保護應用程式

Astra Control Center

NetApp
November 21, 2023

目錄

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"。

此外、您也可以將應用程式複寫到遠端叢集、以便做好災難恢復的準備。

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] 保護所有應用程式

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] 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] 將應用程式複寫到遠端叢集

"複寫應用程式" 使用NetApp SnapMirror技術將其移至遠端叢集。Astra Control會將Snapshot複寫到遠端叢集、提供非同步的災難恢復功能。

[五] 發生災難時、請使用最新的備份或複寫功能、將應用程式還原至遠端系統

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）。或者、您也可以使用複寫功能來複寫到遠端系統。

利用快照與備份來保護應用程式

使用自動保護原則或以臨機操作的方式、擷取快照與備份資料、以保護所有應用程式。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。

```
oc new-project wordpress
oc adm policy add-scc-to-group anyuid system:serviceaccounts:wordpress
oc adm policy add-scc-to-user privileged -z default -n wordpress
```

您可以執行下列與保護應用程式資料相關的工作：

- [設定保護原則]
- [建立快照]
- [建立備份]
- [檢視快照與備份]
- [刪除快照]
- [取消備份]
- [刪除備份]

設定保護原則

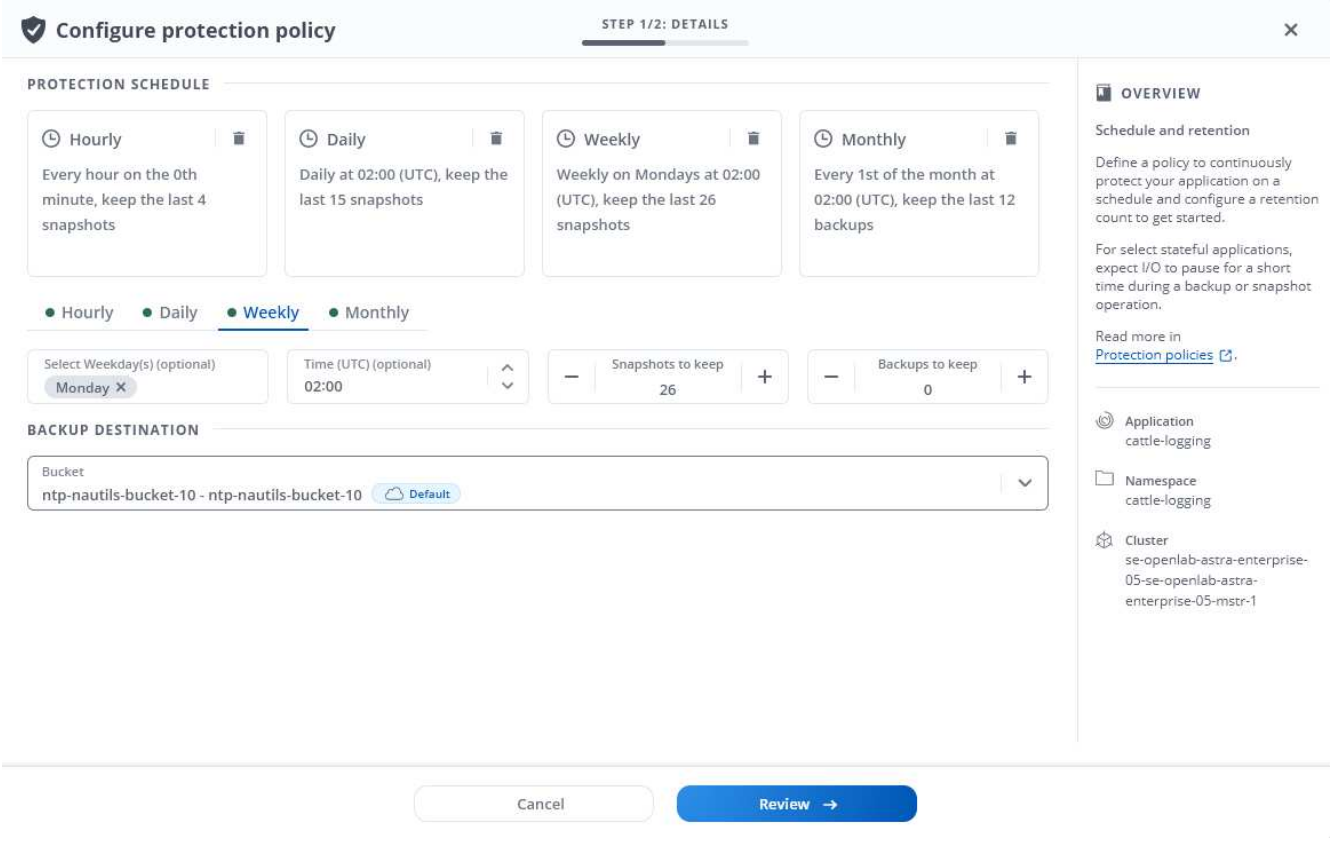
保護原則可在已定義的排程中建立快照、備份或兩者、以保護應用程式。您可以選擇每小時、每天、每週和每月建立快照和備份、也可以指定要保留的複本數量。例如、保護原則可能會建立每週備份和每日快照、並將備份和快照保留一個月。建立快照和備份的頻率、以及保留快照的時間長短、取決於組織的需求。

