儲存區。

關於這項工作

- 如果您部署的應用程式已明確設定StorageClass、且需要複製應用程式、則目標叢集必須具有原本指定的StorageClass。將具有明確設定StorageClass的應用程式複製到沒有相同StorageClass的叢集、將會失敗。
- 如果您複製由操作人員部署的Jenkins CI執行個體、則需要手動還原持續性資料。這是應用程式部署模式的限制。
- Astra Control Center中的S3鏟斗未報告可用容量。在備份或複製由Astra Control Center管理的應用程式之前、請先查看ONTAP 資訊庫（英文）或StorageGRID 資訊庫（英文）管理系統中的庫位資訊。
- 在應用程式備份或應用程式還原期間、您可以選擇性地指定庫位ID。不過、應用程式複製作業一律會使用已定義的預設儲存區。沒有選項可變更實體複本的儲存區。如果您想要控制所使用的儲存桶、您也可以選擇 "變更庫位預設值" 或執行 "備份" 接著是A "還原" 獨立提供。
- 任何具有命名空間名稱/ ID或命名空間標籤限制的成員使用者、都可以將應用程式複製或還原到同一個叢集上的新命名空間、或是組織帳戶中的任何其他叢集。不過、相同的使用者無法存取新命名空間中的複製或還原應用程式。在複製或還原作業建立新命名空間之後、帳戶管理員/擁有者可以編輯成員使用者帳戶、並更新受影響使用者的角色限制、以便授予新命名空間的存取權。

OpenShift考量

- 如果您在叢集之間複製應用程式、來源叢集和目的地叢集必須是OpenShift的相同發佈版本。例如、如果您從OpenShift 4.7叢集複製應用程式、請使用同樣為OpenShift 4.7的目的地叢集。
- 當您在OpenShift叢集上建立裝載應用程式的專案時、專案（或Kubernetes命名空間）會被指派安全性轉換唯一碼。若要啟用Astra Control Center來保護應用程式、並將應用程式移至OpenShift中的其他叢集或專案、您必須新增原則、讓應用程式以任何唯一識別碼的形式執行。例如、下列OpenShift CLI命令會將適當的原則授予WordPress應用程式。

  「occ new－project worm新聞」（occ new－project wormPress）「ocadm policy add－scc to－group anyuid system:serviceaccounts：wormPress」（ocadm policy add－scc對使用者權限-z預設值-n wormPress）

步驟

1. 選擇*應用程式*。
2. 執行下列其中一項：
   - 在所需應用程式的*「Actions」（動作）欄中、選取「Options」（選項）功能表。
   - 選取所需應用程式的名稱、然後選取頁面右上角的狀態下拉式清單。
3. 選擇* Clone（克隆）*。

4. 複製詳細資料：指定複製的詳細資料：
    - 輸入名稱。
    - 輸入複本的命名空間。
    - 選擇要複製的目的地叢集。
    - 選擇是要從現有的快照或備份建立複本。如果您未選取此選項、Astra Control Center會從應用程式的目前狀態建立複本。
5. 資料來源：如果您選擇從現有的快照或備份中複製、請選擇您要使用的快照或備份。
6. 選擇* Review *。
7. * Clone Summary（複製摘要）：檢閱有關複製的詳細資料、然後選取 Clone（複製）*。

結果

Astra Control Center會根據您提供的資訊來複製該應用程式。當新的應用程式實體複本在「應用程式」頁面上處於「可用」狀態時、即表示該實體複本作業成功。

> 資料保護作業（複製、備份、還原）及後續持續調整磁碟區大小之後、UI中會顯示新的磁碟區大小、延遲最多20分鐘。資料保護作業只需幾分鐘就能成功完成、您可以使用儲存後端的管理軟體來確認磁碟區大小的變更。

管理應用程式執行掛勾

執行攔截是自訂指令碼、您可以在託管應用程式的快照之前或之後執行。例如、如果您有資料庫應用程式、您可以使用執行掛勾來暫停快照之前的所有資料庫交易、並在快照完成後繼續交易。如此可確保應用程式一致的快照。

預設執行掛勾和規則運算式

對於某些應用程式、Astra Control隨附NetApp提供的預設執行掛勾、可處理快照前後的凍結和解凍作業。Astra Control使用規則運算式、將應用程式的容器映像與下列應用程式配對：

- MariaDB
    - 符合規則運算式：\bmariadb\b
- MySQL
    - 相符的規則運算式：\bmysql\b
- PostgreSQL
    - 相符的規則運算式：\bpostgresql\b

