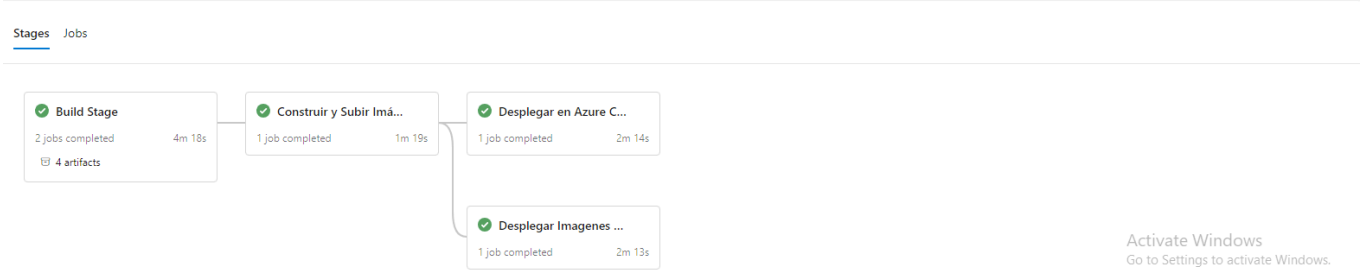
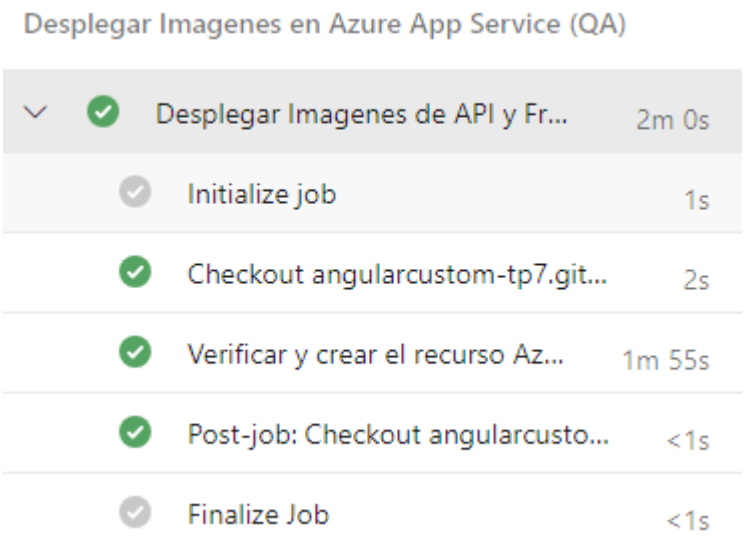
4.1 Modificar nuestro pipeline para incluir el deploy en QA y PROD de Imagenes Docker en Servicio Azure App Services con Soporte para Contenedores

4.1.1 - Agregar a nuestro pipeline una nueva etapa que dependa de nuestra etapa de Construcción y Pruebas y de la etapa de Construcción de Imagenes Docker y subida a ACR realizada en el TP08

```yaml
#---------------------------------------
### STAGE DEPLOY TO AZURE APP SERVICE QA
#---------------------------------------
- stage: DeployImagesToAppServiceQA
  displayName: 'Desplegar Imagenes en Azure App Service (QA)'
  dependsOn:
    - BuildAndTestBackAndFront
    - DockerBuildAndPush
  condition: succeeded()
  jobs:
    - job: DeployImagesToAppServiceQA
      displayName: 'Desplegar Imagenes de API y Front en Azure App Service (QA)'
      pool:
        vmImage: 'ubuntu-latest'
      steps:
        #-------------------------------------------------------
        # DEPLOY DOCKER API IMAGE TO AZURE APP SERVICE (QA)
        #-------------------------------------------------------
        Settings
        - task: AzureCLI@2
          displayName: 'Verificar y crear el recurso Azure App Service para API (QA) si no existe'
          inputs:
            azureSubscription: '$(ConnectedServiceName)'
            scriptType: 'bash'
            scriptLocation: 'inlineScript'
            inlineScript: |
              # Verificar si el App Service para la API ya existe
              if ! az webapp list --query "[?name=='$(WebAppApiNameContainersQA)' && resourceGroup=='$(ResourceGroupName)']" | length(@)" -o tsv | grep -q
                echo "El App Service para API QA no existe. Creando..."
                # Crear el App Service sin especificar la imagen del contenedor
                az webapp create --resource-group $(ResourceGroupName) --plan $(AppServicePlanLinux) --name $(WebAppApiNameContainersQA) --deployment-con
              else
                echo "El App Service para API QA ya existe. Actualizando la imagen..."
              fi

              # Configurar el App Service para usar Azure Container Registry (ACR)
              az webapp config container set --name $(WebAppApiNameContainersQA) --resource-group $(ResourceGroupName) \
                --container-image-name $(acrLoginServer)/$(backImageName):$(backImageTag) \
                --container-registry-url https://$(acrLoginServer) \
```





