You're absolutely right! If your agents are running on Red Hat machines, PowerShell won't be natively available or the preferred scripting language. We need to switch the PowerShell@2 tasks to Bash@3 and use curl and jq for interacting with the Azure DevOps Server REST API. Here's the updated YAML.

Key Changes Made:

1. **PowerShell@2 replaced with Bash@3:** All tasks that previously used PowerShell are now Bash tasks.
2. **curl and jq for API Interaction:**
    - curl is used for making HTTP requests to the Azure DevOps Server REST API.
    - jq is used for parsing and modifying JSON responses/requests. **It's crucial that jq is installed on your Red Hat agent machines.** A step to install jq has been added for robustness.
3. **Bash Scripting Logic:**
    - Error handling now uses $? for exit codes.
    - Base64 encoding for PATs is done using echo -n ":$PAT" | base64.
    - Conditional logic for adding/updating CroninSuspend in the Variable Group JSON is handled with jq.

Prerequisites & Setup Steps (Revised for Red Hat Agents)

1. **Agent Setup:**
    - Ensure your Red Hat agent machines have curl installed (usually default).
    - **Crucially, ensure jq is installed.** You can add sudo dnf install -y jq or sudo yum install -y jq to your agent provisioning script, or rely on the pipeline to install it (as added in the YAML).
2. **Azure DevOps Server Configuration (Same as before):**
    - **Service Connections:** Kubernetes connections for your AKS clusters (still in Azure Public Cloud).
    - **Variable Group IDs:** For your WEU and NEU regions in your *current* Azure DevOps Server collection.
    - **Other Azure DevOps Server Collection (if applicable):**
        - **Generate a PAT:** From the *other* Azure DevOps Server instance/collection. Grant Variable Groups -> Read & Write permissions.
        - **Secret Variable Group (DRSecrets):** In your *current* Azure DevOps Server collection, create a secret variable group named DRSecrets containing your OtherOrgDevOpsPat PAT. Link this group to your pipeline.
        - **otherOrgUrl Variable:** Set this to the *exact* URL of the other Azure DevOps Server collection (e.g., http://your-ados-server:8080/tfs/YourOtherCollection).
3. **Update Placeholders in YAML:**
    - Replace all your-... placeholders.
    - Verify adoServerApiVersion matches your Azure DevOps Server version.

The Complete YAML Pipeline (for Red Hat Agents)

```
# azure-pipelines.yml
```

```
# Link your secret variable group here for cross-organization PAT.
# This group must contain the 'OtherOrgDevOpsPat' secret variable.
# Ensure this variable group is linked in your pipeline settings as
well.
variables:
- group: DRSecrets

trigger: none # This pipeline will be triggered manually

parameters:
  - name: activeRegion
    displayName: Target Active Region
    type: string
    default: 'weu' # Default to WEU as the active region
    values:
      - 'weu' # Represents Primary Region
      - 'neu' # Represents DR Region

# --- Configuration Variables (REPLACE WITH YOUR ACTUAL VALUES) ---
variables:
  # Current Organization/Collection Service Connections for AKS
  weuKubernetesServiceConnection:
'your-weu-aks-service-connection-name'
  neuKubernetesServiceConnection:
'your-neu-aks-service-connection-name'

  # Current Organization/Collection Variable Group IDs
  weuVariableGroupId: 123 # Replace with your WEU environment variable
group ID
  neuVariableGroupId: 456 # Replace with your NEU environment variable
group ID

  # AKS Cluster & Resource Group Names (for patching)
  weuAksResourceGroup: 'your-weu-aks-resource-group-name'
  weuAksClusterName: 'your-weu-aks-cluster-name'
  neuAksResourceGroup: 'your-neu-aks-resource-group-name'
  neuAksClusterName: 'your-neu-aks-cluster-name'

  # Naming convention for your CronJobs and their namespaces
  # The kubectl command expects namespaces that start with this
prefix.