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。

> 您無法刪除目前正在複寫的Snapshot複本。

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。

取消備份

您可以取消進行中的備份。

> 若要取消備份、備份必須處於執行中狀態。您無法取消處於「擱置中」狀態的備份。

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*。
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「Cancel*」（取消*）。
5. 輸入「cancel」一詞以確認刪除、然後選擇「* Yes、cancel backup*（是、取消備份*）」。

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"。

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。

  ```
  oc new-project wordpress
  oc adm policy add-scc-to-group anyuid system:serviceaccounts:wordpress
  oc adm policy add-scc-to-user privileged -z default -n wordpress
  ```

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。

使用SnapMirror技術將應用程式複寫到遠端系統

使用Astra Control、您可以利用NetApp SnapMirror技術的非同步複寫功能、利用低RPO（恢復點目標）和低RTO（恢復時間目標）、為應用程式建立營運不中斷。設定完成後、您的應用程式就能將資料和應用程式變更從一個叢集複寫到另一個叢集。

如需備份/還原與複寫之間的比較、請參閱 "資料保護概念"。

您可以在不同的案例中複寫應用程式、例如下列僅限內部部署、混合式和多雲端的案例：

- 內部部署站台A到內部部署站台B
- 內部部署至雲端、Cloud Volumes ONTAP 使用不整合技術
- 將Cloud Volumes ONTAP 雲端技術整合至內部部署
- 雲端搭配從功能到雲端（在同一個雲端供應商的不同地區或不同的雲端供應商之間）Cloud Volumes ONTAP

Astra Control可在內部部署叢集、內部部署到雲端（使用Cloud Volumes ONTAP 原地功能）或在雲端之間（Cloud Volumes ONTAP 從地到Cloud Volumes ONTAP 地）複寫應用程式。

> 您可以同時以相反方向複寫不同的應用程式（在其他叢集或站台上執行）。例如、應用程式A、B、C可以從資料中心1複寫到資料中心2、而應用程式X、Y、Z可以從資料中心2複寫到資料中心1。

使用Astra Control、您可以執行下列與複寫應用程式相關的工作：

- [設定複寫關係]
- [在目的地叢集上使複寫的應用程式上線（容錯移轉）]
- [重新同步複寫失敗的情況]
- [反轉應用程式複寫]
- [將應用程式容錯移轉至原始來源叢集]
- [刪除應用程式複寫關係]

複寫先決條件

請參閱 "複寫先決條件" 開始之前。

設定複寫關係

設定複寫關係時、會涉及複寫原則的下列項目；

- 選擇您希望Astra Control執行應用程式Snapshot的頻率（包括應用程式的Kubernetes資源、以及每個應用程式磁碟區的Volume Snapshot）
- 選擇複寫排程（包括Kubernetes資源及持續磁碟區資料）
- 設定拍攝Snapshot的時間

