You are absolutely right! A production-ready solution isn't complete without the **CI/CD pipeline** to automate deployment. My apologies for omitting that crucial part in the previous comprehensive response.
Since you mentioned "automation described in my first question" and asked for "full code for production ready and reusable with any environment or subscriptions with all the automations," an Azure DevOps Pipeline is the perfect fit.
Here's an example of a **multi-stage Azure DevOps Pipeline (YAML)** that automates the deployment of your Terraform Azure AI Hub accelerator. This pipeline is designed for **reusability** across environments (dev, prod), uses a **service principal** for secure authentication, and follows **best practices** for Terraform in CI/CD.

Azure DevOps CI/CD Pipeline (YAML)

This YAML pipeline can be placed in your repository (e.g., .azuredevops/azure-ai-hub-pipeline.yml).

```
# .azuredevops/azure-ai-hub-pipeline.yml
# Azure CI/CD Pipeline for Azure AI Hub Accelerator (Terraform)

trigger:
  branches:
    include:
      - main
  paths:
    include:
      - terraform/*
      - .azuredevops/*

pr:
  branches:
    include:
      - main
  paths:
    include:
      - terraform/*
      - .azuredevops/*

variables:
  # General Terraform settings
  - group: TF_GLOBAL_VARS # Link to an Azure DevOps Variable Group for
global variables
  - name: terraformWorkingDirectory
    value: '$(Build.SourcesDirectory)/terraform' # Root of your
Terraform code

  # Azure service connection name (created in Azure DevOps -> Project
Settings -> Service connections)
  # This service connection should use a Service Principal with
```

```
appropriate permissions on your Azure Subscription.
  - name: azureServiceConnection
    value: 'AzureServiceConnection-YourSubscription' # **UPDATE THIS**

stages:
  - stage: BuildAndValidate
    displayName: 'Build and Validate Terraform'
    jobs:
      - job: TerraformValidation
        displayName: 'Terraform Code Validation'
        pool:
          vmImage: 'ubuntu-latest' # Use a Linux agent for Terraform

        steps:
          - checkout: self
            displayName: 'Checkout Code'

          - task: PowerShell@2
            displayName: 'Zip Python Function App Code'
            inputs:
              targetType: 'inline'
              script: |
                $sourceDir =
"$(terraformWorkingDirectory)/modules/function_app_cost_processor/src"
                $zipFile =
"$(terraformWorkingDirectory)/modules/function_app_cost_processor/src/
function_app.zip"

                # Check if the source directory exists
                if (-not (Test-Path $sourceDir)) {
                  Write-Error "Source directory for Function App code
not found: $sourceDir"
                  exit 1
                }

                # Create the zip file
                Compress-Archive -Path "$($sourceDir)\*"
-DestinationPath $zipFile -Force
                Write-Host "##vso[task.setvariable
variable=functionAppZipPath]$zipFile"
            env:
              TF_WORKING_DIRECTORY: $(terraformWorkingDirectory) #
Pass to ensure correct paths

          - task: TerraformInstaller@1
            displayName: 'Install Terraform'
            inputs:
              terraformVersion: 'latest' # Or a specific version like
```

```
'1.8.x'

          - task: AzureCLI@2
            displayName: 'Azure Login for Terraform Init'
            inputs:
              azureSubscription: $(azureServiceConnection)
              scriptType: 'bash'
              scriptLocation: 'inlineScript'
              inlineScript: |
                az account show
            env:
              ARM_SUBSCRIPTION_ID:
$(TF_GLOBAL_VARS.ARM_SUBSCRIPTION_ID) # From Variable Group
              ARM_CLIENT_ID: $(TF_GLOBAL_VARS.ARM_CLIENT_ID) # From
Variable Group
              ARM_CLIENT_SECRET: $(TF_GLOBAL_VARS.ARM_CLIENT_SECRET) #
From Variable Group
              ARM_TENANT_ID: $(TF_GLOBAL_VARS.ARM_TENANT_ID) # From
Variable Group

          - task: PowerShell@2 # Need PowerShell for the backend.tf
file creation due to syntax
            displayName: 'Create Terraform Backend Config for Init'
            inputs:
              targetType: 'inline'
              script: |
                $backendConfigPath =
"$(terraformWorkingDirectory)/environments/$(environmentName)/backend.