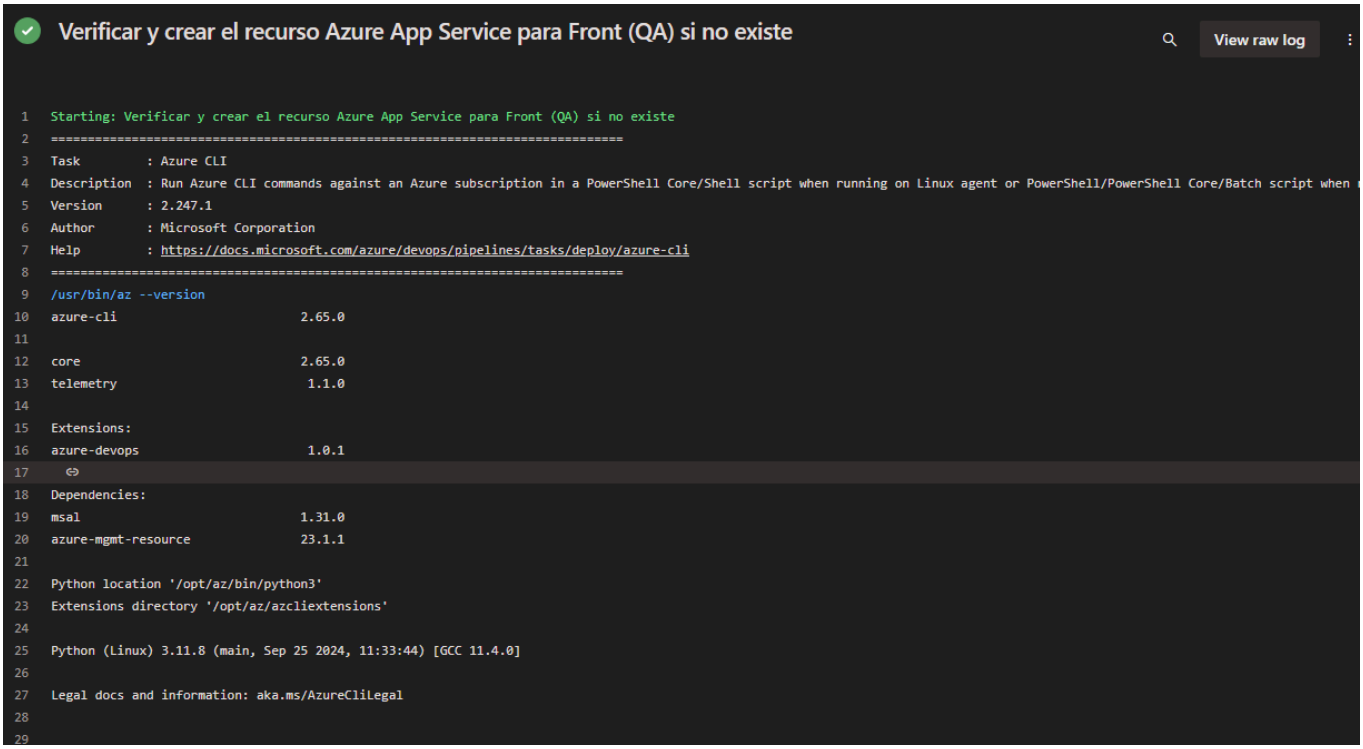
- 4.2.1 Agregar tareas para generar Front en Azure App Service con Soporte para Contenedores

```
#----------------------------------------------------------
# DEPLOY DOCKER FRONT IMAGE TO AZURE APP SERVICE (QA)
#----------------------------------------------------------
Settings
- task: AzureCLI@2
  displayName: 'Verificar y crear el recurso Azure App Service para Front (QA) si no existe'
  inputs:
    azureSubscription: '$(ConnectedServiceName)'
    scriptType: 'bash'
    scriptLocation: 'inlineScript'
    inlineScript: |
      # Verificar si el App Service para el Front ya existe
      if ! az webapp list --query "[?name=='$(WebAppFrontNameContainersQA)' && resourceGroup=='$(ResourceGroupName)']" | length(@)" -o tsv | grep
        echo "El App Service para Front QA no existe. Creando..."
        # Crear el App Service sin especificar la imagen del contenedor
        az webapp create --resource-group $(ResourceGroupName) --plan $(AppServicePlanLinux) --name $(WebAppFrontNameContainersQA) --deployment-c
      else
        echo "El App Service para Front QA ya existe. Actualizando la imagen..."
      fi

      # Configurar el App Service para usar Azure Container Registry (ACR)
      az webapp config container set --name $(WebAppFrontNameContainersQA) --resource-group $(ResourceGroupName) \
        --container-image-name $(acrLoginServer)/$(frontImageName):$(frontImageTag) \
        --container-registry-url https://$(acrLoginServer) \
        --container-registry-user $(acrName) \
        --container-registry-password $(az acr credential show --name $(acrName) --query "passwords[0].value" -o tsv)

      # Establecer variables de entorno para el front-end
      az webapp config appsettings set --name $(WebAppFrontNameContainersQA) --resource-group $(ResourceGroupName) \
        --settings API_URL="http://$(WebAppApiNameContainersQA).azurewebsites.net"
```

```yaml
41
42      # Opcional: Reiniciar las aplicaciones web para aplicar cambios
        Settings
43      - task: AzureCLI@2
44        displayName: 'Reiniciar App Service de API (QA)'
45        inputs:
46          azureSubscription: '$(ConnectedServiceName)'
47          scriptType: 'bash'
48          scriptLocation: 'inlineScript'
49          inlineScript: |
50            az webapp restart --name $(WebAppApiNameContainersQA) --resource-group $(ResourceGroupName)
51
        Settings
52      - task: AzureCLI@2
53        displayName: 'Reiniciar App Service de Front (QA)'
54        inputs:
55          azureSubscription: '$(ConnectedServiceName)'
56          scriptType: 'bash'
57          scriptLocation: 'inlineScript'
58          inlineScript: |
59            az webapp restart --name $(WebAppFrontNameContainersQA) --resource-group $(ResourceGroupName)
60
```



- 4.2.2 Agregar variables necesarias para el funcionamiento de la nueva etapa considerando que debe haber 2 entornos QA y PROD para Back y Front.

```yaml
# Variables para QA
backContainerInstanceNameQA: 'as-crud-api-qa-sm'
frontContainerInstanceNameQA: 'as-crud-front-qa-sm'
backImageTagQA: 'qa'
frontImageTagQA: 'qa'
container-cpu-api-qa: 1 # CPUs de nuestro container de QA
container-memory-api-qa: 1.5 # RAM de nuestro container de QA
WebAppApiNameContainersQA: 'my-webapp-api-qa'
WebAppFrontNameContainersQA: 'my-webapp-front-qa'
container-url-api-qa: 'as-crud-api-qa-sm.westeurope.azurecontainer.io'

# Variables para PROD
backContainerInstanceNamePROD: 'as-crud-api-prod-sm'
frontContainerInstanceNamePROD: 'as-crud-front-prod-sm'
backImageTagPROD: 'prod'
frontImageTagPROD: 'prod'
container-cpu-api-prod: 1 # Ajusta según sea necesario
container-memory-api-prod: 1.5 # Ajusta según sea necesario
WebAppApiNameContainersPROD: 'my-webapp-api-prod'
WebAppFrontNameContainersPROD: 'my-webapp-front-prod'
container-url-api-prod: 'as-crud-api-prod-sm.westeurope.azurecontainer.io'

# Variables comunes
AppServicePlanLinux: 'MiWebApp02'
cnn_string_qaa: '$(cnn_string_qa)' # Usamos la misma cadena para QA y PROD
```

- 4.2.3 Agregar tareas para correr pruebas de integración en el entorno de QA de Back y Front creado en Azure App Services con Soporte para Contenedores.

```yaml
#------------------------------------------------------
# Ejecutar pruebas E2E del Front-End en Azure App Service (QA)
#------------------------------------------------------
    - script: |
        echo "Ejecutando pruebas E2E del Front-End en Azure App Service (QA)..."
        npx cypress run --config baseUrl=https://$(WebAppFrontNameContainersQA).azurewebsites.net
      displayName: 'Ejecutar Pruebas E2E de Front-End en Azure App Service (QA) con Cypress'

    # Publicar resultados de pruebas E2E del Front-End
    Settings
    - task: PublishTestResults@2
      displayName: 'Publicar Resultados de Pruebas E2E de Front-End en Azure App Service (QA)'
      inputs:
        testResultsFormat: 'JUnit'
        testResultsFiles: 'cypress/results/*.xml'
        failTaskOnFailedTests: true
```