步驟

1. 從Astra Control左側導覽中、選取* Applications*。
2. 在「應用程式」頁面中、選取「資料保護>*複寫*」索引標籤。
3. 在「Data Protection（資料保護）」>「Replication（複寫）」索引標籤中、選取「* Configure replReplication polici*或者、從「應用程式保護」方塊中選取「動作」選項、然後選取「設定複寫原則*」。
4. 輸入或選取下列資訊：
   - 目的地叢集
   - 目的地儲存類別：選取或輸入使用目的地ONTAP 叢集上配對SVM的儲存類別。
   - 複寫類型：「非同步」目前是唯一可用的複寫類型。
   - 目的地命名空間：為目的地叢集輸入新的或現有的目的地命名空間。

     所選命名空間中的任何衝突資源都會被覆寫。

   - 複寫頻率：設定您希望Astra Control多久拍攝一次Snapshot並將其複寫到目的地。
   - 偏移：設定您想要Astra Control拍攝Snapshot的小時數（分鐘）。您可能想要使用偏移、使其不與其他排程作業一致。例如、如果您想要從10：02開始每5分鐘拍攝一次Snapshot、請輸入「02」作為偏移分鐘數。結果為10：02、10：07、10：12等
5. 選取*下一步*、檢閱摘要、然後選取*儲存*。

   一開始、狀態會在第一個排程發生之前顯示「app-mirror」（應用程式鏡射）。

   Astra Control會建立用於複寫的應用程式Snapshot。

6. 若要查看應用程式Snapshot狀態、請選取* Applications*>* Snapshot*索引標籤。

   Snapshot名稱使用「repl複 寫排程-」格式。Astra Control保留上次用於複寫的Snapshot。複寫成功完成後、會刪除任何舊版複寫Snapshot。

結果

這會建立複寫關係。

Astra Control在建立關係後完成下列行動：

- 在目的地上建立命名空間（如果不存在）
- 在目的地命名空間上建立一個與來源應用程式PVCS對應的PVc。
- 取得應用程式一致的初始Snapshot。
- 使用初始Snapshot建立持續磁碟區的SnapMirror關係。

「Data Protection（資料保護）」頁面會顯示複寫關係狀態和狀態：<健全狀況狀態>|<關係生命週期狀態>

例如：正常|已建立

深入瞭解複寫狀態和狀態、如下所示。

在目的地叢集上使複寫的應用程式上線（容錯移轉）

使用Astra Control、您可以將複寫的應用程式「容錯移轉」到目的地叢集。此程序會停止複寫關係、並在目的地叢集上使應用程式上線。此程序不會停止來源叢集上的應用程式（如果運作正常）。

步驟

1. 從Astra Control左側導覽中、選取* Applications*。
2. 在「應用程式」頁面中、選取「資料保護>*複寫*」索引標籤。
3. 在Data Protection（資料保護）> Replication（複寫）索引標籤的Actions（動作）功能表中、選取* Fail over（容錯移轉）*。
4. 在「容錯移轉」頁面中、檢閱資訊並選取*容錯移轉*。

結果

容錯移轉程序會導致下列動作：

- 在目的地叢集上、應用程式是根據最新複寫的Snapshot來啟動。
- 來源叢集和應用程式（如果運作正常）不會停止、將會繼續執行。
- 複寫狀態會變更為「容錯移轉」、並在完成後變更為「容錯移轉」。
- 來源應用程式的保護原則會根據容錯移轉時來源應用程式上的排程、複製到目的地應用程式。
- Astra Control會在來源叢集和目的地叢集上顯示應用程式及其各自的健全狀況。