如果有相符項目、則NetApp提供的該應用程式預設執行掛勾會顯示在應用程式的作用中執行掛勾清單中、而這些掛勾會在擷取該應用程式的快照時自動執行。如果其中一個自訂應用程式有類似的映像名稱、正好符合其中一個規則運算式（而且您不想使用預設的執行掛勾）、您可以變更映像名稱、 或停用該應用程式的預設執行掛勾、改用自訂掛勾。

您無法刪除或修改預設的執行掛勾。

關於自訂執行掛勾的重要注意事項

規劃應用程式的執行掛勾時、請考量下列事項。

- Astra Control需要以執行Shell指令碼的格式寫入執行掛勾。
- 指令碼大小限制為128 KB。
- Astra Control會使用執行掛勾設定和任何符合條件來判斷哪些掛勾適用於快照。
- 所有執行掛機故障都是軟性故障、即使掛機故障、仍會嘗試其他掛機和快照。但是、當掛機失敗時、會在*活動*頁面事件記錄中記錄警告事件。
- 若要建立、編輯或刪除執行掛勾、您必須是擁有擁有者、管理員或成員權限的使用者。
- 如果執行掛機執行時間超過25分鐘、掛機將會失敗、並建立傳回代碼為「N/A」的事件記錄項目。任何受影響的快照都會逾時並標示為故障、並會出現一個事件記錄項目、指出逾時時間。

> 由於執行掛勾通常會減少或完全停用執行中應用程式的功能、因此您應該一律盡量縮短自訂執行掛勾執行所需的時間。

執行快照時、執行掛機事件會依照下列順序進行：

1. 任何適用的NetApp提供的預設快照前執行掛勾、都會在適當的容器上執行。
2. 任何適用的自訂快照前執行掛勾都會在適當的容器上執行。您可以視需要建立及執行任意數量的自訂快照前置掛勾、但在快照之前執行這些掛勾的順序並不保證也無法設定。
3. 快照即會執行。
4. 任何適用的自訂快照後執行掛勾都會在適當的容器上執行。您可以視需要建立及執行任意數量的自訂快照後掛勾、但快照後執行這些掛勾的順序並不保證也無法設定。
5. 任何適用的NetApp提供的預設快照後執行掛勾、都會在適當的容器上執行。

> 在正式作業環境中啟用執行攔截指令碼之前、請務必先進行測試。您可以使用'kubecl exec'命令來方便地測試指令碼。在正式作業環境中啟用執行掛勾之後、請測試所產生的快照、以確保它們一致。您可以將應用程式複製到暫用命名空間、還原快照、然後測試應用程式、藉此完成此作業。

檢視現有的執行掛勾

您可以檢視現有的自訂或NetApp提供的應用程式預設執行勾點。

步驟

1. 移至*應用程式*、然後選取託管應用程式的名稱。
2. 選取*執行掛勾*索引標籤。

   您可以在結果清單中檢視所有已啟用或已停用的執行掛勾。您可以查看Hook的狀態、來源、以及何時執行（快照前或快照後）。若要檢視執行掛起的相關事件記錄、請前往左側導覽區域的*活動*頁面。

建立自訂執行掛勾

您可以為應用程式建立自訂執行掛勾。請參閱 "執行攔截範例" 如需攔截範例、您需要擁有擁有擁有者、管理員

或成員權限、才能建立執行掛勾。

> 當您建立自訂Shell指令碼作為執行掛勾時、請記得在檔案開頭指定適當的Shell、除非您執行Linux命令或提供執行檔的完整路徑。

步驟

1. 選取*應用程式*、然後選取託管應用程式的名稱。
2. 選取*執行掛勾*索引標籤。
3. 選取*新增連結*。
4. 在* HookDetails（掛機詳細資料）區域中、視掛機的執行時間而定、選擇*預先快照*或*快照後。
5. 輸入掛機的唯一名稱。
6. （選用）輸入執行期間要傳遞至掛機的任何引數、並在您輸入的每個引數之後按Enter鍵以記錄每個引數。
7. 在「* Container images"（* Container映像*）區域中、如果掛勾應針對應用程式中包含的所有容器映像執行、請啟用「* Apply to all Container images"（套用至所有容器映像）核取方塊。如果掛機只能對一個或多個指定的容器映像起作用、請在「要比對的容器映像名稱」欄位中輸入容器映像名稱。
8. 在*指令碼*區域中、執行下列其中一項：
   - 上傳自訂指令碼。
     i. 選取*上傳檔案*選項。
     ii. 瀏覽至檔案並上傳。
     iii. 為指令碼指定唯一名稱。
     iv. （選用）輸入其他系統管理員應該知道的任何指令碼附註。
   - 從剪貼簿貼入自訂指令碼。
     i. 選擇*從剪貼簿貼上*選項。
     ii. 選取文字欄位、然後將指令碼文字貼到欄位中。
     iii. 為指令碼指定唯一名稱。
     iv. （選用）輸入其他系統管理員應該知道的任何指令碼附註。
9. 選取*新增攔截*。