- 4.2.4 Agregar etapa que dependa de la etapa de Deploy en QA que genere un entorno de PROD.

```yaml
#-------------------------------------------------------------
### STAGE DEPLOY TO AZURE APP SERVICE PROD
#-------------------------------------------------------------

- stage: DeployImagesToAppServicePROD
  displayName: 'Desplegar Imágenes en Azure App Service (PROD)'
  dependsOn:
    - DeployImagesToAppServiceQA
    - IntegrationTestsQA
  condition: succeeded()
  jobs:
    - job: DeployImagesToAppServicePROD
      displayName: 'Desplegar Imágenes de API y Front en Azure App Service (PROD)'
      pool:
        vmImage: 'ubuntu-latest'
      steps:
        # Fetch ACR credentials
        Settings
        - task: AzureCLI@2
          displayName: 'Fetch ACR credentials'
          inputs:
            azureSubscription: '$(ConnectedServiceName)'
            scriptType: bash
            scriptLocation: inlineScript
            inlineScript: |
              ACR_PASSWORD=$(az acr credential show --name $(acrName) --query "passwords[0].value" -o tsv)
              echo "##vso[task.setvariable variable=ACR_PASSWORD]$ACR_PASSWORD"
450         #-------------------------------------------------------
451         # DEPLOY DOCKER API IMAGE TO AZURE APP SERVICE (PROD)
452         #-------------------------------------------------------
            Settings
453         - task: AzureCLI@2
454           displayName: 'Verificar y crear el recurso Azure App Service para API (PROD) si no existe'
455           inputs:
456             azureSubscription: '$(ConnectedServiceName)'
457             scriptType: 'bash'
458             scriptLocation: 'inlineScript'
459             inlineScript: |
460               # Verificar si el App Service para la API ya existe
461               if ! az webapp list --query "[?name=='$(WebAppApiNameContainersPROD)' && resourceGroup=='$(ResourceGroupName)']" | length(@)" -o tsv | grep
462                 echo "El App Service para API PROD no existe. Creando..."
463                 # Crear el App Service sin especificar la imagen del contenedor
464                 az webapp create --resource-group $(ResourceGroupName) --plan $(AppServicePlanLinux) --name $(WebAppApiNameContainersPROD) --deployment-c
465               else
466                 echo "El App Service para API PROD ya existe. Actualizando la imagen..."
467               fi
468
469               # Configurar el App Service para usar Azure Container Registry (ACR)
470               az webapp config container set --name $(WebAppApiNameContainersPROD) --resource-group $(ResourceGroupName) \
471                 --container-image-name $(acrLoginServer)/$(backImageName):$(backImageTagPROD) \
472                 --container-registry-url https://$(acrLoginServer) \
473                 --container-registry-user $(acrName) \
474                 --container-registry-password $(ACR_PASSWORD)
475
476               # Establecer variables de entorno para el back-end
477               az webapp config appsettings set --name $(WebAppApiNameContainersPROD) --resource-group $(ResourceGroupName) \
478                 --settings ConnectionStrings__DefaultConnection="$(cnn_string_qaa)" # Usamos la misma cadena
479
```

```yaml
#----------------------------------------------------------
# DEPLOY DOCKER FRONT IMAGE TO AZURE APP SERVICE (PROD)
#----------------------------------------------------------
Settings
- task: AzureCLI@2
  displayName: 'Verificar y crear el recurso Azure App Service para Front (PROD) si no existe'
  inputs:
    azureSubscription: '$(ConnectedServiceName)'
    scriptType: 'bash'
    scriptLocation: 'inlineScript'
    inlineScript: |
      # Verificar si el App Service para el Front ya existe
      if ! az webapp list --query "[?name=='$(WebAppFrontNameContainersPROD)' && resourceGroup=='$(ResourceGroupName)'] | length(@)" -o tsv | grep
        echo "El App Service para Front PROD no existe. Creando..."
        # Crear el App Service sin especificar la imagen del contenedor
        az webapp create --resource-group $(ResourceGroupName) --plan $(AppServicePlanLinux) --name $(WebAppFrontNameContainersPROD) --deployment-c
      else
        echo "El App Service para Front PROD ya existe. Actualizando la imagen..."
      fi

      # Configurar el App Service para usar Azure Container Registry (ACR)
      az webapp config container set --name $(WebAppFrontNameContainersPROD) --resource-group $(ResourceGroupName) \
        --container-image-name $(acrLoginServer)/$(frontImageName):$(frontImageTagPROD) \
        --container-registry-url https://$(acrLoginServer) \
        --container-registry-user $(acrName) \
        --container-registry-password $(ACR_PASSWORD)

      # Establecer variables de entorno para el front-end
      az webapp config appsettings set --name $(WebAppFrontNameContainersPROD) --resource-group $(ResourceGroupName) \
        --settings API_URL="http://$(WebAppApiNameContainersPROD).azurewebsites.net"
```

```yaml
# Opcional: Reiniciar las aplicaciones web para aplicar cambios
Settings
- task: AzureCLI@2
  displayName: 'Reiniciar App Service de API (PROD)'
  inputs:
    azureSubscription: '$(ConnectedServiceName)'
    scriptType: 'bash'
    scriptLocation: 'inlineScript'
    inlineScript: |
      az webapp restart --name $(WebAppApiNameContainers) --resource-group $(ResourceGroupName)

Settings
- task: AzureCLI@2
  displayName: 'Reiniciar App Service de Front (PROD)'
  inputs:
    azureSubscription: '$(ConnectedServiceName)'
    scriptType: 'bash'
    scriptLocation: 'inlineScript'
    inlineScript: |
      az webapp restart --name $(WebAppFrontNameContainers) --resource-group $(ResourceGroupName)
```

- 4.2.5 Entregar un pipeline que incluya:

- A) Etapa Construcción y Pruebas Unitarias y Code Coverage Back y Front

```yaml
stages:
- stage: Build
  displayName: 'Build Stage'
  jobs:
    # Job para el Back-End
    - job: Backend
      displayName: 'Build y Análisis del Back-End'
      steps:
        - checkout: self
          fetchDepth: 0

        # Restaurar paquetes NuGet
        Settings
        - task: DotNetCoreCLI@2
          displayName: 'Restore NuGet Packages'
          inputs:
            command: 'restore'
            projects: '$(solution)'

        # Compilar la solución del back-end y pruebas
        Settings
        - task: DotNetCoreCLI@2
          displayName: 'Build Backend y Pruebas Unitarias'
          inputs:
            command: 'build'
            projects: '**/*.sln'  # Esto compila tanto EmployeeCrudApi como EmployeeCrudApi.Tests
            arguments: '--configuration $(configuration)'

        # Publicar Back-End
        Settings
        - task: DotNetCoreCLI@2
          displayName: 'Publicar Back-End'
          inputs:
            command: publish
            publishWebProjects: true
            arguments: '--configuration $(configuration) --output $(Build.ArtifactStagingDirectory)'
            zipAfterPublish: false
```

```yaml
    # Publicar Artefactos del Back-End
      Settings
    - task: PublishBuildArtifacts@1
      displayName: 'Publicar Artefactos de Back'
      inputs:
        pathtoPublish: '$(buildOutput)'
        artifactName: 'drop-back'
        publishLocation: 'Container'

      Settings
    - task: PublishPipelineArtifact@1
      displayName: 'Publicar Dockerfile de Back'
      inputs:
        targetPath: '$(Build.SourcesDirectory)/api/Dockerfile'
        artifact: 'dockerfile-back'

  # Job para el Front-End
  - job: Frontend
    displayName: 'Build y Análisis del Front-End'
    steps:
    - checkout: self
      fetchDepth: 0

    # Instalar Node.js
      Settings
    - task: NodeTool@0
      inputs:
        versionSpec: '18.x'
      displayName: 'Instalar Node.js'

    # Instalar dependencias del front-end
    - script: npm install
      displayName: 'Instalar Dependencias de Front'
      workingDirectory: 'EmployeeCrudAngular'

    # Generar el archivo env.js
    - script: node src/generate-env.js
      displayName: 'Generar archivo env.js'
      workingDirectory: './EmployeeCrudAngular'

    # Compilar el front-end
    - script: npx ng build --configuration production
      displayName: 'Build de Front'
      workingDirectory: './EmployeeCrudAngular'
      condition: succeeded()

    # Publicar artefactos del front-end
      Settings
    - task: PublishBuildArtifacts@1
      displayName: 'Publicar Artefactos de Front'
      inputs:
        pathtoPublish: './EmployeeCrudAngular/dist'
        artifactName: 'drop-front'
        publishLocation: 'Container'

      Settings
    - task: PublishPipelineArtifact@1
      displayName: 'Publicar Dockerfile de Front'
      inputs:
        targetPath: '$(Build.SourcesDirectory)/front/Dockerfile'
        artifact: 'dockerfile-front'
```

- B) Construcción de Imágenes Docker y subida a ACR

```yaml
#-------------------------------------------------------------
### STAGE BUILD DOCKER IMAGES Y PUSH A AZURE CONTAINER REGISTRY
#-------------------------------------------------------------

- stage: DockerBuildAndPush
  displayName: 'Construir y Subir Imágenes Docker a ACR'
  jobs:
    - job: docker_build_and_push
      displayName: 'Construir y Subir Imágenes Docker a ACR'
      pool:
        vmImage: 'ubuntu-latest'

      steps:
        - checkout: self
```

```yaml
153         #-------------------------------------------------------------
154         # BUILD DOCKER BACK IMAGE Y PUSH A AZURE CONTAINER REGISTRY
155         #-------------------------------------------------------------
156
            Settings
157         - task: DownloadPipelineArtifact@2
158           displayName: 'Descargar Artefactos de Back'
159           inputs:
160             buildType: 'current'
161             artifactName: 'drop-back'
162             targetPath: '$(Pipeline.Workspace)/drop-back'
163
            Settings
164         - task: DownloadPipelineArtifact@2
165           displayName: 'Descargar Dockerfile de Back'
166           inputs:
167             buildType: 'current'
168             artifactName: 'dockerfile-back'
169             targetPath: '$(Pipeline.Workspace)/dockerfile-back'
170
            Settings
171         - task: AzureCLI@2
172           displayName: 'Iniciar Sesión en Azure Container Registry (ACR)'
173           inputs:
174             azureSubscription: '$(ConnectedServiceName)'
175             scriptType: bash
176             scriptLocation: inlineScript
177             inlineScript: |
178               az acr login --name $(acrName)
179
            Settings
180         - task: Docker@2
181           displayName: 'Construir Imagen Docker para Back'
182           inputs:
183             command: build
184             repository: $(acrLoginServer)/$(backImageName)
185             dockerfile: $(Pipeline.Workspace)/dockerfile-back/Dockerfile
186             buildContext: $(Pipeline.Workspace)/drop-back
187             tags: '$(backImageTagQA),$(backImageTagPROD)'
```

```yaml
                Settings
189         - task: Docker@2
190           displayName: 'Subir Imagen Docker de Back a ACR'
191           inputs:
192             command: push
193             repository: $(acrLoginServer)/$(backImageName)
194             tags: '$(backImageTagQA),$(backImageTagPROD)'
195
```

```yaml
#----------------------------------------------------------
# BUILD DOCKER FRONT IMAGE Y PUSH A AZURE CONTAINER REGISTRY
#----------------------------------------------------------

Settings
- task: DownloadPipelineArtifact@2
  displayName: 'Descargar Artefactos de Front'
  inputs:
    buildType: 'current'
    artifactName: 'drop-front'
    targetPath: '$(Pipeline.Workspace)/drop-front'

Settings
- task: DownloadPipelineArtifact@2
  displayName: 'Descargar Dockerfile de Front'
  inputs:
    buildType: 'current'
    artifactName: 'dockerfile-front'
    targetPath: '$(Pipeline.Workspace)/dockerfile-front'

Settings
- task: Docker@2
  displayName: 'Construir Imagen Docker para Front'
  inputs:
    command: build
    repository: $(acrLoginServer)/$(frontImageName)
    dockerfile: $(Pipeline.Workspace)/dockerfile-front/Dockerfile
    buildContext: $(Pipeline.Workspace)/drop-front
    tags: '$(frontImageTagQA),$(frontImageTagPROD)'

Settings
- task: Docker@2
  displayName: 'Subir Imagen Docker de Front a ACR'
  inputs:
    command: push
    repository: $(acrLoginServer)/$(frontImageName)
    tags: '$(frontImageTagQA),$(frontImageTagPROD)'
```

- C) Deploy Back y Front en QA con pruebas de integración para Azure Web Apps

```yaml
#---------------------------------------
### STAGE: Deploy to Azure Web Apps (QA)
#---------------------------------------
- stage: DeployToAzureWebAppsQA
  displayName: 'Deploy to Azure Web Apps (QA)'
  dependsOn: DockerBuildAndPush
  condition: succeeded()
  jobs:
    # Job para el Back-End
    - job: DeployQABackend
      displayName: 'Deploy to QA - Back-End'
      steps:
```