重新同步複寫失敗的情況

重新同步作業會重新建立複寫關係。您可以選擇關聯的來源、以保留來源或目的地叢集上的資料。此作業會重新建立SnapMirror關係、以便在選擇的方向開始磁碟區複寫。

此程序會在重新建立複寫之前、停止新目的地叢集上的應用程式。

> 在重新同步程序期間、生命週期狀態會顯示為「Establishing」。

步驟

1. 從Astra Control左側導覽中、選取* Applications*。
2. 在「應用程式」頁面中、選取「資料保護>*複寫*」索引標籤。
3. 在「Data Protection（資料保護）」>「Replication（複寫）」索引標籤中、從「Actions（動作）」功能表中選取* Resyn美食*。
4. 在「ResSync（重新同步）」頁面中、選取包含您要保留之資料的來源或目的地應用程式執行個體。

> 請謹慎選擇重新同步來源、因為目的地上的資料將被覆寫。

5. 選擇*重新同步*以繼續。
6. 輸入「resSync」以確認。
7. 選取*是、重新同步*以完成。

結果

- 「複寫」頁面會顯示「建立」作為複寫狀態。
- Astra Control會在新的目的地叢集上停止應用程式。
- Astra Control會使用SnapMirror重新同步、在所選方向重新建立持續Volume複寫。
- 「複寫」頁面會顯示更新的關係。

反轉應用程式複寫

這是將應用程式移至目的地叢集、同時繼續複寫回原始來源叢集的計畫性作業。Astra Control會停止來源叢集上的應用程式、並將資料複寫到目的地、然後再將應用程式容錯移轉到目的地叢集。

在這種情況下、您要交換來源和目的地。原始來源叢集會成為新的目的地叢集、而原始目的地叢集會成為新的來源叢集。

步驟

1. 從Astra Control左側導覽中、選取* Applications*。
2. 在「應用程式」頁面中、選取「資料保護>*複寫*」索引標籤。
3. 在「Data Protection（資料保護）」>「Replication（複寫）」索引標籤中、從「Actions（動作）」功能表中、選取「* Reverse Replic
4. 在「Reverse Replication」（反轉複寫）頁面中、檢閱資訊、然後選取* Reverse Replication*繼續。

結果

下列動作是因為反轉複寫而發生：

- 快照是從原始來源應用程式的Kubernetes資源中取得。
- 刪除應用程式的Kubernetes資源（保留PVCS和PVs）、即可順利停止原始來源應用程式的Pod。
- 在Pod關機之後、便會取得並複寫應用程式磁碟區的Snapshot快照。
- SnapMirror關係中斷、使目的地磁碟區準備好進行讀寫。
- 應用程式的Kubernetes資源會使用在原始來源應用程式關閉後複寫的Volume資料、從關機前的Snapshot還原。
- 複寫會以相反方向重新建立。

將應用程式容錯移轉至原始來源叢集

使用Astra Control、您可以使用下列作業順序、在「容錯移轉」作業之後達到「容錯移轉」。在此工作流程中、為了還原原始複寫方向、Astra Control會在反轉複寫方向之前、將任何應用程式變更複寫回原始來源叢集。

此程序從已完成容錯移轉至目的地的關係開始、並涉及下列步驟：

- 從容錯移轉狀態開始。

- 重新同步關係。
- 反轉複寫。

步驟

1. 從Astra Control左側導覽中、選取* Applications*。
2. 在「應用程式」頁面中、選取「資料保護>*複寫*」索引標籤。
3. 在「Data Protection（資料保護）」>「Replication（複寫）」索引標籤中、從「Actions（動作）」功能表中選取* Resyn美食*。
4. 若要執行故障恢復作業、請選擇容錯移轉應用程式作為重新同步作業的來源（保留任何在容錯移轉後寫入的資料）。
5. 輸入「resSync」以確認。
6. 選取*是、重新同步*以完成。
7. 重新同步完成後、請在「Data Protection（資料保護）」>「Replication（複寫）」索引標籤的「Actions（動作）」功能表中、選取* Reverse replection*（反轉複寫）。
8. 在「Reverse Replication」（反轉複寫）頁面中、檢閱資訊並選取* Reverse Replication*。