tfvars"
                $content = @"
                resource_group_name  =
"$(TF_GLOBAL_VARS.TFSTATE_RG_NAME)"
                storage_account_name =
"$(TF_GLOBAL_VARS.TFSTATE_SA_NAME_$(environmentName))"
                container_name       = "tfstate"
                key                  =
"$(environmentName).aihub.terraform.tfstate"
                "@
                $content | Out-File -FilePath $backendConfigPath
-Encoding utf8
                Write-Host "Created backend config:
$backendConfigPath"
            env:
              environmentName: dev # Use 'dev' for init in validation,
as it needs *some* backend
              TF_GLOBAL_VARS_TFSTATE_RG_NAME:
$(TF_GLOBAL_VARS.TFSTATE_RG_NAME)
              TF_GLOBAL_VARS_TFSTATE_SA_NAME_dev:
```

```
$(TF_GLOBAL_VARS.TFSTATE_SA_NAME_dev)
            workingDirectory:
$(terraformWorkingDirectory)/environments/dev # Point to an
environment for backend config

          - task: TerraformTaskV4@4 # For Terraform specific commands
            displayName: 'Terraform Init (for validation)'
            inputs:
              provider: 'azurerm'
              command: 'init'
              workingDirectory:
'$(terraformWorkingDirectory)/environments/dev' # Init 'dev' for
validation
              backendServiceARM: $(azureServiceConnection)
              backendConfiguration: |
                resource_group_name=$(TF_GLOBAL_VARS.TFSTATE_RG_NAME)

storage_account_name=$(TF_GLOBAL_VARS.TFSTATE_SA_NAME_dev)
                container_name=tfstate
                key=dev.aihub.terraform.tfstate
              # Note: backendConfig: 'path/to/backend.tfvars' can be
used instead of inline config

          - task: TerraformTaskV4@4
            displayName: 'Terraform Format Check'
            inputs:
              provider: 'azurerm'
              command: 'fmt'
              workingDirectory: '$(terraformWorkingDirectory)' # Root
of all Terraform modules
              commandOptions: '-check -recursive'

          - task: TerraformTaskV4@4
            displayName: 'Terraform Validate'
            inputs:
              provider: 'azurerm'
              command: 'validate'
              workingDirectory:
'$(terraformWorkingDirectory)/environments/dev' # Validate the 'dev'
environment
              commandOptions: '-json' # Output in JSON for potential
parsing/reporting

          - publish:
$(terraformWorkingDirectory)/modules/function_app_cost_processor/src/f
unction_app.zip
            artifact: functionAppZip
            displayName: 'Publish Function App Zip Artifact'
```

```
  - stage: DeployDev
    displayName: 'Deploy to Development'
    dependsOn: BuildAndValidate
    condition: succeeded()
    variables:
      - name: environmentName
        value: 'dev'
      - name: tfvarsFile
        value: 'dev.tfvars' # For plan/apply
      - group: TF_DEV_VARS # Link to an Azure DevOps Variable Group
for dev variables
    jobs:
      - deployment: DevDeployment
        displayName: 'Dev Environment Deployment'
        environment: 'dev' # Link to an Azure DevOps Environment
        pool:
          vmImage: 'ubuntu-latest'

        strategy:
          runOnce:
            preDeployHook:
              steps:
                - checkout: self
                  displayName: 'Checkout Code'

                - download: current
                  artifact: functionAppZip
                  displayName: 'Download Function App Zip'
                  path:
$(terraformWorkingDirectory)/modules/function_app_cost_processor/src/

                - task: TerraformInstaller@1
                  displayName: 'Install Terraform'
                  inputs:
                    terraformVersion: 'latest'

                - task: AzureCLI@2
                  displayName: 'Azure Login for Terraform'
                  inputs:
                    azureSubscription: $(azureServiceConnection)
                    scriptType: 'bash'
                    scriptLocation: 'inlineScript'
                    inlineScript: |
                      az account show
                  env:
                    ARM_SUBSCRIPTION_ID:
$(TF_GLOBAL_VARS.ARM_SUBSCRIPTION_ID)
```

```
                    ARM_CLIENT_ID: $(TF_GLOBAL_VARS.ARM_CLIENT_ID)
                    ARM_CLIENT_SECRET:
$(TF_GLOBAL_VARS.ARM_CLIENT_SECRET)