Settings

```yaml
      - task: DownloadPipelineArtifact@2
        displayName: 'Download Build Artifacts'
        inputs:
          buildType: 'current'
          artifactName: 'drop-back'
          targetPath: '$(Pipeline.Workspace)/drop-back'

```

Settings

```yaml
      - task: AzureWebApp@1
        displayName: 'Deploy to QA - Back-End'
        inputs:
          azureSubscription: '$(ConnectedServiceName)'
          appType: 'webApp'
          appName: 'EmployeeCrudBack-QA'
          package: '$(Pipeline.Workspace)/drop-back'

    # Job para el Front-End
    - job: DeployQAFrontend
      displayName: 'Deploy to QA - Front-End'
      steps:
```

Settings

```yaml
      - task: DownloadPipelineArtifact@2
        displayName: 'Download Build Artifacts'
        inputs:
          buildType: 'current'
          artifactName: 'drop-front'
          targetPath: '$(Pipeline.Workspace)/drop-front'

```

Settings

```yaml
      - task: AzureWebApp@1
        displayName: 'Deploy to QA - Front-End'
        inputs:
          azureSubscription: '$(ConnectedServiceName)'
          appType: 'webApp'
          appName: 'EmployeeCrudFront-QA'
          package: '$(Pipeline.Workspace)/drop-front/employee-crud-angular/browser'
```

```yaml
    # Job para los Test de Integración (ejecutado después del despliegue)
    - job: IntegrationTestsAzureWebAppsQA
      displayName: 'Tests de Integracion en Azure Web Apps (QA)'
      dependsOn:
        - DeployQABackend
        - DeployQAFrontend
      steps:
      - script: npm install
        workingDirectory: ./EmployeeCrudAngular
        displayName: 'Instalar Dependencias para Pruebas'

      - script: npx cypress run --config baseUrl=https://EmployeeCRUDFront.azurewebsites.net
        workingDirectory: ./EmployeeCrudAngular
        displayName: 'Correr Tests en Cypress E2E'

    Settings
      - task: PublishTestResults@2
        displayName: 'Publicar Resultados de Cypress'
        inputs:
          testResultsFormat: 'JUnit'
          testResultsFiles: 'EmployeeCrudAngular/cypress/results/*.xml'
          testRunTitle: 'Cypress E2E Tests - QA'
          failTaskOnFailedTests: true
```

- D) Deploy Back y Front en QA con pruebas de integración para ACI

```yaml
#-------------------------------------------------------------
### STAGE DEPLOY TO ACI QA
#-------------------------------------------------------------

- stage: DeployToACIQA
  displayName: 'Desplegar en Azure Container Instances (ACI) QA'
  dependsOn: DockerBuildAndPush
  jobs:
    - job: deploy_to_aci_qa
      displayName: 'Desplegar en Azure Container Instances (ACI) QA'
      pool:
        vmImage: 'ubuntu-latest'
      steps:

        # Fetch ACR credentials
          Settings
        - task: AzureCLI@2
          displayName: 'Fetch ACR credentials'
          inputs:
            azureSubscription: '$(ConnectedServiceName)'
            scriptType: bash
            scriptLocation: inlineScript
            inlineScript: |
              ACR_PASSWORD=$(az acr credential show --name $(acrName) --query "passwords[0].value" -o tsv)
              echo "##vso[task.setvariable variable=ACR_PASSWORD]$ACR_PASSWORD"

        # DEPLOY DOCKER BACK IMAGE TO ACI QA
          Settings
        - task: AzureCLI@2
          displayName: 'Desplegar Imagen Docker de Back en ACI QA'
          inputs:
            azureSubscription: '$(ConnectedServiceName)'
            scriptType: bash
            scriptLocation: inlineScript
            inlineScript: |
              echo "Resource Group: $(ResourceGroupName)"
              echo "Container Instance Name: $(backContainerInstanceNameQA)"
              echo "ACR Login Server: $(acrLoginServer)"
              echo "Image Name: $(backImageName)"
              echo "Image Tag: $(backImageTagQA)"
              echo "Connection String: $(cnn_string_qaa)"

              # Delete the existing container if present
              az container delete --resource-group $(ResourceGroupName) --name $(backContainerInstanceNameQA) --yes || true

              # Create the container
              az container create --resource-group $(ResourceGroupName) \
                --location westeurope \
                --name $(backContainerInstanceNameQA) \
                --image $(acrLoginServer)/$(backImageName):$(backImageTagQA) \
                --registry-login-server $(acrLoginServer) \
                --registry-username $(acrName) \
                --registry-password $(ACR_PASSWORD) \
                --dns-name-label $(backContainerInstanceNameQA) \
                --ports 80 \
                --environment-variables ConnectionStrings__DefaultConnection="$(cnn_string_qaa)" \
                --restart-policy Always \
                --cpu $(container-cpu-api-qa) \
                --memory $(container-memory-api-qa)
```

```yaml
    # DEPLOY DOCKER FRONT IMAGE TO ACI QA
      Settings
    - task: AzureCLI@2
      displayName: 'Desplegar Imagen Docker de Front en ACI QA'
      inputs:
        azureSubscription: '$(ConnectedServiceName)'
        scriptType: bash
        scriptLocation: inlineScript
        inlineScript: |
          echo "Resource Group: $(ResourceGroupName)"
          echo "Container Instance Name: $(frontContainerInstanceNameQA)"
          echo "ACR Login Server: $(acrLoginServer)"
          echo "Image Name: $(frontImageName)"
          echo "Image Tag: $(frontImageTagQA)"

          # Delete the existing front container if present
          az container delete --resource-group $(ResourceGroupName) --name $(frontContainerInstanceNameQA) --yes || true

          # Create the front-end container
          az container create --resource-group $(ResourceGroupName) \
            --location westeurope \
            --name $(frontContainerInstanceNameQA) \
            --image $(acrLoginServer)/$(frontImageName):$(frontImageTagQA) \
            --registry-login-server $(acrLoginServer) \
            --registry-username $(acrName) \
            --registry-password $(ACR_PASSWORD) \
            --dns-name-label $(frontContainerInstanceNameQA) \
            --ports 80 \
            --environment-variables API_URL=$(container-url-api-qa) \
            --restart-policy Always \
            --cpu 1 \
            --memory 1.5

    # Ejecutar pruebas E2E del Front-End en el contenedor de ACI QA
    - script: |
        echo "Ejecutando pruebas E2E del Front-End en ACI QA..."
        npx cypress run --config baseUrl=http://$(frontContainerInstanceNameQA).westeurope.azurecontainer.io
      displayName: 'Ejecutar Pruebas E2E de Front-End en ACI QA con Cypress'

    # Publicar resultados de pruebas E2E del Front-End
      Settings
    - task: PublishTestResults@2
      displayName: 'Publicar Resultados de Pruebas E2E de Front-End en ACI QA'
      inputs:
        testResultsFormat: 'JUnit'
        testResultsFiles: 'cypress/results/*.xml'
        failTaskOnFailedTests: true
```