結果

這將「重新同步」和「反轉關係」作業的結果結合在一起、以便在原始來源叢集上使應用程式上線、並將複寫恢復至原始目的地叢集。

刪除應用程式複寫關係

刪除關係會產生兩個獨立的應用程式、兩者之間沒有任何關係。

步驟

1. 從Astra Control左側導覽中、選取* Applications*。
2. 在「應用程式」頁面中、選取「資料保護>*複寫*」索引標籤。
3. 在Data Protection（資料保護）> Replication（複寫）索引標籤中、從Application Protection（應用程式保護）方塊或關係圖中、選取* Delete Replication election*（刪除複寫關係*）。

結果

刪除複寫關係之後會發生下列動作：

- 如果建立關係、但應用程式尚未在目的地叢集上上線（容錯移轉）、Astra Control會保留初始化期間建立的PVCS、並在目的地叢集上留下「空白」的託管應用程式、並保留目的地應用程式、以保留可能建立的任何備份。
- 如果應用程式已在目的地叢集上線（容錯移轉）、Astra Control會保留PVCS和目的地應用程式。來源和目的地應用程式現在被視為獨立的應用程式。備份排程會保留在兩個應用程式上、但不會彼此關聯。

複寫關係健全狀況狀態和關係生命週期狀態

Astra Control會顯示複寫關係的關係健全狀況、以及複寫關係的生命週期狀態。

複寫關係健全狀況狀態

下列狀態表示複寫關係的健全狀況：

- 正常：這種關係正在建立或已經建立、而且最近的Snapshot已成功傳輸。
- 警告：關係可能是容錯移轉或容錯移轉（因此不再保護來源應用程式）。
- 重大
  - 關係正在建立或容錯移轉、最後一次的協調嘗試失敗。
  - 建立關係、最後一次嘗試協調新增的永久虛擬基礎虛擬基礎虛擬基礎虛擬基礎虛擬基礎虛擬基礎層面時、就會失敗。
  - 建立關係（因此已複寫成功的Snapshot、並可進行容錯移轉）、但最近的Snapshot失敗或無法複寫。

複寫生命週期狀態

下列狀態反映複寫生命週期的不同階段：

- 正在建立：正在建立新的複寫關係。Astra Control會視需要建立命名空間、在目的地叢集的新磁碟區上建立持續磁碟區宣告（PVCS）、並建立SnapMirror關係。此狀態也表示複寫正在重新同步或反轉複寫。
- 已建立：存在複寫關係。Astra Control會定期檢查PVCS是否可用、檢查複寫關係、定期建立應用程式的Snapshot快照、並識別應用程式中的任何新來源PVCS。如果是、Astra Control會建立資源以將其納入複寫中。
- 容錯移轉：Astra Control會中斷SnapMirror關係、並從上次成功複寫的應用程式Snapshot中還原應用程式的Kubernetes資源。
- 故障移轉：Astra Control會停止從來源叢集複寫、在目的地使用最新（成功）的複寫應用程式Snapshot、並還原Kubernetes資源。
- 重新同步：Astra Control使用SnapMirror重新同步、將重新同步來源上的新資料重新同步至重新同步目的地。此作業可能會根據同步方向覆寫目的地上的部分資料。Astra Control會停止在目的地命名空間上執行的應用程式、並移除Kubernetes應用程式。在重新同步程序期間、狀態會顯示為「Establing（正在建立）」。
- 反轉：是將應用程式移至目的地叢集、同時繼續複寫回原始來源叢集的計畫性作業。Astra Control會停止來源叢集上的應用程式、將資料複寫到目的地、然後再將應用程式容錯移轉到目的地叢集。在反向複寫期間、狀態會顯示為「Establing（正在建立）」。
- 刪除：
  - 如果複寫關係已建立但尚未容錯移轉、Astra Control會移除複寫期間建立的PVCS、並刪除目的地託管應用程式。
  - 如果複寫已失敗、Astra Control會保留PVCS和目的地應用程式。