  # Adjust `your-app-namespace-prefix` to match your actual namespaces
(e.g., 'prod-app-', 'default').
  cronJobNamespacePrefix: 'your-app-namespace-prefix'

  # Other Azure DevOps Server Collection Variables (Leave blank/0 if
you don't have another collection to update)
  # IMPORTANT: This URL must be the base URL for the OTHER Azure
```

```
DevOps Server Collection.
  # Example: 'http://your-ados-server:8080/tfs/YourOtherCollection' or
'http://another-ados-server:8080/tfs/DefaultCollection'
  otherOrgUrl: 'http://your-ados-server:8080/tfs/YourOtherCollection'
  otherOrgProjectName: 'your-other-org-project-name' # e.g.,
'MyOtherProjectInOtherCollection'
  otherOrgVariableGroupId: 789 # Replace with the Variable Group ID in
the other collection/project

  # Azure DevOps Server REST API Version - IMPORTANT: VERIFY THIS FOR
YOUR SERVER VERSION!
  # Common versions: 7.1 (ADO Server 2022), 6.0 (ADO Server 2020), 5.1
(ADO Server 2019)
  adoServerApiVersion: '6.0'
# --- END OF CONFIGURATION VARIABLES ---


stages:
  - stage: PreSwitchoverApproval
    displayName: '1. Pre-Switchover Manual Approval'
    jobs:
      - job: AwaitOperatorApproval
        displayName: 'Awaiting Operator Approval to Proceed'
        pool:
          vmImage: 'ubuntu-latest' # Use 'ubuntu-latest' or your
specific Red Hat agent image label
        steps:
          # Ensure jq is installed on the agent.
          - script: |
              echo "Checking for 'jq' installation..."
              if ! command -v jq &> /dev/null
              then
                  echo "jq not found. Installing..."
                  # Use the appropriate package manager for Red Hat
(dnf/yum)
                  sudo dnf install -y jq || sudo yum install -y jq
                  if [ $? -ne 0 ]; then
                    echo "##vso[task.logissue type=error]Failed to
install 'jq'. Please ensure 'jq' is pre-installed or agent has
permissions."
                    exit 1
                  fi
                  echo "'jq' installed successfully."
              else
                  echo "'jq' is already installed."
              fi
            displayName: 'Ensure jq is installed'
            condition: always() # Always try to install jq
```

```
          - task: ManualValidation@0
            displayName: 'Review and Approve DR Switchover Execution'
            inputs:
              instructions: |
                ## Manual Intervention Required for DR Switchover

                **Target Active Region:** `${{ parameters.activeRegion
}}`

                Please carefully review the proposed switchover:

                1.  **Inactive Region (WEU/NEU):** CronJobs will be
**suspended**.
                2.  **Active Region (WEU/NEU):** CronJobs will be
**resumed**.
                3.  **Variable Groups:** `CroninSuspend` variable will
be updated (`true` for inactive, `false` for active) in both current
and other Azure DevOps Server collections.

                **Confirm if you want to proceed.** Click 'Resume' to
continue the switchover process, or 'Cancel' to abort.