停用執行掛勾

如果您想要暫時避免在應用程式快照之前或之後執行、可以停用執行掛勾。您需要擁有擁有擁有者、管理員或成員權限、才能停用執行掛勾。

步驟

1. 選取*應用程式*、然後選取託管應用程式的名稱。
2. 選取*執行掛勾*索引標籤。
3. 在「動作」欄中選取「選項」功能表、以顯示您要停用的掛勾。
4. 選擇*停用*。

刪除執行掛勾

如果不再需要執行掛勾、您可以完全移除該掛勾。您需要擁有擁有擁有者、管理員或成員權限、才能刪除執行掛勾。

步驟

1. 選取*應用程式*、然後選取託管應用程式的名稱。
2. 選取*執行掛勾*索引標籤。
3. 在「動作」欄中選取「選項」功能表、以選取您要刪除的掛勾。
4. 選擇*刪除*。

執行攔截範例

請使用下列範例、瞭解如何建立執行掛勾的架構。您可以使用這些hooks做為範本、或做為測試指令碼。

簡單的成功範例

這是一個簡單的勾點、可成功將訊息寫入標準輸出和標準錯誤。

```
#!/bin/sh

# success_sample.sh
#
# A simple noop success hook script for testing purposes.
#
# args: None
#



#
# Writes the given message to standard output
#
# $* - The message to write
#
msg() {
    echo "$*"
}



#
# Writes the given information message to standard output
#
# $* - The message to write
#
info() {
```

```
    msg "INFO: $*"
}

#
# Writes the given error message to standard error
#
# $* - The message to write
#
error() {
    msg "ERROR: $*" 1>&2
}


#
# main
#

# log something to stdout
info "running success_sample.sh"

# exit with 0 to indicate success
info "exit 0"
exit 0
```

簡單的成功範例（**Bash**版本）

這是一個簡單的攔截、成功將訊息寫入標準輸出和標準錯誤（寫入Bash）。

```
#!/bin/bash

# success_sample.bash
#
# A simple noop success hook script for testing purposes.
#
# args: None


#
# Writes the given message to standard output
#
# $* - The message to write
#
msg() {
    echo "$*"
}
```

```
#
# Writes the given information message to standard output
#
# $* - The message to write
#
info() {
    msg "INFO: $*"
}

#
# Writes the given error message to standard error
#
# $* - The message to write
#
error() {
    msg "ERROR: $*" 1>&2
}


#
# main
#

# log something to stdout
info "running success_sample.bash"

# exit with 0 to indicate success
info "exit 0"
exit 0
```

簡單的成功範例（**zsh**版本）

這是一個簡單的勾點範例、可成功將訊息寫入標準輸出和標準錯誤、並寫入Z Shell。

```
#!/bin/zsh

# success_sample.zsh
#
# A simple noop success hook script for testing purposes.
#
# args: None
#


#
```

```
# Writes the given message to standard output
#
# $* - The message to write
#
msg() {
    echo "$*"
}


#
# Writes the given information message to standard output
#
# $* - The message to write
#
info() {
    msg "INFO: $*"
}

#
# Writes the given error message to standard error
#
# $* - The message to write
#
error() {
    msg "ERROR: $*" 1>&2
}

#
# main
#

# log something to stdout
info "running success_sample.zsh"

# exit with 0 to indicate success
info "exit 0"
exit 0
```