- E) Deploy Back y Front en QA con pruebas de integración para Azure Web Apps con Soporte para contenedores

```yaml
#-------------------------------------------------------------
### STAGE DEPLOY TO AZURE APP SERVICE QA
#-------------------------------------------------------------

- stage: DeployImagesToAppServiceQA
  displayName: 'Desplegar Imágenes en Azure App Service (QA)'
  dependsOn:
    - DockerBuildAndPush
  condition: succeeded()
  jobs:
    - job: DeployImagesToAppServiceQA
      displayName: 'Desplegar Imágenes de API y Front en Azure App Service (QA)'
      pool:
        vmImage: 'ubuntu-latest'
      steps:
        # Fetch ACR credentials
          Settings
        - task: AzureCLI@2
          displayName: 'Fetch ACR credentials'
          inputs:
            azureSubscription: '$(ConnectedServiceName)'
            scriptType: bash
            scriptLocation: inlineScript
            inlineScript: |
              ACR_PASSWORD=$(az acr credential show --name $(acrName) --query "passwords[0].value" -o tsv)
              echo "##vso[task.setvariable variable=ACR_PASSWORD]$ACR_PASSWORD"
```

```yaml
#----------------------------------------------------------
# DEPLOY DOCKER API IMAGE TO AZURE APP SERVICE (QA)
#----------------------------------------------------------
Settings
- task: AzureCLI@2
  displayName: 'Verificar y crear el recurso Azure App Service para API (QA) si no existe'
  inputs:
    azureSubscription: '$(ConnectedServiceName)'
    scriptType: 'bash'
    scriptLocation: 'inlineScript'
    inlineScript: |
      # Verificar si el App Service para la API ya existe
      if ! az webapp list --query "[?name=='$(WebAppApiNameContainersQA)' && resourceGroup=='$(ResourceGroupName)']" | length(@)" -o tsv | grep -q '
        echo "El App Service para API QA no existe. Creando..."
        # Crear el App Service sin especificar la imagen del contenedor
        az webapp create --resource-group $(ResourceGroupName) --plan $(AppServicePlanLinux) --name $(WebAppApiNameContainersQA) --deployment-conta
      else
        echo "El App Service para API QA ya existe. Actualizando la imagen..."
      fi

      # Configurar el App Service para usar Azure Container Registry (ACR)
      az webapp config container set --name $(WebAppApiNameContainersQA) --resource-group $(ResourceGroupName) \
        --container-image-name $(acrLoginServer)/$(backImageName):$(backImageTagQA) \
        --container-registry-url https://$(acrLoginServer) \
        --container-registry-user $(acrName) \
        --container-registry-password $(ACR_PASSWORD)

      # Establecer variables de entorno para el back-end
      az webapp config appsettings set --name $(WebAppApiNameContainersQA) --resource-group $(ResourceGroupName) \
        --settings ConnectionStrings__DefaultConnection="$(cnn_string_qaa)"
```

```yaml
#----------------------------------------------------------
# DEPLOY DOCKER FRONT IMAGE TO AZURE APP SERVICE (QA)
#----------------------------------------------------------
Settings
- task: AzureCLI@2
  displayName: 'Verificar y crear el recurso Azure App Service para Front (QA) si no existe'
  inputs:
    azureSubscription: '$(ConnectedServiceName)'
    scriptType: 'bash'
    scriptLocation: 'inlineScript'
    inlineScript: |
      # Verificar si el App Service para el Front ya existe
      if ! az webapp list --query "[?name=='$(WebAppFrontNameContainersQA)' && resourceGroup=='$(ResourceGroupName)']" | length(@)" -o tsv | grep -
        echo "El App Service para Front QA no existe. Creando..."
        # Crear el App Service sin especificar la imagen del contenedor
        az webapp create --resource-group $(ResourceGroupName) --plan $(AppServicePlanLinux) --name $(WebAppFrontNameContainersQA) --deployment-con
      else
        echo "El App Service para Front QA ya existe. Actualizando la imagen..."
      fi

      # Configurar el App Service para usar Azure Container Registry (ACR)
      az webapp config container set --name $(WebAppFrontNameContainersQA) --resource-group $(ResourceGroupName) \
        --container-image-name $(acrLoginServer)/$(frontImageName):$(frontImageTagQA) \
        --container-registry-url https://$(acrLoginServer) \
        --container-registry-user $(acrName) \
        --container-registry-password $(ACR_PASSWORD)

      # Establecer variables de entorno para el front-end
      az webapp config appsettings set --name $(WebAppFrontNameContainersQA) --resource-group $(ResourceGroupName) \
        --settings API_URL="http://$(WebAppApiNameContainersQA).azurewebsites.net"
```

```yaml
#----------------------------------------------------------
# Ejecutar pruebas E2E del Front-End en Azure App Service (QA)
#----------------------------------------------------------
    - script: |
        echo "Ejecutando pruebas E2E del Front-End en Azure App Service (QA)..."
        npx cypress run --config baseUrl=https://$(WebAppFrontNameContainersQA).azurewebsites.net
      displayName: 'Ejecutar Pruebas E2E de Front-End en Azure App Service (QA) con Cypress'

    # Publicar resultados de pruebas E2E del Front-End
    Settings
    - task: PublishTestResults@2
      displayName: 'Publicar Resultados de Pruebas E2E de Front-End en Azure App Service (QA)'
      inputs:
        testResultsFormat: 'JUnit'
        testResultsFiles: 'cypress/results/*.xml'
        failTaskOnFailedTests: true
```