              # Optional: Email notifications
              # notifyUsers:
'operator1@example.com,operator2@example.com'
              # timeoutInMinutes: 1440 # 24 hours to approve (adjust
as needed)

  - stage: ExecuteDRSwitchover
    displayName: '2. Execute DR Switchover Operations'
    dependsOn: PreSwitchoverApproval # This stage runs only after the
approval stage
    condition: succeeded('PreSwitchoverApproval') # Only proceed if
the approval stage succeeded

    jobs:
      - job: SuspendInactiveRegionJobs
        displayName: 'Suspend CronJobs & Update VG in Inactive Region'
        pool:
          vmImage: 'ubuntu-latest' # Use 'ubuntu-latest' or your
specific Red Hat agent image label

        steps:
          - script: |
              echo "--- Initiating suspension for inactive region ---"
              echo "Target Active Region: ${{ parameters.activeRegion
}}"
              echo "Inactive Region: ${{ parameters.activeRegion ==
```

```
'weu' && 'NEU' || 'WEU' }}"
            displayName: 'Display Current Action Context'

          - task: Kubernetes@1
            displayName: 'Patch CronJobs to Suspend in Inactive
Region'
            inputs:
              connectionType: 'Azure Resource Manager'
              azureSubscriptionEndpoint: ${{ parameters.activeRegion
== 'weu' && variables.neuKubernetesServiceConnection ||
variables.weuKubernetesServiceConnection }}
              azureResourceGroup: ${{ parameters.activeRegion == 'weu'
&& variables.neuAksResourceGroup || variables.weuAksResourceGroup }}
              kubernetesCluster: ${{ parameters.activeRegion == 'weu'
&& variables.neuAksClusterName || variables.weuAksClusterName }}
              command: 'kubectl'
              arguments: |
                # Get cronjobs in namespaces starting with the prefix
and loop through them
                kubectl get cronjobs -n $(cronJobNamespacePrefix)* -o
name | while read cronjob_full_name; do
                  NAMESPACE=$(echo $cronjob_full_name | cut -d'/' -f1
| sed 's/cronjob.batch\///') # Extract namespace
                  CRONJOB_NAME=$(echo $cronjob_full_name | cut -d'/'
-f2) # Extract cronjob name

                  echo "Attempting to suspend CronJob: $CRONJOB_NAME
in namespace: $NAMESPACE"
                  kubectl patch cronjob $CRONJOB_NAME -n $NAMESPACE -p
'{"spec" : {"suspend" : true}}'

                  if [ $? -eq 0 ]; then
                    echo "✅ Successfully suspended CronJob:
$CRONJOB_NAME in namespace: $NAMESPACE."
                  else
                    echo "❌ Failed to suspend CronJob: $CRONJOB_NAME
in namespace: $NAMESPACE."
                    echo "##vso[task.logissue type=error]Failed to
suspend CronJob: $CRONJOB_NAME in namespace: $NAMESPACE."
                    # For DR, often continueOnError is preferred to
attempt all steps.
                  fi
                done
            continueOnError: true # Continue even if some cronjobs
fail to patch
            name: suspendResult

          - task: Bash@3
```

```
            displayName: 'Update Variable Group in Inactive Region to
Suspend (Current Collection)'
            inputs:
              targetType: 'inline'
              script: |
                ORGANIZATION_URL="$(System.CollectionUri)"
                PROJECT_NAME="$(System.TeamProject)"
                VARIABLE_GROUP_ID=${{ parameters.activeRegion == 'weu'
&& variables.neuVariableGroupId || variables.weuVariableGroupId }}
                PAT="$(System.AccessToken)"
                API_VERSION="$(adoServerApiVersion)"
                TARGET_SUSPEND_VALUE="true"

                if [ -z "$PAT" ]; then
                    echo "##vso[task.logissue type=error]❌
System.AccessToken is not available. Ensure pipeline permissions are
correct."
                    exit 1
                fi

                AUTH_HEADER="Authorization: Basic $(echo -n ":$PAT" |
base64)"
                CONTENT_TYPE_HEADER="Content-Type: application/json"


VARIABLE_GROUP_URL="${ORGANIZATION_URL}${PROJECT_NAME}/_apis/distribut
edtask/variablegroups/${VARIABLE_GROUP_ID}?api-version=${API_VERSION}"

                echo "ℹ️ Fetching variable group ${VARIABLE_GROUP_ID}
from current collection..."
                VG_JSON=$(curl -s -X GET -H "${AUTH_HEADER}" -H
"${CONTENT_TYPE_HEADER}" "${VARIABLE_GROUP_URL}")

                if [ $? -ne 0 ] || [ -z "$VG_JSON" ]; then
                    echo "❌ Failed to fetch variable group
${VARIABLE_GROUP_ID}. Response: ${VG_JSON}"
                    echo "##vso[task.logissue type=error]Failed to
fetch variable group ${VARIABLE_GROUP_ID}."
                    exit 1
                fi

                echo "✅ Successfully fetched variable group."

                # Check if CroninSuspend exists and update/add it
                if echo "$VG_JSON" | jq -e '.variables.CroninSuspend'
> /dev/null; then
                    echo "ℹ️ CroninSuspend variable found. Setting
value to '${TARGET_SUSPEND_VALUE}'."