成功的引數範例

下列範例示範如何在攔截中使用args。

```
#!/bin/sh

# success_sample_args.sh
#
```

```
# A simple success hook script with args for testing purposes.
#
# args: Up to two optional args that are echoed to stdout

#
# Writes the given message to standard output
#
# $* - The message to write
#
msg() {
    echo "$*"
}


#
# Writes the given information message to standard output
#
# $* - The message to write
#
info() {
    msg "INFO: $*"
}

#
# Writes the given error message to standard error
#
# $* - The message to write
#
error() {
    msg "ERROR: $*" 1>&2
}


#
# main
#

# log something to stdout
info "running success_sample_args.sh"


# collect args
arg1=$1
arg2=$2

# output args and arg count to stdout
info "number of args: $#"
```

```
info "arg1 ${arg1}"
info "arg2 ${arg2}"

# exit with 0 to indicate success
info "exit 0"
exit 0
```

快照前/快照後掛機範例

以下範例說明如何將相同的指令碼同時用於快照前和快照後掛勾。

```
#!/bin/sh

# success_sample_pre_post.sh
#
# A simple success hook script example with an arg for testing purposes
# to demonstrate how the same script can be used for both a prehook and
posthook
#
# args: [pre|post]

# unique error codes for every error case
ebase=100
eusage=$((ebase+1))
ebadstage=$((ebase+2))
epre=$((ebase+3))
epost=$((ebase+4))


#
# Writes the given message to standard output
#
# $* - The message to write
#
msg() {
    echo "$*"
}


#
# Writes the given information message to standard output
#
# $* - The message to write
#
info() {
    msg "INFO: $*"
```

```
}

#
# Writes the given error message to standard error
#
# $* - The message to write
#
error() {
    msg "ERROR: $*" 1>&2
}


#
# Would run prehook steps here
#
prehook() {
    info "Running noop prehook"
    return 0
}

#
# Would run posthook steps here
#
posthook() {
    info "Running noop posthook"
    return 0
}


#
# main
#

# check arg
stage=$1
if [ -z "${stage}" ]; then
    echo "Usage: $0 <pre|post>"
    exit ${eusage}
fi

if [ "${stage}" != "pre" ] && [ "${stage}" != "post" ]; then
    echo "Invalid arg: ${stage}"
    exit ${ebadstage}
fi

# log something to stdout
```

```
info "running success_sample_pre_post.sh"

if [ "${stage}" = "pre" ]; then
    prehook
    rc=$?
    if [ ${rc} -ne 0 ]; then
        error "Error during prehook"
    fi
fi

if [ "${stage}" = "post" ]; then
    posthook
    rc=$?
    if [ ${rc} -ne 0 ]; then
        error "Error during posthook"
    fi
fi

exit ${rc}
```

故障範例

下列範例示範如何處理攔截式故障。

```
#!/bin/sh

# failure_sample_arg_exit_code.sh
#
# A simple failure hook script for testing purposes.
#
# args: [the exit code to return]
#



#
# Writes the given message to standard output
#
# $* - The message to write
#
msg() {
    echo "$*"
}



#
# Writes the given information message to standard output
```

```
#
# $* - The message to write
#
info() {
    msg "INFO: $*"
}

#
# Writes the given error message to standard error
#
# $* - The message to write
#
error() {
    msg "ERROR: $*" 1>&2
}


#
# main
#

# log something to stdout
info "running failure_sample_arg_exit_code.sh"

argexitcode=$1

# log to stderr
error "script failed, returning exit code ${argexitcode}"

# exit with specified exit code
exit ${argexitcode}
```

詳細故障範例

下列範例示範如何以更詳細的記錄功能來處理掛機中的失敗。

```
#!/bin/sh

# failure_sample_verbose.sh
#
# A simple failure hook script with args for testing purposes.
#
# args: [The number of lines to output to stdout]


#
```

```
# Writes the given message to standard output
#
# $* - The message to write
#
msg() {
    echo "$*"
}


#
# Writes the given information message to standard output
#
# $* - The message to write
#
info() {
    msg "INFO: $*"
}

#
# Writes the given error message to standard error
#
# $* - The message to write
#
error() {
    msg "ERROR: $*" 1>&2
}


#
# main
#

# log something to stdout
info "running failure_sample_verbose.sh"


# output arg value to stdout
linecount=$1
info "line count ${linecount}"

# write out a line to stdout based on line count arg
i=1
while [ "$i" -le ${linecount} ]; do
    info "This is line ${i} from failure_sample_verbose.sh"
    i=$(( i + 1 ))
done
```

```
error "exiting with error code 8"
exit 8
```

退出程式碼範例失敗

下列範例示範攔截失敗、並顯示結束程式碼。

```
#!/bin/sh

# failure_sample_arg_exit_code.sh
#
# A simple failure hook script for testing purposes.
#
# args: [the exit code to return]
#



#
# Writes the given message to standard output
#
# $* - The message to write
#
msg() {
    echo "$*"
}



#
# Writes the given information message to standard output
#
# $* - The message to write
#
info() {
    msg "INFO: $*"
}

#
# Writes the given error message to standard error
#
# $* - The message to write
#
error() {
    msg "ERROR: $*" 1>&2
}
```

```
#
# main
#

# log something to stdout
info "running failure_sample_arg_exit_code.sh"

argexitcode=$1

# log to stderr
error "script failed, returning exit code ${argexitcode}"

# exit with specified exit code
exit ${argexitcode}
```

失敗後成功範例

下列範例示範第一次執行時發生掛機故障、但在第二次執行後成功。

```
#!/bin/sh

# failure_then_success_sample.sh
#
# A hook script that fails on initial run but succeeds on second run for
testing purposes.
#
# Helpful for testing retry logic for post hooks.
#
# args: None
#

#
# Writes the given message to standard output
#
# $* - The message to write
#
msg() {
    echo "$*"
}


#
# Writes the given information message to standard output
#
# $* - The message to write
#
```

```
info() {
    msg "INFO: $*"
}

#
# Writes the given error message to standard error
#
# $* - The message to write
#
error() {
    msg "ERROR: $*" 1>&2
}


#
# main
#

# log something to stdout
info "running failure_success sample.sh"


if [ -e /tmp/hook-test.junk ] ; then
    info "File does exist.  Removing /tmp/hook-test.junk"
    rm /tmp/hook-test.junk
    info "Second run so returning exit code 0"
    exit 0
else
    info "File does not exist.  Creating /tmp/hook-test.junk"
    echo "test" > /tmp/hook-test.junk
    error "Failed first run, returning exit code 5"
    exit 5
fi
```

版權資訊

Copyright © 2023 NetApp, Inc. 版權所有。台灣印製。非經版權所有人事先書面同意，不得將本受版權保護文件的任何部分以任何形式或任何方法（圖形、電子或機械）重製，包括影印、錄影、錄音或儲存至電子檢索系統中。

由 NetApp 版權資料衍伸之軟體必須遵守下列授權和免責聲明：

此軟體以 NETAPP「原樣」提供，不含任何明示或暗示的擔保，包括但不限於有關適售性或特定目的適用性之擔保，特此聲明。於任何情況下，就任何已造成或基於任何理論上責任之直接性、間接性、附隨性、特殊性、懲罰性或衍生性損害（包括但不限於替代商品或服務之採購；使用、資料或利潤上的損失；或企業營運中斷），無論是在使用此軟體時以任何方式所產生的契約、嚴格責任或侵權行為（包括疏忽或其他）等方面，NetApp 概不負責，即使已被告知有前述損害存在之可能性亦然。

NetApp 保留隨時變更本文所述之任何產品的權利，恕不另行通知。NetApp 不承擔因使用本文所述之產品而產生的責任或義務，除非明確經過 NetApp 書面同意。使用或購買此產品並不會在依據任何專利權、商標權或任何其他 NetApp 智慧財產權的情況下轉讓授權。

本手冊所述之產品受到一項（含）以上的美國專利、國外專利或申請中專利所保障。

有限權利說明：政府機關的使用、複製或公開揭露須受 DFARS 252.227-7013（2014 年 2 月）和 FAR 52.227-19（2007 年 12 月）中的「技術資料權利 - 非商業項目」條款 (b)(3) 小段所述之限制。

此處所含屬於商業產品和 / 或商業服務（如 FAR 2.101 所定義）的資料均為 NetApp, Inc. 所有。根據本協議提供的所有 NetApp 技術資料和電腦軟體皆屬於商業性質，並且完全由私人出資開發。 美國政府對於該資料具有非專屬、非轉讓、非轉授權、全球性、有限且不可撤銷的使用權限，僅限於美國政府為傳輸此資料所訂合約所允許之範圍，並基於履行該合約之目的方可使用。除非本文另有規定，否則未經 NetApp Inc. 事前書面許可，不得逕行使用、揭露、重製、修改、履行或展示該資料。美國政府授予國防部之許可權利，僅適用於 DFARS 條款 252.227-7015(b)（2014 年 2 月）所述權利。

商標資訊

NETAPP、NETAPP 標誌及 http://www.netapp.com/TM 所列之標章均為 NetApp, Inc. 的商標。文中所涉及的所有其他公司或產品名稱，均為其各自所有者的商標，不得侵犯。