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。

```
oc new-project wordpress
oc adm policy add-scc-to-group anyuid system:serviceaccounts:wordpress
oc adm policy add-scc-to-user privileged -z default -n wordpress
```

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位於時、即表示複製作業成功 `Available` 請在「應用程式」頁面上說明。

> 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

執行攔截是一種自訂動作、可設定搭配託管應用程式的資料保護作業一起執行。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。

執行掛勾的類型

Astra Control支援下列類型的執行掛勾、視執行時間而定：

- 快照前
- 快照後
- 預先備份
- 備份後
- 還原後

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。

- 執行攔截必須使用指令碼來執行動作。許多執行掛勾可以參照相同的指令碼。
- Astra Control需要執行掛勾所使用的指令碼、以執行Shell指令碼的格式寫入。
- 指令碼大小上限為96KB。
- Astra Control使用執行掛勾設定及任何符合條件、來判斷哪些掛勾適用於快照、備份或還原作業。
- 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資料保護作業。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。
- 對於Adhoc*資料保護作業、所有Hook事件都會產生並儲存在*活動*頁面事件記錄中。不過、對於排程的資料保護作業、事件記錄中只會記錄攔截故障事件（排程資料保護作業本身所產生的事件仍會記錄下來）。

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。如果您以相關的執行掛勾開始備份或快照作業、但隨後取消它、則如果備份或快照作業已經開始、仍允許掛勾執行。這表示備份後執行掛勾無法假設備份已完成。

執行順序

執行資料保護作業時、執行掛機事件會依照下列順序發生：

1. 任何適用的自訂操作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操作前掛勾、但在作業之前執行這些掛勾的順序既不保證也無法設定。
2. 執行資料保護作業。
3. 任何適用的自訂操作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後置作業掛勾、但在作業後執行這些掛勾的順序並不保證也無法設定。

如果您建立同一類型的多個執行掛勾（例如預先快照）、則無法保證這些掛勾的執行順序。不過、不同類型的掛勾的執行順序也有保證。例如、具有所有五種不同類型掛勾的組態執行順序如下所示：

1. 執行備份前掛勾
2. 執行快照前掛勾
3. 快照後掛勾已執行
4. 執行備份後掛勾
5. 執行還原後的掛勾

如需此組態的範例、請參閱中表格的案例編號2 [確定掛機是否會執行]。

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和備份、以確保它們一致。您可以將應用程式複製到暫用命名空間、還原快照或備份、然後測試應用程式、藉此完成此作業。

確定掛機是否會執行

請使用下表協助判斷您的應用程式是否會執行自訂執行掛勾。

請注意、所有的高階應用程式作業都是執行快照、備份或還原等基本作業之一。視案例而定、複製作業可能由這些作業的各種組合組成、因此複製作業執行的執行掛勾內容會有所不同。

就地還原作業需要現有的快照或備份、因此這些作業不會執行快照或備份掛勾。

如果您先開始、然後取消包含快照的備份、並有相關的執行掛勾、有些掛勾可能會執行、有些則不會執行。這表示備份後執行掛勾無法假設備份已完成。請謹記以下幾點、以相關的執行掛勾來取消備份：

- 備份前和備份後的掛勾一律會執行。
- 如果備份包含新的快照、而且快照已啟動、則會執行快照前和快照後的掛勾。
- 如果在快照開始之前取消備份、則不會執行快照前和快照後掛勾。

案例	營運	現有快照	現有備份	命名空間	叢集	**Snapshot hooks**會執行	備份掛勾運轉	執行還原掛勾
1.	複製	n	n	新功能	相同	是	n	是
2.	複製	n	n	新功能	與眾不同	是	是	是
3.	複製或還原	是	n	新功能	相同	n	n	是
4.	複製或還原	n	是	新功能	相同	n	n	是
5.	複製或還原	是	n	新功能	與眾不同	n	是	是
6.	複製或還原	n	是	新功能	與眾不同	n	n	是
7.	還原	是	n	現有的	相同	n	n	是
8.	還原	n	是	現有的	相同	n	n	是
9.	Snapshot	不適用	不適用	不適用	不適用	是	不適用	不適用
10.	備份	n	不適用	不適用	不適用	是	是	不適用
11.	備份	是	不適用	不適用	不適用	n	是	不適用