```

```
                    UPDATED_VG_JSON=$(echo "$VG_JSON" | jq
".variables.CroninSuspend.value = \"${TARGET_SUSPEND_VALUE}\"")
                else
                    echo "⚠️ CroninSuspend variable not found. Adding
it with value '${TARGET_SUSPEND_VALUE}'."
                    UPDATED_VG_JSON=$(echo "$VG_JSON" | jq
".variables.CroninSuspend = {\"value\": \"${TARGET_SUSPEND_VALUE}\",
\"isSecret\": false}")
                fi

                if [ $? -ne 0 ] || [ -z "$UPDATED_VG_JSON" ]; then
                    echo "❌ Failed to modify JSON for variable group
${VARIABLE_GROUP_ID}."
                    echo "##vso[task.logissue type=error]Failed to
modify JSON."
                    exit 1
                fi

                echo "ℹ️ Updating variable group
${VARIABLE_GROUP_ID}..."
                UPDATE_RESPONSE=$(echo "$UPDATED_VG_JSON" | curl -s -X
PUT -H "${AUTH_HEADER}" -H "${CONTENT_TYPE_HEADER}" --data @-
"${VARIABLE_GROUP_URL}")

                if [ $? -ne 0 ]; then
                    echo "❌ Failed to update variable group
${VARIABLE_GROUP_ID}. Response: ${UPDATE_RESPONSE}"
                    echo "##vso[task.logissue type=error]Failed to
update variable group ${VARIABLE_GROUP_ID}."
                    exit 1
                fi

                echo "✅ Successfully updated variable group
${VARIABLE_GROUP_ID} in current collection:
CroninSuspend='${TARGET_SUSPEND_VALUE}'."
                echo "##vso[task.setvariable
variable=vgUpdateStatus;isOutput=true]Succeeded"
            name: inactiveVgUpdateResult # Set an output variable for
status reporting
            continueOnError: true # Log error but continue pipeline
execution if VG update fails


      - job: ResumeActiveRegionJobs
        displayName: 'Resume CronJobs & Update VG in Active Region'
        pool:
          vmImage: 'ubuntu-latest' # Use 'ubuntu-latest' or your
specific Red Hat agent image label
```

```
        dependsOn: SuspendInactiveRegionJobs # Run this after inactive
region suspension starts

        steps:
          - script: |
              echo "--- Initiating resumption for active region ---"
              echo "Target Active Region: ${{ parameters.activeRegion
}}"
              echo "Active Region: ${{ parameters.activeRegion ==
'weu' && 'WEU' || 'NEU' }}"
            displayName: 'Display Current Action Context'

          - task: Kubernetes@1
            displayName: 'Patch CronJobs to Resume in Active Region'
            inputs:
              connectionType: 'Azure Resource Manager'
              azureSubscriptionEndpoint: ${{ parameters.activeRegion
== 'weu' && variables.weuKubernetesServiceConnection ||
variables.neuKubernetesServiceConnection }}
              azureResourceGroup: ${{ parameters.activeRegion == 'weu'
&& variables.weuAksResourceGroup || variables.neuAksResourceGroup }}
              kubernetesCluster: ${{ parameters.activeRegion == 'weu'
&& variables.weuAksClusterName || variables.neuAksClusterName }}
              command: 'kubectl'
              arguments: |
                # Get cronjobs in namespaces starting with the prefix
and loop through them
                kubectl get cronjobs -n $(cronJobNamespacePrefix)* -o
name | while read cronjob_full_name; do
                  NAMESPACE=$(echo $cronjob_full_name | cut -d'/' -f1
| sed 's/cronjob.batch\///') # Extract namespace
                  CRONJOB_NAME=$(echo $cronjob_full_name | cut -d'/'
-f2) # Extract cronjob name

                  echo "Attempting to resume CronJob: $CRONJOB_NAME in
namespace: $NAMESPACE"
                  kubectl patch cronjob $CRONJOB_NAME -n $NAMESPACE -p
'{"spec" : {"suspend" : false}}'

                  if [ $? -eq 0 ]; then
                    echo "✅ Successfully resumed CronJob:
$CRONJOB_NAME in namespace: $NAMESPACE."
                  else
                    echo "❌ Failed to resume CronJob: $CRONJOB_NAME
in namespace: $NAMESPACE."
                    echo "##vso[task.logissue type=error]Failed to
resume CronJob: $CRONJOB_NAME in namespace: $NAMESPACE."
                  fi
```

```
                done
            continueOnError: true
            name: resumeResult

          - task: Bash@3
            displayName: 'Update Variable Group in Active Region to
Resume (Current Collection)'
            inputs:
              targetType: 'inline'
              script: |
                ORGANIZATION_URL="$(System.CollectionUri)"
                PROJECT_NAME="$(System.TeamProject)"
                VARIABLE_GROUP_ID=${{ parameters.activeRegion == 'weu'
&& variables.weuVariableGroupId || variables.neuVariableGroupId }}
                PAT="$(System.AccessToken)"
                API_VERSION="$(adoServerApiVersion)"
                TARGET_SUSPEND_VALUE="false"

                if [ -z "$PAT" ]; then
                    echo "##vso[task.logissue type=error]❌
System.AccessToken is not available. Ensure pipeline permissions are
correct."
                    exit 1
                fi

                AUTH_HEADER="Authorization: Basic $(echo -n ":$PAT" |
base64)"
                CONTENT_TYPE_HEADER="Content-Type: application/json"


VARIABLE_GROUP_URL="${ORGANIZATION_URL}${PROJECT_NAME}/_apis/distribut
edtask/variablegroups/${VARIABLE_GROUP_ID}?api-version=${API_VERSION}"

                echo "ℹ️ Fetching variable group ${VARIABLE_GROUP_ID}
from current collection..."
                VG_JSON=$(curl -s -X GET -H "${AUTH_HEADER}" -H
"${CONTENT_TYPE_HEADER}" "${VARIABLE_GROUP_URL}")

                if [ $? -ne 0 ] || [ -z "$VG_JSON" ]; then
                    echo "❌ Failed to fetch variable group
${VARIABLE_GROUP_ID}. Response: ${VG_JSON}"
                    echo "##vso[task.logissue type=error]Failed to
fetch variable group ${VARIABLE_GROUP_ID}."
                    exit 1
                fi

                echo "✅ Successfully fetched variable group."
```

```
                # Check if CroninSuspend exists and update/add it
                if echo "$VG_JSON" | jq -e '.variables.CroninSuspend'
> /dev/null; then
                    echo "ℹ️ CroninSuspend variable found. Setting
value to '${TARGET_SUSPEND_VALUE}'."
                    UPDATED_VG_JSON=$(echo "$VG_JSON" | jq
".variables.CroninSuspend.value = \"${TARGET_SUSPEND_VALUE}\"")
                else
                    echo "⚠️ CroninSuspend variable not found. Adding
it with value '${TARGET_SUSPEND_VALUE}'."
                    UPDATED_VG_JSON=$(echo "$VG_JSON" | jq
".variables.CroninSuspend = {\"value\": \"${TARGET_SUSPEND_VALUE}\",
\"isSecret\": false}")
                fi

                if [ $? -ne 0 ] || [ -z "$UPDATED_VG_JSON" ]; then
                    echo "❌ Failed to modify JSON for variable group
${VARIABLE_GROUP_ID}."
                    echo "##vso[task.logissue type=error]Failed to
modify JSON."
                    exit 1
                fi

                echo "ℹ️ Updating variable group
${VARIABLE_GROUP_ID}..."
