The user has uploaded an image describing a scenario for automating failover between Azure Service Bus instances in two regions (SB-A in region 1 and SB-B in region 2) using Azure DevOps YAML pipelines. The process involves several steps with manual intervention.
Here's a breakdown of the steps and how they can be implemented or addressed within an Azure DevOps YAML pipeline:

**Overall Approach:**

The core idea is to use Azure CLI tasks (AzureCLI@2) within your YAML pipeline to execute the necessary az servicebus commands. For manual decisions, you'll leverage Approval gates or Manual Validation tasks.

**Step-by-Step Implementation in Azure DevOps YAML Pipeline:**

```
# Azure DevOps YAML Pipeline for Service Bus Failover

trigger: none # Manual trigger or specific branch triggers

pool:
  vmImage: 'ubuntu-latest' # Or 'windows-latest' if preferred

variables:
  serviceBusNamespaceA: 'SB-A' # Replace with your actual SB-A
namespace name
  resourceGroupA: 'your-rg-a' # Replace with your actual RG for SB-A
  serviceBusNamespaceB: 'SB-B' # Replace with your actual SB-B
namespace name
  resourceGroupB: 'your-rg-b' # Replace with your actual RG for SB-B
  aliasName: 'SB-PAIR-AB' # The alias name for your geo-recovery
configuration
  locationA: 'region1' # Example: 'eastus'
  locationB: 'region2' # Example: 'westus'

stages:
- stage: PreFailoverChecks
  displayName: 'Pre-Failover Checks and Pairing'
  jobs:
  - job: CheckPairingStatus
    displayName: 'Check Service Bus Geo-Recovery Pairing Status'
    steps:
    - task: AzureCLI@2
      displayName: 'Check existing pairing (SB-A to SB-B)'
      inputs:
        azureSubscription: 'YourAzureServiceConnection' # Replace with
your Azure Service Connection Name
        scriptType: 'bash'
        scriptLocation: 'inlineScript'
        inlineScript: |
          echo "Checking geo-recovery pairing status for alias:
$aliasName"
          az servicebus georecovery-alias show --alias $aliasName
--namespace-name $(serviceBusNamespaceA) --resource-group
```

```
$(resourceGroupA) --query 'partnerNamespace' -o tsv >
partner_namespace.txt
          partner_namespace=$(cat partner_namespace.txt)

          if [[ "$partner_namespace" == "$(serviceBusNamespaceB)" ]];
then
            echo "##vso[task.setvariable variable=isPaired]true"
            echo "Service Bus namespaces are already paired (SB-A to
SB-B)."
          else
            echo "##vso[task.setvariable variable=isPaired]false"
            echo "Service Bus namespaces are NOT paired (SB-A to
SB-B)."
          fi
      name: CheckPairing

    # Step 2: If not paired, pair the 2 instances
    - task: AzureCLI@2
      displayName: 'Pair Service Bus Instances (if not paired)'
      condition: eq(variables.isPaired, 'false')
      inputs:
        azureSubscription: 'YourAzureServiceConnection'
        scriptType: 'bash'
        scriptLocation: 'inlineScript'
        inlineScript: |
          echo "Pairing Service Bus instances: $(serviceBusNamespaceA)
with $(serviceBusNamespaceB) using alias: $aliasName"
          az servicebus georecovery-alias set --alias $aliasName
--namespace-name $(serviceBusNamespaceA) --resource-group
$(resourceGroupA) --partner-namespace $(serviceBusNamespaceB)
          echo "Pairing initiated. Please verify in Azure Portal if
needed."
      name: InitiatePairing
      continueOnError: false # Stop if pairing fails

    - task: ManualValidation@0
      displayName: 'Manual Validation: Confirm Pairing and Readiness'
      timeoutInMinutes: 1440 # 24 hours to approve
      inputs:
        instructions: |
          Please verify that the Service Bus instances are correctly
paired (SB-A to SB-B) via the Azure Portal.
          Ensure that the geo-recovery configuration is in a healthy
state.
          Approve this step to proceed with the failover simulation.
        # Add email notifications if desired
        notifyUsers: |
          your_email@example.com
          another_email@example.com

- stage: FailoverSimulation
  displayName: 'Failover Operation (SB-A to SB-B)'
  dependsOn: PreFailoverChecks
  jobs:
  - job: PerformFailover
    displayName: 'Execute Failover from SB-A to SB-B'
    steps:
    # Step 3: Verify which region is instance is now active
    # This check is implicitly handled by the failover command itself.
    # After a successful failover, the *original* primary (SB-A)
becomes passive, and the *original* secondary (SB-B) becomes active.
    # You can add a check here *after* the failover to confirm the
role.
    - task: AzureCLI@2
      displayName: 'Initiate Failover from SB-A to SB-B'
      inputs:
        azureSubscription: 'YourAzureServiceConnection'
        scriptType: 'bash'
        scriptLocation: 'inlineScript'
        inlineScript: |
          echo "Initiating failover for alias: $aliasName from
$(serviceBusNamespaceA) to $(serviceBusNamespaceB)..."
          az servicebus georecovery-alias fail-over --alias $aliasName
--namespace-name $(serviceBusNamespaceA) --resource-group
$(resourceGroupA)
          echo "Failover command issued. It might take some time to
complete."
      name: InitiateFailover

    - task: ManualValidation@0
      displayName: 'Manual Validation: Confirm Failover to SB-B'
      timeoutInMinutes: 1440 # 24 hours
      inputs:
        instructions: |
          Please verify that the failover from SB-A to SB-B has
completed successfully.
          Confirm that SB-B is now the active namespace and SB-A is
passive.
          Check your applications are now connected to SB-B.
          Approve to proceed with queue deletion (if needed) on the
*passive* namespace.
        notifyUsers: |
          your_email@example.com

    # Step 4: After successful verification, another manual
determination task to delete the all the queues from the passive
```

```
instance
    # NOTE: You cannot delete queues from the active namespace. You
must delete from the passive one (SB-A after failover to SB-B).
    - task: AzureCLI@2
      displayName: 'Delete Queues from Passive SB-A (if applicable)'
      # This step should only run if you need to clear queues from the
*old* primary which is now passive.
      # Be extremely careful with this step as it's data destructive.
      inputs:
        azureSubscription: 'YourAzureServiceConnection'
        scriptType: 'bash'
        scriptLocation: 'inlineScript'
        inlineScript: |
          echo "Listing queues in passive namespace
$(serviceBusNamespaceA)..."
          queue_names=$(az servicebus queue list --namespace-name
$(serviceBusNamespaceA) --resource-group $(resourceGroupA) --query
'[].name' -o tsv)

          if [ -z "$queue_names" ]; then
            echo "No queues found in $(serviceBusNamespaceA) to
delete."
          else
            echo "Queues found: $queue_names"
            echo "Deleting queues from passive namespace
$(serviceBusNamespaceA)..."
            for queue_name in $queue_names; do
              echo "Deleting queue: $queue_name"
              az servicebus queue delete --name $queue_name
--namespace-name $(serviceBusNamespaceA) --resource-group
$(resourceGroupA)
            done
            echo "All queues deleted from $(serviceBusNamespaceA)."
          fi
      condition: |
        and(succeeded(), eq(variables['Agent.JobStatus'],
'Succeeded')) # Only run if previous steps succeeded.
      name: DeleteQueuesFromSBA

- stage: ReinitiatePairing
  displayName: 'Re-initiate Pairing and Second Failover Scenario'
  dependsOn: FailoverSimulation
  jobs:
  - job: RePairAndFailoverBack
    displayName: 'Re-pair and prepare for SB-B to SB-A failover'
    steps:
    # Step 5: Re-initiate the pairing again.
    - task: AzureCLI@2
```

```
      displayName: 'Break existing pairing (if any) and Re-initiate
pairing for SB-B to SB-A'
      inputs:
        azureSubscription: 'YourAzureServiceConnection'
        scriptType: 'bash'
        scriptLocation: 'inlineScript'
        inlineScript: |
          echo "Breaking current geo-recovery pairing for alias:
$aliasName..."
          # Note: There isn't a direct "break" command for
geo-recovery alias.
          # You typically delete and recreate if you want to change
the primary/secondary or re-initiate.
          # If the failover moved the primary, the original primary
(SB-A) is now passive.
          # To re-initiate pairing from SB-B to SB-A, you need to set
SB-B as primary.
          # If the current state is SB-B active, SB-A passive, you are
already set for failing back.
          # The "az servicebus georecovery-alias set" command
effectively re-establishes or updates the pairing.
          echo "Re-establishing pairing with SB-B as primary to SB-A
as secondary for failback..."
          az servicebus georecovery-alias set --alias $aliasName
--namespace-name $(serviceBusNamespaceB) --resource-group
$(resourceGroupB) --partner-namespace $(serviceBusNamespaceA)
          echo "Pairing re-initiated from SB-B to SB-A. Please
verify."
      name: ReInitiatePairingForFailback

    - task: ManualValidation@0
      displayName: 'Manual Validation: Confirm Re-pairing for
Failback'
      timeoutInMinutes: 1440
      inputs:
        instructions: |
          Please verify that the pairing has been re-established with
SB-B as the primary and SB-A as the secondary.
          This is crucial before attempting to fail back to SB-A.
          Approve to proceed with the failover back to SB-A.
        notifyUsers: |
          your_email@example.com

    # Step 6: Same way for failover between SB-B to SB-A scenario
    - task: AzureCLI@2
      displayName: 'Initiate Failover from SB-B to SB-A'
      inputs:
        azureSubscription: 'YourAzureServiceConnection'
        scriptType: 'bash'
        scriptLocation: 'inlineScript'
        inlineScript: |
          echo "Initiating failover for alias: $aliasName from
$(serviceBusNamespaceB) to $(serviceBusNamespaceA)..."
          az servicebus georecovery-alias fail-over --alias $aliasName
--namespace-name $(serviceBusNamespaceB) --resource-group
$(resourceGroupB)
          echo "Failover command issued. It might take some time to
complete."
      name: FailoverBackToSBA

    - task: ManualValidation@0
      displayName: 'Manual Validation: Confirm Failover to SB-A'
      timeoutInMinutes: 1440
      inputs:
        instructions: |
          Please verify that the failover from SB-B to SB-A has
completed successfully.
          Confirm that SB-A is now the active namespace and SB-B is
passive.
          Check your applications are now connected back to SB-A.
          Approve to finish the pipeline.
        notifyUsers: |
          your_email@example.com

    - task: AzureCLI@2
      displayName: 'Delete Queues from Passive SB-B (if applicable)'
      inputs:
        azureSubscription: 'YourAzureServiceConnection'
        scriptType: 'bash'
        scriptLocation: 'inlineScript'
        inlineScript: |
          echo "Listing queues in passive namespace
$(serviceBusNamespaceB)..."
          queue_names=$(az servicebus queue list --namespace-name
$(serviceBusNamespaceB) --resource-group $(resourceGroupB) --query
'[].name' -o tsv)

          if [ -z "$queue_names" ]; then
            echo "No queues found in $(serviceBusNamespaceB) to
delete."
          else
            echo "Queues found: $queue_names"
            echo "Deleting queues from passive namespace
$(serviceBusNamespaceB)..."
            for queue_name in $queue_names; do
              echo "Deleting queue: $queue_name"
              az servicebus queue delete --name $queue_name
--namespace-name $(serviceBusNamespaceB) --resource-group
$(resourceGroupB)
            done
            echo "All queues deleted from $(serviceBusNamespaceB)."
          fi
      condition: |
        and(succeeded(), eq(variables['Agent.JobStatus'],
'Succeeded'))
      name: DeleteQueuesFromSBB
```

**Explanation and Key Considerations:**

1. **Azure Service Connection:** Replace 'YourAzureServiceConnection' with the name of your Azure Resource Manager service connection configured in Azure DevOps. This connection needs appropriate permissions (e.g., "Azure Service Bus Data Owner" or a custom role with Microsoft.ServiceBus/namespaces/* permissions) to manage Service Bus.
2. **Variables:** Define your Service Bus namespace names, resource groups, and the geo-recovery alias as variables for easy management and reusability.
3. **AzureCLI@2 Task:**
   - This is the core task for executing Azure CLI commands.
   - scriptType: 'bash' is used for Linux agents, 'pscore' or 'powershell' for Windows agents.
   - scriptLocation: 'inlineScript' allows you to write your commands directly in the YAML.
4. **Step 1: Check Service Bus Regions Already Paired:**
   - The az servicebus georecovery-alias show command is used to get details about the alias, specifically checking the partnerNamespace to see if it matches SB-B.
   - echo "##vso[task.setvariable variable=isPaired]true" is a special command in Azure DevOps that sets a pipeline variable, which can then be used in subsequent conditions.
5. **Step 2: If Not Paired, Pair the 2 Instances:**
   - The condition: eq(variables.isPaired, 'false') ensures this step only runs if the isPaired variable (set in the previous step) is false.
   - az servicebus georecovery-alias set --alias $aliasName --namespace-name $(serviceBusNamespaceA) --resource-group $(resourceGroupA) --partner-namespace $(serviceBusNamespaceB) initiates the pairing.
6. **ManualValidation@0 Task (Manual Decision):**
   - This task pauses the pipeline execution until a specified user or group manually approves or rejects the step.
   - timeoutInMinutes: Crucial for defining how long the pipeline waits for approval. If it times out, the pipeline will typically fail.
   - instructions: Provides clear instructions to the approver.
   - notifyUsers: Sends email notifications to specified users for approval requests.
   - **Crucial for your requirement of "manual decision to trigger a flow."**
7. **Step 3: Verify Which Region is Active (Implicitly handled + Manual Validation):**
   - The failover command az servicebus georecovery-alias fail-over itself handles the role switching.

    - You use a ManualValidation step *after* the failover command to have an operator visually confirm the active/passive roles in the Azure Portal before proceeding.
8. **Step 4: Delete Queues from Passive Instance:**
    - This is a destructive operation. The script first lists queues and then iterates to delete them.
    - **Crucially, this must be done on the *passive* namespace.** After a failover from SB-A to SB-B, SB-A becomes passive.
    - The condition ensures this step only runs if the preceding steps were successful.
9. **Step 5 & 6: Re-initiate Pairing and Failover SB-B to SB-A:**
    - After the first failover (SB-A to SB-B), SB-B is now the primary. To fail back to SB-A, you need to re-establish the pairing with SB-B as the *current* primary.
    - The az servicebus georecovery-alias set command is used again, but this time with $(serviceBusNamespaceB) as the --namespace-name (the new primary) and $(serviceBusNamespaceA) as the --partner-namespace.
    - Another ManualValidation step is included to confirm the re-pairing before the second failover.
    - The failover command is executed again, this time from SB-B (current primary) to SB-A (new secondary).
    - A final ManualValidation ensures the failback to SB-A is successful.
    - Finally, the queues on the *new* passive namespace (SB-B) are deleted if required.

**Permissions:**
Ensure the Service Principal or Managed Identity associated with your Azure Service Connection has sufficient permissions to:
- Read Service Bus namespaces and geo-recovery aliases.
- Update Service Bus geo-recovery aliases (set and fail-over).
- Delete Service Bus queues.

**Testing and Best Practices:**
- **Test in a Non-Production Environment:** Always thoroughly test this pipeline in a non-production environment first.
- **Rollback Plan:** Have a clear rollback plan in case of issues during the failover.
- **Monitoring:** Implement robust monitoring for your Service Bus namespaces to quickly detect any anomalies during and after the failover.
- **Application Connection Strings:** Remember that your applications will need to use the geo-recovery alias connection string, not the direct namespace connection strings, to automatically connect to the active namespace.
- **Error Handling:** For production, you might want more sophisticated error handling in your bash scripts (e.g., checking exit codes of az commands).
- **Idempotency:** The Azure CLI commands are generally idempotent, meaning running them multiple times with the same parameters will have the same effect. This is good for pipeline retries.

This YAML pipeline provides a solid foundation for automating your Service Bus failover process, incorporating the necessary manual decision points as requested. Remember to customize the variables and service connection names to match your environment.