                    ARM_TENANT_ID: $(TF_GLOBAL_VARS.ARM_TENANT_ID)

                - task: PowerShell@2
                  displayName: 'Create Terraform Backend Config'
                  inputs:
                    targetType: 'inline'
                    script: |
                      $backendConfigPath =
"$(terraformWorkingDirectory)/environments/$(environmentName)/backend.
tfvars"
                      $content = @"
                      resource_group_name  =
"$(TF_GLOBAL_VARS.TFSTATE_RG_NAME)"
                      storage_account_name =
"$(TF_GLOBAL_VARS.TFSTATE_SA_NAME_$(environmentName))"
                      container_name       = "tfstate"
                      key                  =
"$(environmentName).aihub.terraform.tfstate"
                      "@
                      $content | Out-File -FilePath $backendConfigPath
-Encoding utf8
                      Write-Host "Created backend config:
$backendConfigPath"
                  env:
                    environmentName: $(environmentName)
                    TF_GLOBAL_VARS_TFSTATE_RG_NAME:
$(TF_GLOBAL_VARS.TFSTATE_RG_NAME)
                    TF_GLOBAL_VARS_TFSTATE_SA_NAME_dev:
$(TF_GLOBAL_VARS.TFSTATE_SA_NAME_dev)
                  workingDirectory:
$(terraformWorkingDirectory)/environments/$(environmentName)

                - task: TerraformTaskV4@4
                  displayName: 'Terraform Init (Dev)'
                  inputs:
                    provider: 'azurerm'
                    command: 'init'
                    workingDirectory:
'$(terraformWorkingDirectory)/environments/$(environmentName)'
                    backendServiceARM: $(azureServiceConnection)
                    backendConfiguration: |

resource_group_name=$(TF_GLOBAL_VARS.TFSTATE_RG_NAME)

storage_account_name=$(TF_GLOBAL_VARS.TFSTATE_SA_NAME_$(environmentNam
```

```
e))
                      container_name=tfstate
                      key=$(environmentName).aihub.terraform.tfstate

            deployHook:
              steps:
                - task: TerraformTaskV4@4
                  displayName: 'Terraform Plan (Dev)'
                  inputs:
                    provider: 'azurerm'
                    command: 'plan'
                    workingDirectory:
'$(terraformWorkingDirectory)/environments/$(environmentName)'
                    commandOptions:
'-var-file=$(environmentName).tfvars
-out=$(Build.ArtifactStagingDirectory)/$(environmentName).tfplan'
                    environmentServiceNameARM:
$(azureServiceConnection)
                  env:
                    # Pass variables from TF_DEV_VARS variable group
                    ARM_SUBSCRIPTION_ID:
$(TF_GLOBAL_VARS.ARM_SUBSCRIPTION_ID)
                    TF_VAR_location: $(TF_DEV_VARS.LOCATION)
                    TF_VAR_environment: $(environmentName)
                    TF_VAR_resource_group_prefix:
$(TF_DEV_VARS.RESOURCE_GROUP_PREFIX)
                    TF_VAR_existing_vnet_name:
$(TF_DEV_VARS.EXISTING_VNET_NAME)
                    TF_VAR_existing_vnet_resource_group_name:
$(TF_DEV_VARS.EXISTING_VNET_RESOURCE_GROUP_NAME)
                    TF_VAR_existing_subnets:
$(TF_DEV_VARS.EXISTING_SUBNETS)
                    TF_VAR_ai_services_sku:
$(TF_DEV_VARS.AI_SERVICES_SKU)
                    TF_VAR_openai_model_deployments_dev:
$(TF_DEV_VARS.OPENAI_MODEL_DEPLOYMENTS_DEV)
                    TF_VAR_openai_api_version:
$(TF_DEV_VARS.OPENAI_API_VERSION)
                    TF_VAR_apim_sku_name: $(TF_DEV_VARS.APIM_SKU_NAME)
                    TF_VAR_apim_products: $(TF_DEV_VARS.APIM_PRODUCTS)
                    TF_VAR_cost_management_event_hub_namespace_name:
$(TF_DEV_VARS.COST_MANAGEMENT_EVENT_HUB_NAMESPACE_NAME)
                    TF_VAR_cost_management_event_hub_name:
$(TF_DEV_VARS.COST_MANAGEMENT_EVENT_HUB_NAME)
                    TF_VAR_cost_management_consumption_api_scope:
$(TF_DEV_VARS.COST_MANAGEMENT_CONSUMPTION_API_SCOPE)
                    TF_VAR_function_app_name:
$(TF_DEV_VARS.FUNCTION_APP_NAME)
```

```
                    TF_VAR_function_app_storage_sku:
$(TF_DEV_VARS.FUNCTION_APP_STORAGE_SKU)
                    TF_VAR_function_app_python_version:
$(TF_DEV_VARS.FUNCTION_APP_PYTHON_VERSION)
                    TF_VAR_key_vault_sku_name:
$(TF_DEV_VARS.KEY_VAULT_SKU_NAME)
                    TF_VAR_openai_api_key:
$(TF_DEV_VARS.OPENAI_API_KEY) # **IMPORTANT**: Pass securely!
                    TF_VAR_application_insights_connection_string:
$(TF_GLOBAL_VARS.APPLICATION_INSIGHTS_CONNECTION_STRING) # Example for
general, use appropriate variable group
                    TF_VAR_application_insights_instrumentation_key:
$(TF_GLOBAL_VARS.APPLICATION_INSIGHTS_INSTRUMENTATION_KEY) # Example
for general, use appropriate variable group

                - publish:
$(Build.ArtifactStagingDirectory)/$(environmentName).tfplan
                  artifact: $(environmentName)TerraformPlan
                  displayName: 'Publish Dev Plan Artifact'

            postDeployHook:
              steps:
                - task: TerraformTaskV4@4
                  displayName: 'Terraform Apply (Dev)'
                  inputs:
                    provider: 'azurerm'
                    command: 'apply'
                    workingDirectory:
'$(terraformWorkingDirectory)/environments/$(environmentName)'
                    commandOptions:
'$(Build.ArtifactStagingDirectory)/$(environmentName).tfplan'
                    environmentServiceNameARM:
$(azureServiceConnection)
                  env:
                    # Pass variables again for apply
                    ARM_SUBSCRIPTION_ID:
$(TF_GLOBAL_VARS.ARM_SUBSCRIPTION_ID)
                    TF_VAR_location: $(TF_DEV_VARS.LOCATION)
                    TF_VAR_environment: $(environmentName)
                    TF_VAR_resource_group_prefix:
$(TF_DEV_VARS.RESOURCE_GROUP_PREFIX)
                    TF_VAR_existing_vnet_name:
$(TF_DEV_VARS.EXISTING_VNET_NAME)
                    TF_VAR_existing_vnet_resource_group_name:
$(TF_DEV_VARS.EXISTING_VNET_RESOURCE_GROUP_NAME)
                    TF_VAR_existing_subnets:
$(TF_DEV_VARS.EXISTING_SUBNETS)
                    TF_VAR_ai_services_sku:
```

```
$(TF_DEV_VARS.AI_SERVICES_SKU)
                    TF_VAR_openai_model_deployments_dev:
$(TF_DEV_VARS.OPENAI_MODEL_DEPLOYMENTS_DEV)
                    TF_VAR_openai_api_version:
$(TF_DEV_VARS.OPENAI_API_VERSION)
                    TF_VAR_apim_sku_name: $(TF_DEV_VARS.APIM_SKU_NAME)
                    TF_VAR_apim_products: $(TF_DEV_VARS.APIM_PRODUCTS)
                    TF_VAR_cost_management_event_hub_namespace_name:
$(TF_DEV_VARS.COST_MANAGEMENT_EVENT_HUB_NAMESPACE_NAME)
                    TF_VAR_cost_management_event_hub_name:
$(TF_DEV_VARS.COST_MANAGEMENT_EVENT_HUB_NAME)
                    TF_VAR_cost_management_consumption_api_scope:
$(TF_DEV_VARS.COST_MANAGEMENT_CONSUMPTION_API_SCOPE)
                    TF_VAR_function_app_name:
$(TF_DEV_VARS.FUNCTION_APP_NAME)
                    TF_VAR_function_app_storage_sku:
$(TF_DEV_VARS.FUNCTION_APP_STORAGE_SKU)
                    TF_VAR_function_app_python_version:
$(TF_DEV_VARS.FUNCTION_APP_PYTHON_VERSION)
                    TF_VAR_key_vault_sku_name:
$(TF_DEV_VARS.KEY_VAULT_SKU_NAME)
                    TF_VAR_openai_api_key:
$(TF_DEV_VARS.OPENAI_API_KEY) # **IMPORTANT**: Pass securely!
                    TF_VAR_application_insights_connection_string:
$(TF_GLOBAL_VARS.APPLICATION_INSIGHTS_CONNECTION_STRING)
                    TF_VAR_application_insights_instrumentation_key:
$(TF_GLOBAL_VARS.APPLICATION_INSIGHTS_INSTRUMENTATION_KEY)

  - stage: DeployProd
    displayName: 'Deploy to Production'
    dependsOn: DeployDev
    condition: succeeded()
    variables:
      - name: environmentName
        value: 'prod'
      - name: tfvarsFile
        value: 'prod.tfvars' # For plan/apply
      - group: TF_PROD_VARS # Link to an Azure DevOps Variable Group
for prod variables
    jobs:
      - deployment: ProdDeployment
        displayName: 'Prod Environment Deployment'
        environment: 'prod' # Link to an Azure DevOps Environment
        pool:
          vmImage: 'ubuntu-latest'

        strategy:
          runOnce:
```

```
            preDeployHook:
              steps:
                - checkout: self
                  displayName: 'Checkout Code'

                - download: current
                  artifact: functionAppZip
                  displayName: 'Download Function App Zip'
                  path:
$(terraformWorkingDirectory)/modules/function_app_cost_processor/src/

                - task: TerraformInstaller@1
                  displayName: 'Install Terraform'
                  inputs:
                    terraformVersion: 'latest'

                - task: AzureCLI@2
                  displayName: 'Azure Login for Terraform'
                  inputs:
                    azureSubscription: $(azureServiceConnection)
                    scriptType: 'bash'
                    scriptLocation: 'inlineScript'
                    inlineScript: |
                      az account show
                  env:
                    ARM_SUBSCRIPTION_ID:
$(TF_GLOBAL_VARS.ARM_SUBSCRIPTION_ID)
                    ARM_CLIENT_ID: $(TF_GLOBAL_VARS.ARM_CLIENT_ID)
                    ARM_CLIENT_SECRET:
$(TF_GLOBAL_VARS.ARM_CLIENT_SECRET)
                    ARM_TENANT_ID: $(TF_GLOBAL_VARS.ARM_TENANT_ID)

                - task: PowerShell@2
                  displayName: 'Create Terraform Backend Config'
                  inputs:
                    targetType: 'inline'
                    script: |
                      $backendConfigPath =
"$(terraformWorkingDirectory)/environments/$(environmentName)/backend.
tfvars"
                      $content = @"
                      resource_group_name  =
"$(TF_GLOBAL_VARS.TFSTATE_RG_NAME)"
                      storage_account_name =
"$(TF_GLOBAL_VARS.TFSTATE_SA_NAME_$(environmentName))"
                      container_name       = "tfstate"
                      key                  =
"$(environmentName).aihub.terraform.tfstate"
```

```
                    "@
                    $content | Out-File -FilePath $backendConfigPath
-Encoding utf8
                    Write-Host "Created backend config:
$backendConfigPath"
                env:
                  environmentName: $(environmentName)
                  TF_GLOBAL_VARS_TFSTATE_RG_NAME:
$(TF_GLOBAL_VARS.TFSTATE_RG_NAME)
                  TF_GLOBAL_VARS_TFSTATE_SA_NAME_prod:
$(TF_GLOBAL_VARS.TFSTATE_SA_NAME_prod)
                workingDirectory:
$(terraformWorkingDirectory)/environments/$(environmentName)

              - task: TerraformTaskV4@4
                displayName: 'Terraform Init (Prod)'
                inputs:
                  provider: 'azurerm'
                  command: 'init'
                  workingDirectory:
'$(terraformWorkingDirectory)/environments/$(environmentName)'
                  backendServiceARM: $(azureServiceConnection)
                  backendConfiguration: |

resource_group_name=$(TF_GLOBAL_VARS.TFSTATE_RG_NAME)

storage_account_name=$(TF_GLOBAL_VARS.TFSTATE_SA_NAME_$(environmentNam
e))
                    container_name=tfstate
                    key=$(environmentName).aihub.terraform.tfstate

          deployHook:
            steps:
              - task: TerraformTaskV4@4
                displayName: 'Terraform Plan (Prod)'
                inputs:
                  provider: 'azurerm'
                  command: 'plan'
                  workingDirectory:
'$(terraformWorkingDirectory)/environments/$(environmentName)'
                  commandOptions:
'-var-file=$(environmentName).tfvars
-out=$(Build.ArtifactStagingDirectory)/$(environmentName).tfplan'
                  environmentServiceNameARM:
$(azureServiceConnection)
                env:
                  # Pass variables from TF_PROD_VARS variable group
                  ARM_SUBSCRIPTION_ID:
```

```
$(TF_GLOBAL_VARS.ARM_SUBSCRIPTION_ID)
                    TF_VAR_location: $(TF_PROD_VARS.LOCATION)
                    TF_VAR_environment: $(environmentName)
                    TF_VAR_resource_group_prefix:
$(TF_PROD_VARS.RESOURCE_GROUP_PREFIX)
                    TF_VAR_existing_vnet_name:
$(TF_PROD_VARS.EXISTING_VNET_NAME)
                    TF_VAR_existing_vnet_resource_group_name:
$(TF_PROD_VARS.EXISTING_VNET_RESOURCE_GROUP_NAME)
                    TF_VAR_existing_subnets:
$(TF_PROD_VARS.EXISTING_SUBNETS)
                    TF_VAR_ai_services_sku:
$(TF_PROD_VARS.AI_SERVICES_SKU)
                    TF_VAR_openai_model_deployments_dev:
$(TF_PROD_VARS.OPENAI_MODEL_DEPLOYMENTS_PROD) # Note: _PROD for
production
                    TF_VAR_openai_api_version:
$(TF_PROD_VARS.OPENAI_API_VERSION)
                    TF_VAR_apim_sku_name:
$(TF_PROD_VARS.APIM_SKU_NAME)
                    TF_VAR_apim_products:
$(TF_PROD_VARS.APIM_PRODUCTS)
                    TF_VAR_cost_management_event_hub_namespace_name:
$(TF_PROD_VARS.COST_MANAGEMENT_EVENT_HUB_NAMESPACE_NAME)
                    TF_VAR_cost_management_event_hub_name:
$(TF_PROD_VARS.COST_MANAGEMENT_EVENT_HUB_NAME)
                    TF_VAR_cost_management_consumption_api_scope:
$(TF_PROD_VARS.COST_MANAGEMENT_CONSUMPTION_API_SCOPE)
                    TF_VAR_function_app_name:
$(TF_PROD_VARS.FUNCTION_APP_NAME)
                    TF_VAR_function_app_storage_sku:
$(TF_PROD_VARS.FUNCTION_APP_STORAGE_SKU)
                    TF_VAR_function_app_python_version:
$(TF_PROD_VARS.FUNCTION_APP_PYTHON_VERSION)
                    TF_VAR_key_vault_sku_name:
$(TF_PROD_VARS.KEY_VAULT_SKU_NAME)
                    TF_VAR_openai_api_key:
$(TF_PROD_VARS.OPENAI_API_KEY) # **IMPORTANT**: Pass securely!
                    TF_VAR_application_insights_connection_string:
$(TF_GLOBAL_VARS.APPLICATION_INSIGHTS_CONNECTION_STRING)
                    TF_VAR_application_insights_instrumentation_key:
$(TF_GLOBAL_VARS.APPLICATION_INSIGHTS_INSTRUMENTATION_KEY)

                - publish:
$(Build.ArtifactStagingDirectory)/$(environmentName).tfplan
                  artifact: $(environmentName)TerraformPlan
                  displayName: 'Publish Prod Plan Artifact'
```

```
            postDeployHook:
              steps:
                - task: ManualValidation@0
                  displayName: 'Approve Production Deployment'
                  inputs:
                    instructions: |
                      Review the production Terraform plan and approve
or reject the deployment.
                      Plan artifact:
$(Build.ArtifactStagingDirectory)/$(environmentName).tfplan
                    notifyUsers: |
                      your_team_email@yourcompany.com # **UPDATE
THIS**
                  # Recommended to set timeout in days
                  timeoutInMinutes: 1440 # 24 hours

                - task: TerraformTaskV4@4
                  displayName: 'Terraform Apply (Prod)'
                  inputs:
                    provider: 'azurerm'
                    command: 'apply'
                    workingDirectory:
'$(terraformWorkingDirectory)/environments/$(environmentName)'
                    commandOptions:
'$(Build.ArtifactStagingDirectory)/$(environmentName).tfplan'
                    environmentServiceNameARM:
$(azureServiceConnection)
                  env:
                    # Pass variables again for apply
                    ARM_SUBSCRIPTION_ID:
$(TF_GLOBAL_VARS.ARM_SUBSCRIPTION_ID)
                    TF_VAR_location: $(TF_PROD_VARS.LOCATION)
                    TF_VAR_environment: $(environmentName)
                    TF_VAR_resource_group_prefix:
$(TF_PROD_VARS.RESOURCE_GROUP_PREFIX)
                    TF_VAR_existing_vnet_name:
$(TF_PROD_VARS.EXISTING_VNET_NAME)
                    TF_VAR_existing_vnet_resource_group_name:
$(TF_PROD_VARS.EXISTING_VNET_RESOURCE_GROUP_NAME)
                    TF_VAR_existing_subnets:
$(TF_PROD_VARS.EXISTING_SUBNETS)
                    TF_VAR_ai_services_sku:
$(TF_PROD_VARS.AI_SERVICES_SKU)
                    TF_VAR_openai_model_deployments_dev:
$(TF_PROD_VARS.OPENAI_MODEL_DEPLOYMENTS_PROD)
                    TF_VAR_openai_api_version:
$(TF_PROD_VARS.OPENAI_API_VERSION)
                    TF_VAR_apim_sku_name:
```

```
$(TF_PROD_VARS.APIM_SKU_NAME)
                    TF_VAR_apim_products:
$(TF_PROD_VARS.APIM_PRODUCTS)
                    TF_VAR_cost_management_event_hub_namespace_name:
$(TF_PROD_VARS.COST_MANAGEMENT_EVENT_HUB_NAMESPACE_NAME)
                    TF_VAR_cost_management_event_hub_name:
$(TF_PROD_VARS.COST_MANAGEMENT_EVENT_HUB_NAME)
                    TF_VAR_cost_management_consumption_api_scope:
$(TF_PROD_VARS.COST_MANAGEMENT_CONSUMPTION_API_SCOPE)
                    TF_VAR_function_app_name:
$(TF_PROD_VARS.FUNCTION_APP_NAME)
                    TF_VAR_function_app_storage_sku:
$(TF_PROD_VARS.FUNCTION_APP_STORAGE_SKU)
                    TF_VAR_function_app_python_version:
$(TF_PROD_VARS.FUNCTION_APP_PYTHON_VERSION)
                    TF_VAR_key_vault_sku_name:
$(TF_PROD_VARS.KEY_VAULT_SKU_NAME)
                    TF_VAR_openai_api_key:
$(TF_PROD_VARS.OPENAI_API_KEY) # **IMPORTANT**: Pass securely!
                    TF_VAR_application_insights_connection_string:
$(TF_GLOBAL_VARS.APPLICATION_INSIGHTS_CONNECTION_STRING)
                    TF_VAR_application_insights_instrumentation_key:
$(TF_GLOBAL_VARS.APPLICATION_INSIGHTS_INSTRUMENTATION_KEY)
```