                UPDATE_RESPONSE=$(echo "$UPDATED_VG_JSON" | curl -s -X
PUT -H "${AUTH_HEADER}" -H "${CONTENT_TYPE_HEADER}" --data @-
"${VARIABLE_GROUP_URL}")

                if [ $? -ne 0 ]; then
                    echo "❌ Failed to update variable group
${VARIABLE_GROUP_ID}. Response: ${UPDATE_RESPONSE}"
                    echo "##vso[task.logissue type=error]Failed to
update variable group ${VARIABLE_GROUP_ID}."
                    exit 1
                fi

                echo "✅ Successfully updated variable group
${VARIABLE_GROUP_ID} in current collection:
CroninSuspend='${TARGET_SUSPEND_VALUE}'."
                echo "##vso[task.setvariable
variable=vgUpdateStatus;isOutput=true]Succeeded"
            name: activeVgUpdateResult # Set an output variable for
status reporting
            continueOnError: true # Log error but continue pipeline
execution if VG update fails
```

```
      - job: UpdateOtherOrgVariableGroup
        displayName: 'Update Variable Group in Other Azure DevOps
Server Collection'
        pool:
          vmImage: 'ubuntu-latest' # Use 'ubuntu-latest' or your
specific Red Hat agent image label
        dependsOn: SuspendInactiveRegionJobs # No strict dependency on
resume, can run in parallel with resume or after.
        # This job will run only if otherOrgUrl, otherOrgProjectName,
and otherOrgVariableGroupId are defined.
        # This allows you to exclude it by leaving these variables
empty/0 if not needed.
        condition: and(succeeded(), ne(variables.otherOrgUrl, ''),
ne(variables.otherOrgProjectName, ''),
ne(variables.otherOrgVariableGroupId, 0))

        steps:
          - script: |
              echo "--- Initiating Variable Group update in other
Azure DevOps Server collection ---"
              echo "Other Collection URL: $(otherOrgUrl)"
              echo "Other Collection Project: $(otherOrgProjectName)"
              echo "Other Collection Variable Group ID:
$(otherOrgVariableGroupId)"
            displayName: 'Display Other Collection Context'

          - task: Bash@3
            displayName: 'Execute Update for Other Collection Variable
Group'
            inputs:
              targetType: 'inline'
              script: |
                ORGANIZATION_URL="$(otherOrgUrl)"
                PROJECT_NAME="$(otherOrgProjectName)"
                VARIABLE_GROUP_ID="$(otherOrgVariableGroupId)"
                PAT="$(OtherOrgDevOpsPat)" # This variable comes from
the linked secret variable group!
                API_VERSION="$(adoServerApiVersion)"
                TARGET_SUSPEND_VALUE="false" # Always set to false for
the *active* state in the other collection's pipelines

                if [ -z "$PAT" ]; then
                    echo "##vso[task.logissue type=error]❌
OtherOrgDevOpsPat is not set or is empty. Cannot proceed with
cross-collection update. Ensure 'DRSecrets' variable group is linked
and 'OtherOrgDevOpsPat' is defined as a secret."
                    exit 1
                fi
```

```
                AUTH_HEADER="Authorization: Basic $(echo -n ":$PAT" |
base64)"
                CONTENT_TYPE_HEADER="Content-Type: application/json"


VARIABLE_GROUP_URL="${ORGANIZATION_URL}${PROJECT_NAME}/_apis/distribut
edtask/variablegroups/${VARIABLE_GROUP_ID}?api-version=${API_VERSION}"

                echo "ℹ️ Fetching variable group ${VARIABLE_GROUP_ID}
from other collection..."
                VG_JSON=$(curl -s -X GET -H "${AUTH_HEADER}" -H
"${CONTENT_TYPE_HEADER}" "${VARIABLE_GROUP_URL}")

                if [ $? -ne 0 ] || [ -z "$VG_JSON" ]; then
                    echo "❌ Failed to fetch variable group
${VARIABLE_GROUP_ID}. Response: ${VG_JSON}"
                    echo "##vso[task.logissue type=error]Failed to
fetch variable group ${VARIABLE_GROUP_ID}."
                    exit 1
                fi

                echo "✅ Successfully fetched variable group."

                # Check if CroninSuspend exists and update/add it
                if echo "$VG_JSON" | jq -e '.variables.CroninSuspend'
> /dev/null; then
                    echo "ℹ️ CroninSuspend variable found. Setting
value to '${TARGET_SUSPEND_VALUE}'."
                    UPDATED_VG_JSON=$(echo "$VG_JSON" | jq
".variables.CroninSuspend.value = \"${TARGET_SUSPEND_VALUE}\"")
                else
                    echo "⚠️ CroninSuspend variable not found. Adding
it with value '${TARGET_SUSPEND_VALUE}'."
                    UPDATED_VG_JSON=$(echo "$VG_JSON" | jq
".variables.CroninSuspend = {\"value\": \"${TARGET_SUSPEND_VALUE}\",
\"isSecret\": false}")
                fi

                if [ $? -ne 0 ] || [ -z "$UPDATED_VG_JSON" ]; then
                    echo "❌ Failed to modify JSON for variable group
${VARIABLE_GROUP_ID}."
                    echo "##vso[task.logissue type=error]Failed to
modify JSON."
                    exit 1
                fi

                echo "ℹ️ Updating variable group
```

```
${VARIABLE_GROUP_ID}..."
                UPDATE_RESPONSE=$(echo "$UPDATED_VG_JSON" | curl -s -X
PUT -H "${AUTH_HEADER}" -H "${CONTENT_TYPE_HEADER}" --data @-
"${VARIABLE_GROUP_URL}")

                if [ $? -ne 0 ]; then
                    echo "❌ Failed to update variable group
${VARIABLE_GROUP_ID}. Response: ${UPDATE_RESPONSE}"
                    echo "##vso[task.logissue type=error]Failed to
update variable group ${VARIABLE_GROUP_ID}."
                    exit 1
                fi

                echo "✅ Successfully updated variable group
${VARIABLE_GROUP_ID} in other collection:
CroninSuspend='${TARGET_SUSPEND_VALUE}'."
                echo "##vso[task.setvariable
variable=vgUpdateStatus;isOutput=true]Succeeded"
            name: otherOrgVgUpdateResult # Set an output variable for
status reporting
            continueOnError: true

  - stage: FinalStatusReport
    displayName: '3. Final DR Switchover Status Report'
    dependsOn: ExecuteDRSwitchover # Ensure this runs after the
execution stage
    condition: always() # This stage should run regardless of previous
job successes/failures

    jobs:
      - job: ReportSummary
        displayName: 'Generate Summary Report'
        pool:
          vmImage: 'ubuntu-latest' # Use 'ubuntu-latest' or your
specific Red Hat agent image label
        steps:
          - script: |
              echo "--- DR Switchover Execution Summary ---"
              echo "##vso[task.setvariable
variable=activeRegionReport]${{ parameters.activeRegion }}"
              echo "Target Active Region: ${{ parameters.activeRegion
}}"
              echo ""
              echo "=== Inactive Region (Suspension) Status ==="
              echo "CronJob Suspension:
$(SuspendInactiveRegionJobs.suspendResult.suspendStatus)" # From
output variable
              echo "Variable Group Update (Current Collection):
```

```
$(SuspendInactiveRegionJobs.inactiveVgUpdateResult.vgUpdateStatus)"
              echo ""
              echo "=== Active Region (Resumption) Status ==="
              echo "CronJob Resumption:
$(ResumeActiveRegionJobs.resumeResult.resumeStatus)"
              echo "Variable Group Update (Current Collection):
$(ResumeActiveRegionJobs.activeVgUpdateResult.vgUpdateStatus)"
              echo ""

              if [ "${{ variables.otherOrgUrl }}" != "" ] && [ "${{
variables.otherOrgProjectName }}" != "" ] && [ "${{
variables.otherOrgVariableGroupId }}" != "0" ]; then
                echo "=== Other Azure DevOps Server Collection Status
==="
                echo "Variable Group Update:
$(UpdateOtherOrgVariableGroup.otherOrgVgUpdateResult.vgUpdateStatus)"
              else
                echo "=== Other Azure DevOps Server Collection:
Skipped (Configuration missing) ==="
              fi
              echo "--------------------------------------"
            displayName: 'Print Overall Switchover Status'
```