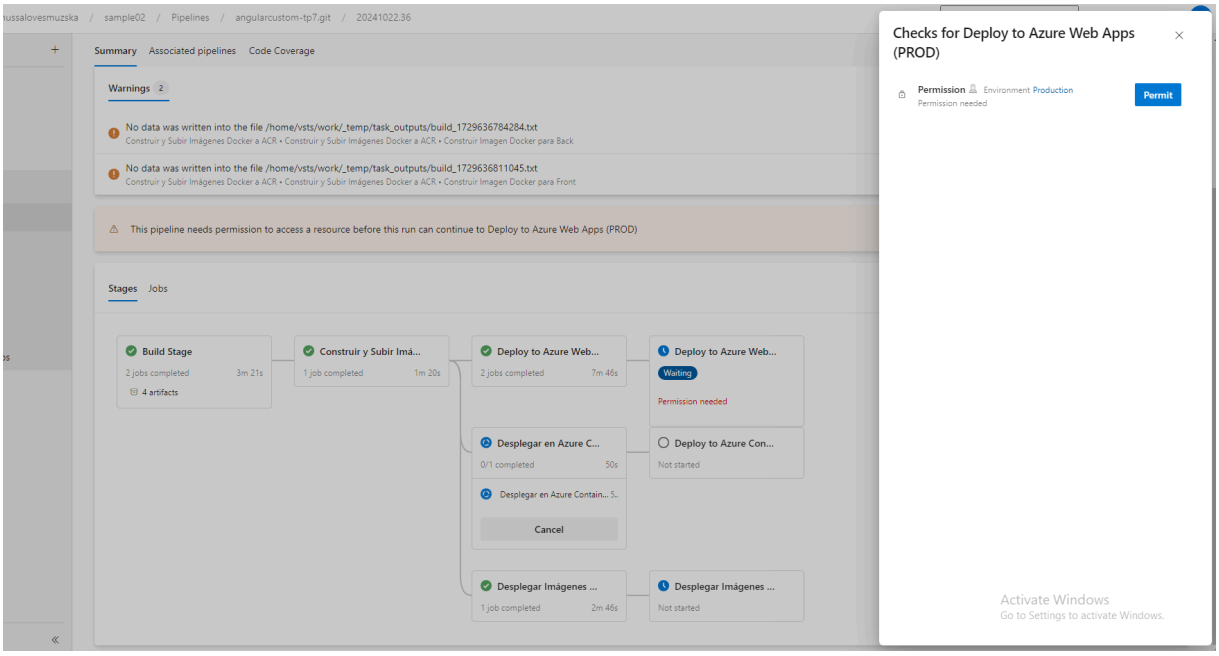
- F) Aprobación manual de QA para los puntos C,D,E

- G) Deploy Back y Front en PROD para Azure Web Apps

```yaml
#--------------------------------------
### STAGE: Deploy to Azure Web Apps (PROD)
#--------------------------------------
- stage: DeployToAzureWebAppsProd
  displayName: 'Deploy to Azure Web Apps (PROD)'
  dependsOn: DeployToAzureWebAppsQA
  condition: succeeded()
  jobs:
  - deployment: DeployToAzureWebAppsProd
    displayName: 'Deploy to Azure Web Apps (PROD)'
    environment: 'Production'
    strategy:
      runOnce:
        deploy:
          steps:
            Settings
          - task: DownloadPipelineArtifact@2
            displayName: 'Download Build Artifacts'
            inputs:
              buildType: 'current'
              artifactName: 'drop-back'
              targetPath: '$(Pipeline.Workspace)/drop-back'

            Settings
          - task: AzureWebApp@1
            displayName: 'Deploy to PROD - Back-End'
            inputs:
              azureSubscription: '$(ConnectedServiceName)'
              appType: 'webApp'
              appName: 'EmployeeCrudBack-Prod'
              package: '$(Pipeline.Workspace)/drop-back'
```

```
                Settings
551             - task: AzureWebApp@1
552               displayName: 'Deploy to PROD - Back-End'
553               inputs:
554                 azureSubscription: '$(ConnectedServiceName)'
555                 appType: 'webApp'
556                 appName: 'EmployeeCrudBack-Prod'
557                 package: '$(Pipeline.Workspace)/drop-back'
558
                Settings
559             - task: DownloadPipelineArtifact@2
560               displayName: 'Download Build Artifacts'
561               inputs:
562                 buildType: 'current'
563                 artifactName: 'drop-front'
564                 targetPath: '$(Pipeline.Workspace)/drop-front'
565
                Settings
566             - task: AzureWebApp@1
567               displayName: 'Deploy to PROD - Front-End'
568               inputs:
569                 azureSubscription: '$(ConnectedServiceName)'
570                 appType: 'webApp'
571                 appName: 'EmployeeCrudFront-Prod'
572                 package: '$(Pipeline.Workspace)/drop-front/employee-crud-angular/browser'
```



- H) Deploy Back y Front en PROD para ACI

```yaml
#-------------------------------------------------------------
### STAGE: Deploy to ACI (PROD)
#-------------------------------------------------------------
- stage: DeployToACIProd
  displayName: 'Deploy to Azure Container Instances (ACI) PROD'
  dependsOn: DeployToACIQA
  condition: succeeded()
  jobs:
    - deployment: DeployToACIProd
      displayName: 'Deploy to Azure Container Instances (ACI) PROD'
      environment: 'Production'
      strategy:
        runOnce:
          deploy:
            steps:
              # Fetch ACR credentials
              Settings
              - task: AzureCLI@2
                displayName: 'Fetch ACR credentials for PROD'
                inputs:
                  azureSubscription: '$(ConnectedServiceName)'
                  scriptType: bash
                  scriptLocation: inlineScript
                  inlineScript: |
                    ACR_PASSWORD=$(az acr credential show --name $(acrName) --query "passwords[0].value" -o tsv)
                    echo "##vso[task.setvariable variable=ACR_PASSWORD]$ACR_PASSWORD"

              # DEPLOY DOCKER BACK IMAGE TO ACI PROD
              Settings
              - task: AzureCLI@2
                displayName: 'Deploy Back-End Docker Image to ACI (PROD)'
                inputs:
                  azureSubscription: '$(ConnectedServiceName)'
                  scriptType: bash
                  scriptLocation: inlineScript
                  inlineScript: |
                    echo "Resource Group: $(ResourceGroupName)"
                    echo "Container Instance Name: $(backContainerInstanceNamePROD)"
                    echo "ACR Login Server: $(acrLoginServer)"
                    echo "Image Name: $(backImageName)"
                    echo "Image Tag: $(backImageTagPROD)"
                    echo "Connection String: $(cnn_string_prod)"  # Usamos cnn_string_qaa

                    # Delete the existing container if present
                    az container delete --resource-group $(ResourceGroupName) --name $(backContainerInstanceNamePROD) --yes || true

                    # Create the container
                    az container create --resource-group $(ResourceGroupName) \
                      --location westeurope \
                      --name $(backContainerInstanceNamePROD) \
                      --image $(acrLoginServer)/$(backImageName):$(backImageTagPROD) \
                      --registry-login-server $(acrLoginServer) \
                      --registry-username $(acrName) \
                      --registry-password $(ACR_PASSWORD) \
                      --dns-name-label $(backContainerInstanceNamePROD) \
                      --ports 80 \
                      --environment-variables ConnectionStrings__DefaultConnection="$(cnn_string_qaa)" \
                      --restart-policy Always \
                      --cpu $(container-cpu-api-prod) \
                      --memory $(container-memory-api-prod)

              # DEPLOY DOCKER FRONT IMAGE TO ACI PROD
              Settings
              - task: AzureCLI@2
                displayName: 'Deploy Front-End Docker Image to ACI (PROD)'
                inputs:
                  azureSubscription: '$(ConnectedServiceName)'
                  scriptType: bash
                  scriptLocation: inlineScript
                  inlineScript: |
                    echo "Resource Group: $(ResourceGroupName)"
                    echo "Container Instance Name: $(frontContainerInstanceNamePROD)"
                    echo "ACR Login Server: $(acrLoginServer)"
                    echo "Image Name: $(frontImageName)"
                    echo "Image Tag: $(frontImageTagPROD)"

                    # Delete the existing container if present
                    az container delete --resource-group $(ResourceGroupName) --name $(frontContainerInstanceNamePROD) --yes || true

                    # Create the container
                    az container create --resource-group $(ResourceGroupName) \
                      --location westeurope \
                      --name $(frontContainerInstanceNamePROD) \
                      --image $(acrLoginServer)/$(frontImageName):$(frontImageTagPROD) \
                      --registry-login-server $(acrLoginServer) \
                      --registry-username $(acrName) \
                      --registry-password $(ACR_PASSWORD) \
                      --dns-name-label $(frontContainerInstanceNamePROD) \
                      --ports 80 \
                      --environment-variables API_URL=$(container-url-api-prod) \
                      --restart-policy Always \
                      --cpu 1 \
                      --memory 1.5
```