Setting Up Azure DevOps for Terraform

Before you can run this pipeline, you need to configure a few things in your Azure DevOps organization/project:

1. Service Connection to Azure

This is how Azure DevOps authenticates to your Azure Subscription.

- Go to **Project settings** > **Service connections**.
- Create a **New service connection**.
- Choose **Azure Resource Manager**.
- Select **Service principal (automatic)** or **Service principal (manual)**.
    - **Automatic:** Easier, Azure DevOps creates the SP for you.
    - **Manual:** You create the Service Principal in Azure CLI/Portal, assign permissions, then paste details.
- **Scope Level:** Choose **Subscription**.
- Select your **Subscription** and provide a **Service connection name** (e.g., AzureServiceConnection-YourSubscription). **This name must match the azureServiceConnection variable in the YAML pipeline.**
- **Permissions:** Ensure the Service Principal used for this connection has:
    - **Contributor** role (or more granular roles for production) on the target Azure Subscription.

    - **Storage Blob Data Contributor** on the Terraform state storage account.
    - If using azurerm_consumption_budget, it needs **Cost Management Contributor** on the subscription.

2. Azure DevOps Environments (for Approval Gates)

Environments in Azure DevOps provide visualization of deployments and allow for approval gates.

- Go to **Pipelines** > **Environments**.
- Create new environments named dev and prod (matching the environment in the YAML).
- For the prod environment, add an **Approval and checks** to require manual approval before deployment. This is crucial for production safety.

3. Azure DevOps Variable Groups

Variable groups allow you to centrally manage variables and secrets, and share them across pipelines. This is where you'll store sensitive data and environment-specific configurations.

- Go to **Pipelines** > **Library**.
- Create a **New variable group**.
    - **TF_GLOBAL_VARS**: For variables common across all environments or highly sensitive, often related to Service Principal authentication.
        - ARM_SUBSCRIPTION_ID: Your Azure Subscription ID
        - ARM_CLIENT_ID: Service Principal Application ID
        - ARM_CLIENT_SECRET: Service Principal Secret (mark as secret!)
        - ARM_TENANT_ID: Your Azure AD Tenant ID
        - TFSTATE_RG_NAME: Resource Group name for your Terraform state storage accounts (e.g., rg-tfstate)
        - TFSTATE_SA_NAME_dev: Storage Account name for dev Terraform state (e.g., tfstatedevaihub)
        - TFSTATE_SA_NAME_prod: Storage Account name for prod Terraform state (e.g., tfstateprodaihub)
        - APPLICATION_INSIGHTS_CONNECTION_STRING: (e.g., from a central App Insights if shared)
        - APPLICATION_INSIGHTS_INSTRUMENTATION_KEY: (e.g., from a central App Insights if shared)
    - **TF_DEV_VARS**: For dev-specific variables.
        - LOCATION: uksouth
        - RESOURCE_GROUP_PREFIX: rg-dev-aihub
        - EXISTING_VNET_NAME: vnet-shared-dev
        - EXISTING_VNET_RESOURCE_GROUP_NAME: rg-network-dev
        - EXISTING_SUBNETS: {"apim_subnet": "snet-apim-dev", ...} (Store this as a JSON string for Terraform map input)
        - AI_SERVICES_SKU: S0 or F0
        - OPENAI_MODEL_DEPLOYMENTS_DEV: {"gpt-35-turbo-dev": {"model_name": "gpt-35-turbo", "model_version": "1106", "capacity": 1}} (as JSON string)
        - OPENAI_API_VERSION: 2024-02-01
        - APIM_SKU_NAME: Developer_1

        - APIM_PRODUCTS: {"open-ai-models": {"display_name": "OpenAI Models", ...}} (as JSON string)
        - COST_MANAGEMENT_EVENT_HUB_NAMESPACE_NAME: evhns-cost-dev
        - COST_MANAGEMENT_EVENT_HUB_NAME: costdata
        - COST_MANAGEMENT_CONSUMPTION_API_SCOPE: /subscriptions/YOUR_DEV_SUB_ID
        - FUNCTION_APP_NAME: func-costprocessor-dev
        - FUNCTION_APP_STORAGE_SKU: Standard_LRS
        - FUNCTION_APP_PYTHON_VERSION: 3.9
        - KEY_VAULT_SKU_NAME: Standard
        - OPENAI_API_KEY: **The actual OpenAI API Key (mark as secret!).** *This is a sensitive value and should be stored securely in Key Vault after the first deployment and read from there in subsequent runs for the Function App.*
    - **TF_PROD_VARS**: Repeat for production, with appropriate prod values (e.g., location, rg-prod-aihub, Premium_1 for APIM SKU, S0 for AI Services, gpt-4o-prod deployments, YOUR_PROD_SUB_ID, OPENAI_API_KEY for prod).
- **Link variable groups:** In the pipeline, link these variable groups to your stages/jobs.

4. Initial Terraform State Storage Account Setup (Manual)

You need to manually create a dedicated Azure Storage Account and container *once* for each environment to store your Terraform state. This is a crucial bootstrap step.

- **Resource Group:** rg-tfstate (from TF_GLOBAL_VARS.TFSTATE_RG_NAME)
- **Storage Account (for Dev):** tfstatedevaihub (from TF_GLOBAL_VARS.TFSTATE_SA_NAME_dev), ensure **Hierarchical namespace** is **disabled**.
    - Container: tfstate
- **Storage Account (for Prod):** tfstateprodaihub (from TF_GLOBAL_VARS.TFSTATE_SA_NAME_prod), ensure **Hierarchical namespace** is **disabled**.
    - Container: tfstate

Running the Pipeline

1. **Commit your Terraform code and the YAML pipeline** to your Git repository.
2. Go to **Pipelines** in Azure DevOps, click **New pipeline**.
3. Select **Azure Repos Git** (or your repo type).
4. Choose your repository.
5. Select **Existing Azure Pipelines YAML file** and point to .azuredevops/azure-ai-hub-pipeline.yml.
6. **Run** the pipeline.

The pipeline will:

1. **Build and Validate:** Perform terraform fmt -check, terraform validate, and zip your Function App code.
2. **Deploy Dev:**
    - Initialize Terraform with the dev backend.

    - Run terraform plan for the dev environment, saving the plan.
    - Run terraform apply for the dev environment using the saved plan.
3. **Deploy Prod:**
    - Initialize Terraform with the prod backend.
    - Run terraform plan for the prod environment, saving the plan.
    - **Pause for Manual Approval:** The pipeline will wait for an authorized user to manually review the plan in the Azure DevOps portal.
    - Run terraform apply for the prod environment using the saved plan.

This comprehensive pipeline structure ensures your Azure AI Hub accelerator deployments are consistent, secure, and automated, providing a true production-ready solution.
Do you have any specific questions about the pipeline steps, variables, or security considerations?