檢視現有的執行掛勾

您可以檢視應用程式的現有自訂執行掛勾。

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。您可以查看某個掛機的狀態、來源、以及其執行時間（作業前或作業後）。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。

檢視現有的指令碼

您可以檢視現有上傳的指令碼。您也可以在此頁面上查看使用中的指令碼、以及使用這些指令碼的攔截器。

步驟

1. 前往*帳戶*。
2. 選取*指令碼*索引標籤。

   您可以在此頁面上看到現有上傳指令碼的清單。「使用者」欄會顯示每個指令碼使用的執行掛勾。

新增指令碼

您可以新增一個或多個執行掛勾可以參考的指令碼。許多執行掛勾可以參照相同的指令碼、只要變更一個指令碼、就能更新許多執行掛勾。

步驟

1. 前往*帳戶*。
2. 選取*指令碼*索引標籤。
3. 選取*「Add*」。
4. 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。
     v. 選取*儲存指令碼*。
   - 從剪貼簿貼入自訂指令碼。
     i. 選取*貼上或類型*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。
5. 選取*儲存指令碼*。

結果

新指令碼會出現在「指令碼」索引標籤的清單中。

刪除指令碼

如果指令碼不再需要、也不被任何執行掛勾使用、您可以從系統中移除指令碼。

步驟

1. 前往*帳戶*。
2. 選取*指令碼*索引標籤。
3. 選擇要移除的指令碼、然後在*「Actions」（動作）*欄中選取功能表。
4. 選擇*刪除*。

> 如果指令碼與一個或多個執行掛勾相關聯、則無法使用*刪除*動作。若要刪除指令碼、請先編輯相關的執行掛勾、然後將其與其他指令碼建立關聯。

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、除非您執行特定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*「Add*」。
4. 在「勾選詳細資料」區域中、從*操作*下拉式功能表中選取作業類型、以決定掛機的執行時間。
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：
   - 新增指令碼。
     i. 選取*「Add*」。
     ii. 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。
         V. 選取*儲存指令碼*。
       - 從剪貼簿貼入自訂指令碼。
         I. 選取*貼上或類型*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。
   - 從清單中選取現有的指令碼。

     這會指示執行掛勾使用此指令碼。
9. 選取*新增攔截*。

檢查執行掛勾的狀態

在快照、備份或還原作業完成執行之後、您可以檢查執行掛勾的狀態、該掛勾是執行作業的一部分。您可以使用此狀態資訊來判斷是否要保留執行掛勾、修改或刪除它。

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*資料保護*索引標籤。
3. 選取* Snapshot*以查看執行中的快照、或選取*備份*以查看執行中的備份。

   「掛機狀態」會顯示執行掛機在作業完成後執行的狀態。您可以將游標暫留在狀態上、以取得更多詳細資料。例如、如果快照期間發生執行掛機故障、則將游標移到該快照的掛機狀態上會顯示故障執行掛勾的清單。若要查看每次失敗的原因、您可以查看左側導覽區域的*活動*頁面。

檢視指令碼使用量

您可以在Astra Control Web UI中查看哪些執行掛勾使用特定指令碼。

步驟

1. 選擇*帳戶*。
2. 選取*指令碼*索引標籤。

   指令碼清單中的「使用者」欄位包含清單中每個指令碼所使用之掛勾的詳細資料。

3. 在「使用者」欄中選取您感興趣的指令碼資訊。

   此時會出現更詳細的清單、其中包含使用指令碼的掛勾名稱、以及設定用來執行的作業類型。

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
```