- I) Deploy Back y Front en PROD para Azure Web Apps con Soporte para contenedores

```yaml
#-------------------------------------------------------------
### STAGE DEPLOY TO AZURE APP SERVICE PROD
#-------------------------------------------------------------

- stage: DeployImagesToAppServicePROD
  displayName: 'Desplegar Imágenes en Azure App Service (PROD)'
  dependsOn:
    - DeployImagesToAppServiceQA
  condition: succeeded()
  jobs:
    - job: DeployImagesToAppServicePROD
      displayName: 'Desplegar Imágenes de API y Front en Azure App Service (PROD)'
      pool:
        vmImage: 'ubuntu-latest'
      steps:
        # Fetch ACR credentials
        Settings
        - task: AzureCLI@2
          displayName: 'Fetch ACR credentials'
          inputs:
            azureSubscription: '$(ConnectedServiceName)'
            scriptType: bash
            scriptLocation: inlineScript
            inlineScript: |
              ACR_PASSWORD=$(az acr credential show --name $(acrName) --query "passwords[0].value" -o tsv)
              echo "##vso[task.setvariable variable=ACR_PASSWORD]$ACR_PASSWORD"

---
        #-----------------------------------------------------
        # DEPLOY DOCKER API IMAGE TO AZURE APP SERVICE (PROD)
        #-----------------------------------------------------
        Settings
        - task: AzureCLI@2
          displayName: 'Verificar y crear el recurso Azure App Service para API (PROD) si no existe'
          inputs:
            azureSubscription: '$(ConnectedServiceName)'
            scriptType: 'bash'
            scriptLocation: 'inlineScript'
            inlineScript: |
              # Verificar si el App Service para la API ya existe
              if ! az webapp list --query "[?name=='$(WebAppApiNameContainersPROD)' && resourceGroup=='$(ResourceGroupName)']" | length(@)" -o tsv | grep
                echo "El App Service para API PROD no existe. Creando..."
                # Crear el App Service sin especificar la imagen del contenedor
                az webapp create --resource-group $(ResourceGroupName) --plan $(AppServicePlanLinux) --name $(WebAppApiNameContainersPROD) --deployment-c
              else
                echo "El App Service para API PROD ya existe. Actualizando la imagen..."
              fi

              # Configurar el App Service para usar Azure Container Registry (ACR)
              az webapp config container set --name $(WebAppApiNameContainersPROD) --resource-group $(ResourceGroupName) \
                --container-image-name $(acrLoginServer)/$(backImageName):$(backImageTagPROD) \
                --container-registry-url https://$(acrLoginServer) \
                --container-registry-user $(acrName) \
                --container-registry-password $(ACR_PASSWORD)

              # Establecer variables de entorno para el back-end
              az webapp config appsettings set --name $(WebAppApiNameContainersPROD) --resource-group $(ResourceGroupName) \
                --settings ConnectionStrings__DefaultConnection="$(cnn_string_qaa)" # Usamos la misma cadena

        #-----------------------------------------------------
        # DEPLOY DOCKER FRONT IMAGE TO AZURE APP SERVICE (PROD)
        #-----------------------------------------------------
        Settings
        - task: AzureCLI@2
          displayName: 'Verificar y crear el recurso Azure App Service para Front (PROD) si no existe'
          inputs:
            azureSubscription: '$(ConnectedServiceName)'
            scriptType: 'bash'
            scriptLocation: 'inlineScript'
            inlineScript: |
              # Verificar si el App Service para el Front ya existe
              if ! az webapp list --query "[?name=='$(WebAppFrontNameContainersPROD)' && resourceGroup=='$(ResourceGroupName)']" | length(@)" -o tsv | grep
                echo "El App Service para Front PROD no existe. Creando..."
                # Crear el App Service sin especificar la imagen del contenedor
                az webapp create --resource-group $(ResourceGroupName) --plan $(AppServicePlanLinux) --name $(WebAppFrontNameContainersPROD) --deployment-
              else
                echo "El App Service para Front PROD ya existe. Actualizando la imagen..."
              fi

              # Configurar el App Service para usar Azure Container Registry (ACR)
              az webapp config container set --name $(WebAppFrontNameContainersPROD) --resource-group $(ResourceGroupName) \
                --container-image-name $(acrLoginServer)/$(frontImageName):$(frontImageTagPROD) \
                --container-registry-url https://$(acrLoginServer) \
                --container-registry-user $(acrName) \
                --container-registry-password $(ACR_PASSWORD)

              # Establecer variables de entorno para el front-end
              az webapp config appsettings set --name $(WebAppFrontNameContainersPROD) --resource-group $(ResourceGroupName) \
                --settings API_URL="http://$(WebAppApiNameContainersPROD).azurewebsites.net"
```

Pipeline Completo: