

Ingeniería en Software 3

TP9

Santiago Agüero

Actividad 4.1

- Agregar a nuestro pipeline una nueva etapa que dependa de nuestra etapa de Construcción y Pruebas y de la etapa de Construcción de Imágenes Docker y subida a ACR realizada en el TP08
- Agregar tareas para crear un recurso Azure Container Instances que levante un contenedor con nuestra imagen de back utilizando un AppServicePlan en Linux
- Creamos el recurso:

Microsoft Azure

Inicio > Planes de App Service >

Crear plan de App Service

Detalles del proyecto

Seleccione una suscripción para administrar los recursos implementados y los costos. Use los grupos de recursos como carpetas para organizar y administrar todos los recursos.

Suscripción *

Grupo de recursos *
[Crear nuevo](#)

Detalles del plan de App Service

Nombre *

Sistema operativo * ☒ Linux ☐ Windows

Región *

Plan de tarifa

El plan de tarifa de App Service determina la ubicación, las características, los costos y los recursos del proceso asociados a la aplicación. [Más información](#)

Plan de precios
[Explorar planes de precios](#)

[Revisar y crear](#) [< Anterior](#) [Siguiente: Etiquetas >](#)

Microsoft Azure

Inicio > Microsoft.Web-ASP-Portal-bc40b236-a4c4 | Información general >

LinuxAppPlan01

Plan de Linux

Buscar

Eliminar [Envíenos sus comentarios](#)

Introducción

- Registro de actividad
- Control de acceso (IAM)
- Etiquetas
- Diagnosticar y solucionar problemas
- Eventos (versión preliminar)
- Configuración
- Supervisión
- Automation
- Ayuda

Información esencial

Grupo de recursos ([mover](#)) : TPSingSw3UCC2024

Estado : Listo

Ubicación : Canada East

Suscripción ([mover](#)) : [Suscripción de Azure 1](#)

Id. de suscripción : 9d2177ae-435d-4648-8377-5298f8b6d4cd

Etiquetas ([editar](#)) : [Agregar etiquetas](#)

Plan de precios : [F1](#)

Recuento de instancias : [1](#)

Aplicaciones y espacios : [0 / 0](#)

Sistema operativo : Linux

Con redundancia de zona : Deshabilitado

Porcentaje de CPU

Porcentaje de memoria

Datos de entrada

- Actualizamos variables y agregamos al pipeline las tareas para hacer el deploy a App Service para contenedores.

```

37 container: memory: 100m
38 WebAppApiNameContainersQA: 'backend-container-qa'
39 AppServicePlanLinux: 'LinuxAppPlan01'
40

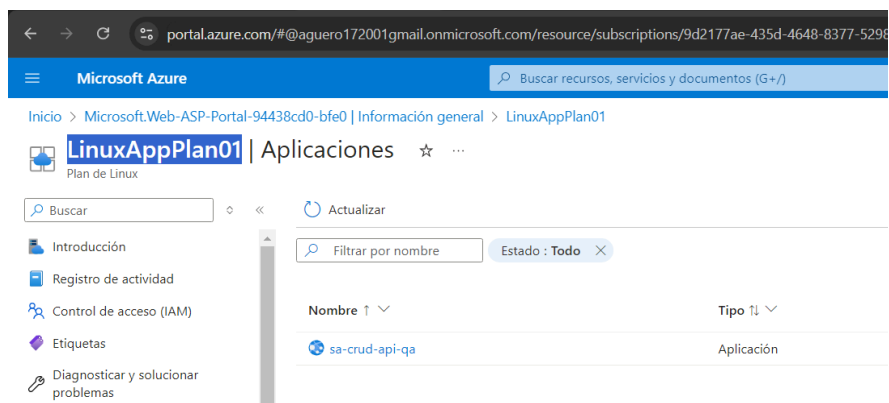
← TP7 Variables Validate and save

main / azure-pipelines.yml *

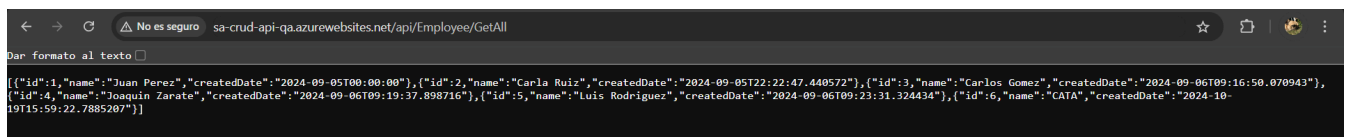
429
430
431
432 ## STAGE DEPLOY TO AZURE APP SERVICE QA
433 #-----
434 stage: DeployImagesToAppServiceQA
435 displayName: 'Desplegar Imágenes en Azure App Service (QA)'
436 dependsOn:
437   - BuildAndTestBackAndFront
438   - DockerBuildAndPush
439 condition: succeeded()
440 jobs:
441   - job: DeployImagesToAppServiceQA
442     displayName: 'Desplegar Imágenes de API y Front en Azure App Service (QA)'
443     pool:
444       vmImage: 'ubuntu-latest'
445     steps:
446       #-----
447       # DEPLOY DOCKER API IMAGE TO AZURE APP SERVICE (QA)
448       #-----
449       task: AzureCLI@2
450       displayName: 'Verificar y crear el recurso Azure App Service para API (QA) si no existe'
451       inputs:
452         azureSubscription: '$(ConnectedServiceName)'
453         scriptType: 'bash'
454         scriptLocation: 'inlineScript'
455         inlineScript: |
456           # Verificar si el App Service para la API ya existe
457           if ! az webapp list --query "[?name=='$(WebAppApiNameContainersQA)' && resourceGroup=='$(ResourceGroupName)']" | length 0 -o tsv | grep -q '15'; then
458             echo "El App Service para API QA no existe. Creando..."
459             # Crear el App Service sin especificar la imagen del contenedor
460             az webapp create --resource-group $(ResourceGroupName) --plan $(AppServicePlanLinux) --name $(WebAppApiNameContainersQA) --deployment-container-image-name "nginx" # Especifica una imagen
461           else
462             echo "El App Service para API QA ya existe. Actualizando la imagen..."
463           fi
464       # Configurar el App Service para usar Azure Container Registry (ACR)
465       az webapp config container set --name $(WebAppApiNameContainersQA) --resource-group $(ResourceGroupName) \
466         --container-image-name $(acrLoginServer)/$(backImageName):$(backImageTag) \
467         --container-registry-url https://$(acrLoginServer) \
468

```

- Ahora corremos el pipeline:



- Chequeamos que este todo funcionando



Actividad 4.2 (DESAFÍOS)

1. Agregar tareas para generar Front en Azure App Service con Soporte para Contenedores

```
/ TP7 / Pipelines [
< TP7
main TP7 / azure-pipelines.yml *
304
305 Settings
306 - task: AzureCLI@2
307   displayName: 'Verificar y crear el recurso Azure App Service para FRONT (QA) si no existe'
308   inputs:
309     azureSubscription: '$(ConnectedServiceName)' # Nombre de la conexión a Azure
310     scriptType: 'bash'
311     scriptLocation: 'inlineScript'
312     inlineScript: |
313       # Verificar si el App Service ya existe
314       if [ $(az webapp list --query "[?name=='$(WebAppFrontNameContainersQA)' && resourceGroup=='$(ResourceGroupName)']" | length(@)) -eq 0 ]; then
315         echo "El App Service para FRONT QA no existe. Creando..."
316         az webapp create \
317           --resource-group $(ResourceGroupName) \
318           --plan $(AppServicePlanLinux) \
319           --name $(WebAppFrontNameContainersQA) \
320           --deployment-container-image-name "nginx"
321       else
322         echo "El App Service para FRONT QA ya existe. Actualizando la imagen..."
323         fi
324       # Configurar el contenedor desde ACR
325       az webapp config container set \
326         --name $(WebAppFrontNameContainersQA) \
327         --resource-group $(ResourceGroupName) \
328         --container-image-name $(acrLoginServer)/$(frontImageName):$(frontImageTag) \
329         --container-registry-url https://$(acrLoginServer) \
330         --container-registry-user $(acrName) \
331         --container-registry-password $(az acr credential show --name $(acrName) --query "passwords[0].value" -o tsv)
332       # Establecer variables de entorno
333       az webapp config appsettings set \
334         --name $(WebAppFrontNameContainersQA) \
335         --resource-group $(ResourceGroupName) \
336         --settings API_URL="$(API_URL_QA)"
337
338
```

- Y vemos que ahora se creó el container de front.

LinuxAppPlan01 | Aplicaciones

Buscar

Actualizar

Filtrar por nombre Estado: Todo

Nombre	Tipo	Variante	Grupo de recursos	Estado
sa-crud-api-qa	Aplicación	app.linux.container	TPSingSw3UCC2024	En ejecución
sa-crud-front-qa	Aplicación	app.linux.container	tpsingsw3ucc2024	En ejecución

No es seguro sa-crud-front-qa.azurewebsites.net

EmployeeCrudAngular

Employees:

Id	Name	Created Date		
1	Juan Perez	05/09/2024 00:00:00		
2	Carla Ruiz	05/09/2024 22:22:47		
3	Carlos Gomez	06/09/2024 09:16:50		
4	Joaquin Zarate	06/09/2024 09:19:37		
5	Luis Rodriguez	06/09/2024 09:23:31		
6	CATA	19/10/2024 15:59:22		

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

Variables

Search variables

API_URL_PROD = https://sa-crud-api-qa.azurewebsites.net/api/Employee

API_URL_QA = https://sa-crud-api-qa.azurewebsites.net/api/Employee

cnn-string-prod *****

cnn-string-qa *****

```
baseUrlBackend: 'http://$(backContainerInstanceNameQA).brazilsouth.azurecontainer.io'
baseUrlFrontEnd: 'http://$(frontContainerInstanceNameQA).brazilsouth.azurecontainer.io'
```

- 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.

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

← TP7

```
main / azure-pipelines.yml

- stage: DeployImagesToAppServicePROD
  displayName: 'Desplegar Imágenes de API y Front en Azure App Service (PROD)'
  dependsOn:
  - DeployImagesToAppServiceQA
  jobs:
  - deployment: DeployToProd
    displayName: 'Desplegar Imágenes de API y Front en Azure App Service (PROD)'
    environment:
      name: 'Production'
    strategy:
      runOnce:
        deploy:
          steps:
            - 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)'] | 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 $(WebAppApiNameContainersPROD) --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 $(WebAppApiNameContainersPROD) --resource-group $(ResourceGroupName) \
              --container-image-name $(acrLoginServer)/$(backImageName):$(backImageTag) \
              --container-registry-url https://$(acrLoginServer)/ \
              --container-registry-user $(acrName) \
              --container-registry-password $(az acr credential show --name $(acrName) --query "passwords[0].value" -o tsv)
            - # Establecer variables de entorno
              az webapp config appsettings set --name $(WebAppApiNameContainersPROD) --resource-group $(ResourceGroupName) \
              --settings ConnectionStrings:DefaultConnectionString=$(conn-string-prod)

335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373

43
44
45 > - stage: BuildAndTest...
151
152
153 > - stage: DockerBuildAndPush...
264 > - stage: DeployImagesToAppServiceQA...
331
332 > - stage: DeployImagesToAppServicePROD...
397
398
399
```

- Corremos el pipeline y vemos que cuando pasa la etapa de QA nos pide aprobación.

Stages Jobs

✓ Build and Test API an...

2 jobs completed 6m 12s

100% tests passed 4 artifacts

✓ Construir y Subir Imá...

1 job completed 1m 26s

✓ Desplegar Imagenes ...

1 job completed 3m 42s

⌚ Desplegar Imágenes ...

Waiting

1 check in progress

LinuxAppPlan01 | Aplicaciones

Buscar Actualizar

Filtrar por nombre Estado: Todo

Nombre	Tipo	Variante	Grupo de recursos	Estado
sa-crud-api-prod	Aplicación	app,linux,container	TPSingSw3UCC2024	En ejecución
sa-crud-api-qa	Aplicación	app,linux,container	TPSingSw3UCC2024	En ejecución
sa-crud-front-prod	Aplicación	app,linux,container	TPSingSw3UCC2024	En ejecución
sa-crud-front-qa	Aplicación	app,linux,container	TPSingSw3UCC2024	En ejecución

EmployeeCrudAngular

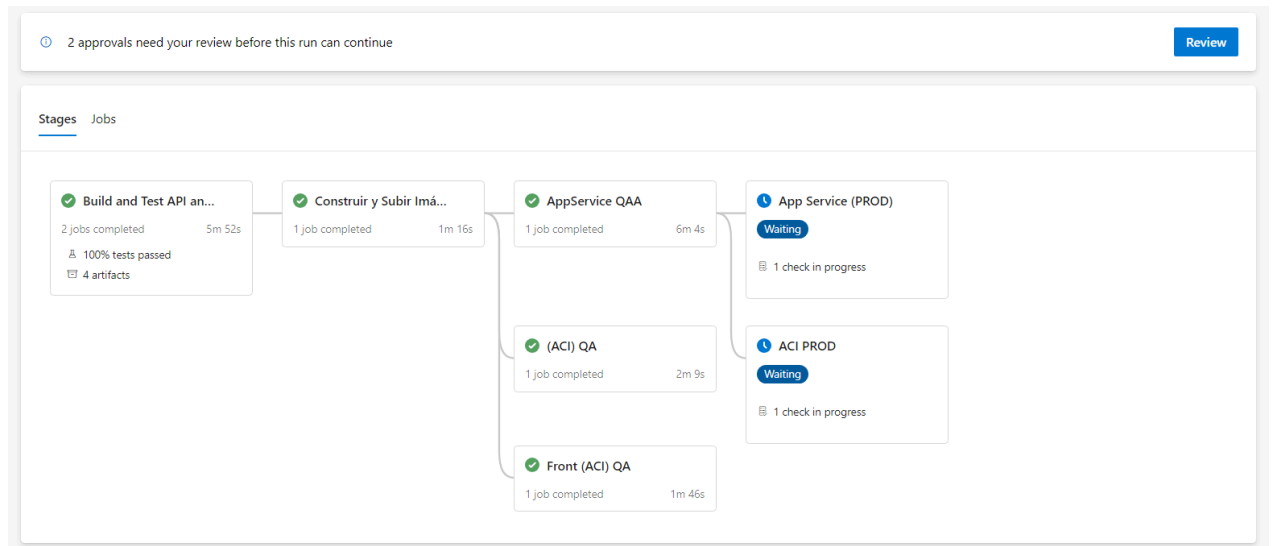
Employees:

[Add New Employee](#)

Id	Name	Created Date		
1	Juan Perez	05/09/2024 00:00:00		
2	Carla Ruiz	05/09/2024 22:22:47		
3	Carlos Gomez	06/09/2024 09:16:50		
4	Joaquin Zarate	06/09/2024 09:19:37		
5	Luis Rodriguez	06/09/2024 09:23:31		

5. Entregar un pipeline que incluya:

- A) Etapa Construcción y Pruebas Unitarias y Code Coverage Back y Front
- B) Construcción de Imágenes Docker y subida a ACR
- C) Deploy Back y Front en QA con pruebas de integración para Azure Web Apps
- D) Deploy Back y Front en QA con pruebas de integración para ACI
- E) Deploy Back y Front en QA con pruebas de integración para Azure Web Apps con Soporte para contenedores
- F) Aprobación manual de QA para los puntos C,D,E
- G) Deploy Back y Front en PROD para Azure Web Apps
- H) Deploy Back y Front en PROD para ACI
- I) Deploy Back y Front en PROD para Azure Web Apps con Soporte para contenedores



PIPELINE FINAL:

trigger:

- main

pool:

vmImage: 'windows-latest'

variables:

```
# AZURE VARIABLES
ConnectedServiceName: 'ServiceConnectionARM' #Por ejemplo
'ServiceConnectionARM'
acrLoginServer: 'saingsw3uccacr.azurecr.io' #Por ejemplo
'ascontainerregistry.azurecr.io'
acrName: 'SAIngSw3UCCACR'
backImageName: 'santiago-employee-crud-api' #Por ejemplo
'employee-crud-api'
frontImageName: 'santiago-employee-crud-front'
solution: '**/*.sln'
buildPlatform: 'Any CPU'
buildConfiguration: 'Release'
Configuration: 'Release'
frontPath: './EmployeeCrudAngular'
ResourceGroupName: 'TPSIngSw3UCC2024' #Por ejemplo 'TPS_INGSOFT3_UCC'
backContainerInstanceNameQA: 'santi-crud-api-qa' #Por ejemplo
'as-crud-api-qa'
backImageTag: 'latest'
container-cpu-api-qa: 1 #CPUS de nuestro container de QA
container-memory-api-qa: 1.5 #RAM de nuestro container de QA
cnn-string-qa:
'Server=tcp:sa-sql-server-1.database.windows.net,1433;Initial
Catalog=sa-sql-bd-01;Persist Security Info=False;User
```

```
ID=sqladmin;Password=4223551Santiago;MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate=False;Connection Timeout=30'
  cnn-string-prod:
'Server=tcp:sa-sql-server-1.database.windows.net,1433;Initial
Catalog=sa-sql-bd-01;Persist Security Info=False;User
ID=sqladmin;Password=4223551Santiago;MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate=False;Connection Timeout=30'
  frontContainerInstanceNameQA: 'santi-crud-front-qa'
  frontImageTag: 'latest'
  container-cpu-front-qa: 1
  container-memory-front-qa: 1.5
  baseUrlBackEnd:
'http://$(backContainerInstanceNameQA).canadaeast.azurecontainer.io'
  baseUrlFrontEnd:
'http://$(frontContainerInstanceNameQA).canadaeast.azurecontainer.io'
  backContainerInstanceNamePROD: 'santi-crud-api-prod'
  frontContainerInstanceNamePROD: 'santi-crud-front-prod'
  container-cpu-api-prod: 1
  container-memory-api-prod: 1.5
  container-cpu-front-prod: 1
  container-memory-front-prod: 1.5
  AppServicePlanLinux: 'LinuxAppPlan01'
  WebAppApiNameContainersQA: 'sa-crud-api-qa'
  WebAppFrontNameContainersQA: 'sa-crud-front-qa'
  baseUrlBackEndQA: 'https://$(WebAppApiNameContainersQA).azurewebsites.net'
  baseUrlFrontEndQA:
'https://$(WebAppFrontNameContainersQA).azurewebsites.net'
  WebAppApiNameContainersPROD: 'sa-crud-api-prod'
  WebAppFrontNameContainersPROD: 'sa-crud-front-prod'
```

```
stages:
```

- stage: BuildAndTest
 - displayName: "Build and Test API and Front"
 - jobs:
 - job: BuildDotnet
 - displayName: "Build and Test API"
 - pool:
 - vmImage: 'windows-latest'
 - steps:
 - checkout: self
 - fetchDepth: 0
 - task: DotNetCoreCLI@2
 - displayName: 'Restaurar paquetes NuGet'
 - inputs:
 - command: restore
 - projects: '\$(solution)'


```

- task: DotNetCoreCLI@2
  displayName: 'Ejecutar pruebas de la API'
  inputs:
    command: 'test'
    projects: '**/*.Tests.csproj'
    arguments: '--collect:"XPlat Code Coverage"'

- task: PublishCodeCoverageResults@2
  displayName: 'Publicar resultados de code coverage del back-end'
  inputs:
    summaryFileLocation: '$(Agent.TempDirectory)/**/*.cobertura.xml'
    failIfCoverageEmpty: false

- task: DotNetCoreCLI@2
  displayName: 'Compilar la API'
  inputs:
    command: build
    projects: '$(solution)'
    arguments: '--configuration $(configuration) --self-contained false'

- task: DotNetCoreCLI@2
  displayName: 'Publicar aplicación'
  inputs:
    command: publish
    publishWebProjects: True
    arguments: '--configuration $(buildConfiguration) --output
$(Build.ArtifactStagingDirectory)'
    zipAfterPublish: true

- task: PublishBuildArtifacts@1
  displayName: 'Publicar artefactos de compilación'
  inputs:
    PathToPublish: '$(Build.ArtifactStagingDirectory)'
    ArtifactName: 'drop-back'
    publishLocation: 'Container'

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

- job: BuildAngular
  displayName: "Build and Test Angular"
  pool:
    vmImage: 'ubuntu-latest'

```

```

steps:
- task: NodeTool@0
  displayName: 'Instalar Node.js'
  inputs:
    versionSpec: '22.x'

- script: npm install
  displayName: 'Instalar dependencias'
  workingDirectory: $(frontPath)

- script: npx ng test --karma-config=karma.conf.js --watch=false
  --browsers ChromeHeadless --code-coverage
  displayName: 'Ejecutar pruebas del front'
  workingDirectory: $(frontPath)
  continueOnError: true # Continue on test failure

- task: PublishCodeCoverageResults@2
  displayName: 'Publicar resultados de code coverage del front'
  inputs:
    summaryFileLocation: '$(frontPath)/coverage/lcov.info'
    failIfCoverageEmpty: false
  condition: always()

- task: PublishTestResults@2
  displayName: 'Publicar resultados de pruebas unitarias del front'
  inputs:
    testResultsFormat: 'JUnit'
    testResultsFiles: '$(frontPath)/test-results/test-results.xml'
    failTaskOnFailedTests: true
  condition: always()

- script: npm run build
  displayName: 'Compilar el proyecto Angular'
  workingDirectory: $(frontPath)

- task: PublishBuildArtifacts@1
  displayName: 'Publicar artefactos Angular'
  inputs:
    PathToPublish: '$(frontPath)/dist'
    ArtifactName: 'drop-front'

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

```

```

# #-----

```

```

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

- stage: DockerBuildAndPush
  displayName: 'Construir y Subir Imágenes Docker a ACR'
  dependsOn: BuildAndTest #NOMBRE DE NUESTRA ETAPA DE BUILD Y TEST
  jobs:
    - job: docker_build_and_push
      displayName: 'Construir y Subir Imágenes Docker a ACR'
      pool:
        vmImage: 'ubuntu-latest'

      steps:
        - checkout: self

#-----
# BUILD DOCKER BACK IMAGE Y PUSH A AZURE CONTAINER REGISTRY
#-----

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

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

- task: AzureCLI@2
  displayName: 'Iniciar Sesión en Azure Container Registry (ACR)'
  inputs:
    azureSubscription: '$(ConnectedServiceName)'
    scriptType: bash
    scriptLocation: inlineScript
    inlineScript: |
      az acr login --name $(acrLoginServer)

- task: ExtractFiles@1
  displayName: 'Descomprimir API en carpeta api'
  inputs:
    archiveFilePatterns:
      '$(Pipeline.Workspace)/drop-back/EmployeeCrudApi.zip'
    destinationFolder: '$(Pipeline.Workspace)/drop-back/api'

```

```

- script: ls -la $(Pipeline.Workspace)/drop-back
  displayName: 'Verificar archivos publicados'

- task: Docker@2
  displayName: 'Construir Imagen Docker para Back'
  inputs:
    command: build
    repository: $(acrLoginServer)/$(backImageName)
    dockerfile: $(Pipeline.Workspace)/dockerfile-back/dockerfile
    buildContext: $(Pipeline.Workspace)/drop-back
    tags: 'latest'

- task: Docker@2
  displayName: 'Subir Imagen Docker de Back a ACR'
  inputs:
    command: push
    repository: $(acrLoginServer)/$(backImageName)
    tags: 'latest'

#-----
# CONSTRUIR Y SUBIR IMAGEN DEL FRONTEND
#-----

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

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

- script: ls -la $(Pipeline.Workspace)/drop-front
  displayName: 'Verificar archivos del frontend'

- 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/employee-crud-angular/browser
        tags: 'latest'

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

#-----
### STAGE DEPLOY TO ACI QA
#-----

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

      steps:
        #-----
        # DEPLOY DOCKER BACK IMAGE A AZURE CONTAINER INSTANCES QA
        #-----

- 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: $(backImageTag)"
      echo "Connection String: $(cnn-string-qa)"

```

```
    az container delete --resource-group $(ResourceGroupName) --name  
$(backContainerInstanceNameQA) --yes
```

```
    az container create --resource-group $(ResourceGroupName) \  
    --name $(backContainerInstanceNameQA) \  
    --image $(acrLoginServer)/$(backImageName):$(backImageTag) \  
    --registry-login-server $(acrLoginServer) \  
    --registry-username $(acrName) \  
    --registry-password $(az acr credential show --name $(acrName)  
--query "passwords[0].value" -o tsv) \  
    --dns-name-label $(backContainerInstanceNameQA) \  
    --ports 80 \  
    --environment-variables  
ConnectionStrings__DefaultConnection="$(cnn-string-qa)" \  
    --restart-policy Always \  
    --cpu $(container-cpu-api-qa) \  
    --memory $(container-memory-api-qa)
```

```
#-----  
### STAGE DEPLOY TO ACI QA FOR FRONT  
#-----
```

```
- stage: DeployToACIQAFront  
  displayName: 'Front (ACI) QA'  
  dependsOn: DockerBuildAndPush # El contenedor del front depende de que las  
imágenes estén creadas
```

```
  jobs:  
    - job: deploy_to_aci_qa_front  
      displayName: 'Desplegar Front ACI QA'  
      pool:  
        vmImage: 'ubuntu-latest'
```

```
  steps:
```

```
#-----  
# DEPLOY DOCKER FRONT IMAGE A AZURE CONTAINER INSTANCES QA  
#-----
```

```
- 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: $(frontImageTag)"
```

```

    echo "API URL: $(API_URL)"

    az container delete --resource-group $(ResourceGroupName) --name
$(frontContainerInstanceNameQA) --yes

    az container create --resource-group $(ResourceGroupName) \
    --name $(frontContainerInstanceNameQA) \
    --image $(acrLoginServer)/$(frontImageName):$(frontImageTag) \
    --registry-login-server $(acrLoginServer) \
    --registry-username $(acrName) \
    --registry-password $(az acr credential show --name $(acrName)
--query "passwords[0].value" -o tsv) \
    --dns-name-label $(frontContainerInstanceNameQA) \
    --ports 80 \
    --environment-variables API_URL=$(API_URL) \
    --restart-policy Always \
    --cpu $(container-cpu-front-qa) \
    --memory $(container-memory-front-qa)

#-----
### STAGE DEPLOY TO AZURE APP SERVICE QA
#-----
- stage: DeployImagesToAppServiceQA
  displayName: 'AppService QAA'
  dependsOn:
    - BuildAndTest
    - 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)
        #-----
        - 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 '^1$';
then
        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-container-image-name "nginx" # Especifica una imagen temporal
para permitir la creación
        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) \
        --container-registry-user $(acrName) \
        --container-registry-password $(az acr credential show
--name $(acrName) --query "passwords[0].value" -o tsv)
        # Establecer variables de entorno
        az webapp config appsettings set --name
$(WebAppApiNameContainersQA) --resource-group $(ResourceGroupName) \
        --settings
ConnectionStrings__DefaultConnection="$(cnn-string-qa)" \

- task: AzureCLI@2
  displayName: 'Verificar y crear el recurso Azure App Service para
fron (QA) si no existe'
  inputs:
    azureSubscription: '$(ConnectedServiceName)'
    scriptType: 'bash'
    scriptLocation: 'inlineScript'
    inlineScript: |
      # Verificar si el App Service para la API ya existe
      if ! az webapp list --query
"[?name=='$(WebAppFrontNameContainersQA)' &&
resourceGroup=='$(ResourceGroupName)'] | length(@) > `0`"; then
        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 $(WebAppFrontNameContainersQA)
--deployment-container-image-name "nginx"
    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
$(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
    az webapp config appsettings set --name
$(WebAppFrontNameContainersQA) --resource-group $(ResourceGroupName) \
    --settings API_URL="$(API_URL_QA)"

#-----
#DEPLOY A PROD
#-----

- stage: DeployImagesToAppServicePROD
  displayName: 'App Service (PROD)'
  dependsOn: DeployImagesToAppServiceQA
  jobs:
    - deployment: DeployToProd
      displayName: 'Desplegar Imágenes de API y Front en Azure App Service
(PROD)'
      environment: 'Production'
      strategy:
        runOnce:
          deploy:
            steps:

#-----
# DEPLOY DOCKER BACK IMAGE A AZURE APP SERVICE PROD
#-----

- 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=='$(ResourceGroupName)'] | length(@)" -o tsv | grep -q '^1$';
then
                    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 $(WebAppApiNameContainersPROD)
--deployment-container-image-name "nginx"
                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
$(WebAppApiNameContainersPROD) --resource-group $(ResourceGroupName) \
--container-image-name
$(acrLoginServer)/$(backImageName):$(backImageTag) \
--container-registry-url https://$(acrLoginServer)/ \
--container-registry-user $(acrName) \
--container-registry-password $(az acr credential show
--name $(acrName) --query "passwords[0].value" -o tsv)

                # Establecer variables de entorno
                az webapp config appsettings set --name
$(WebAppApiNameContainersPROD) --resource-group $(ResourceGroupName) \
--settings
ConnectionStrings__DefaultConnection="$(conn-string-prod)"

- 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 la API ya existe
        if ! az webapp list --query
"['?name=='$(WebAppFrontNameContainersPROD)' &&
resourceGroup=='$(ResourceGroupName)'] | length(@)" -o tsv | grep -q '^1$';
then

        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 $(WebAppFrontNameContainersPROD)
--deployment-container-image-name "nginx"
        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
$(WebAppFrontNameContainersPROD) --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
        az webapp config appsettings set --name
$(WebAppFrontNameContainersPROD) --resource-group $(ResourceGroupName) \
--settings API_URL="$(API_URL_PROD)"

# #-----
# ### DEPLOY A PROD ACI
# #-----

- stage: DeployToACIPROD
  displayName: 'ACI PROD'
  dependsOn: DeployImagesToAppServiceQA
  jobs:
    - deployment: DeployToProd
      displayName: 'Desplegar en Azure Container Instances PROD'
      environment: 'Production'
      strategy:
        runOnce:
          deploy:
            steps:
              #-----

```

```

# DEPLOY DOCKER BACK IMAGE A AZURE CONTAINER INSTANCES PROD
#-----
- task: AzureCLI@2
  displayName: 'Desplegar Imagen Docker de Back en 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: $(backImageTag)"
      echo "API URL: $(cnn-string-prod)"

      az container delete --resource-group $(ResourceGroupName)
--name $(backContainerInstanceNamePROD) --yes

      az container create --resource-group $(ResourceGroupName)
\
      --name $(backContainerInstanceNamePROD) \
      --image
$(acrLoginServer)/$(backImageName):$(backImageTag) \
      --registry-login-server $(acrLoginServer) \
      --registry-username $(acrName) \
      --registry-password $(az acr credential show --name
$(acrName) --query "passwords[0].value" -o tsv) \
      --dns-name-label $(backContainerInstanceNamePROD) \
      --ports 80 \
      --environment-variables
ConnectionStrings__DefaultConnection="$(cnn-string-prod)" \
      --restart-policy Always \
      --cpu $(container-cpu-api-qa) \
      --memory $(container-memory-api-qa)

- task: AzureCLI@2
  displayName: 'Desplegar Imagen Docker de Front en 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: $(frontImageTag) "
        echo "API URL: $(API_URL_PROD) "

        az container delete --resource-group $(ResourceGroupName)
--name $(frontContainerInstanceNamePROD) --yes

        az container create --resource-group $(ResourceGroupName)
\
        --name $(frontContainerInstanceNamePROD) \
        --image
$(acrLoginServer)/$(frontImageName):$(frontImageTag) \
        --registry-login-server $(acrLoginServer) \
        --registry-username $(acrName) \
        --registry-password $(az acr credential show --name
$(acrName) --query "passwords[0].value" -o tsv) \
        --dns-name-label $(frontContainerInstanceNamePROD) \
        --ports 80 \
        --environment-variables API_URL=$(API_URL_PROD) \
        --restart-policy Always \
        --cpu $(container-cpu-front-prod) \
        --memory $(container-memory-front-prod)

```

Stages		Jobs		
Name	Status	Stage	Duration	
✓ Build and Test API	Success	Build and Test ...	⌚ 3m 14s	
✓ Build and Test Angular	Success	Build and Test ...	⌚ 1m 59s	
✓ Construir y Subir Imágenes Docker a ACR	Success	Construir y Sub...	⌚ 1m 8s	
✓ Desplegar en Azure Container Instances (ACI) QA	Success	(ACI) QA	⌚ 1m 53s	
✓ Desplegar Front ACI QA	Success	Front (ACI) QA	⌚ 1m 39s	
✓ Desplegar Imágenes de API y Front en Azure App Service (QA)	Success	AppService QAA	⌚ 5m 57s	
✓ Desplegar Imágenes de API y Front en Azure App Service (PROD)	Success	App Service (PR...	⌚ 4m 59s	
✓ Desplegar en Azure Container Instances PROD	Success	ACI PROD	⌚ 2m 57s	

← → ↻ No es seguro santi-crud-front-prod.canadaeast.azurecontainer.io

EmployeeCrudAngular

Employees:

Id	Name	Created Date		
1	Juan Perez	05/09/2024 00:00:00		
2	Carla Ruiz	05/09/2024 22:22:47		
3	Carlos Gomez	06/09/2024 09:16:50		
4	Joaquin Zarate	06/09/2024 09:19:37		
5	Luis Rodriguez	06/09/2024 09:23:31		

← → ↻ No es seguro santi-crud-front-qa.canadaeast.azurecontainer.io

EmployeeCrudAngular

Employees:

Id	Name	Created Date		
1	Juan Perez	05/09/2024 00:00:00		
2	Carla Ruiz	05/09/2024 22:22:47		
3	Carlos Gomez	06/09/2024 09:16:50		
4	Joaquin Zarate	06/09/2024 09:19:37		
5	Luis Rodriguez	06/09/2024 09:23:31		

← → ↺ No es seguro santi-crud-api-qa.canadaeast.azurecontainer.io/api/Employee/GetAll

Dar formato al texto

```
[{"id":1,"name":"Juan Perez","createdDate":"2024-09-05T00:00:00"}, {"id":2,"name":"Carla Ruiz","createdDate":"2024-09-05T22:22:47.440572"}, {"id":3,"name":"Carlos Gomez","createdDate":"2024-09-06T09:16:50.070943"}, {"id":4,"name":"Joaquin Zarate","createdDate":"2024-09-06T09:19:37.898716"}, {"id":5,"name":"Luis Rodriguez","createdDate":"2024-09-06T09:23:31.324434"}]
```

← → ↺ No es seguro santi-crud-api-prod.canadaeast.azurecontainer.io/api/Employee/GetAll

Dar formato al texto

```
[{"id":1,"name":"Juan Perez","createdDate":"2024-09-05T00:00:00"}, {"id":2,"name":"Carla Ruiz","createdDate":"2024-09-05T22:22:47.440572"}, {"id":3,"name":"Carlos Gomez","createdDate":"2024-09-06T09:16:50.070943"}, {"id":4,"name":"Joaquin Zarate","createdDate":"2024-09-06T09:19:37.898716"}, {"id":5,"name":"Luis Rodriguez","createdDate":"2024-09-06T09:23:31.324434"}]
```

Inicio > TPSIngSw3UCC2024 > LinuxAppPlan01

LinuxAppPlan01 | Aplicaciones

Plan de Linux

Buscar

Actualizar

Filtrar por nombre Estado: Todo

Nombre	Tipo	Variante	Grupo de recursos	Estado
sa-crud-api-prod	Aplicación	app.linux.container	TPSIngSw3UCC2024	En ejecución
sa-crud-api-qa	Aplicación	app.linux.container	TPSIngSw3UCC2024	En ejecución
sa-crud-front-prod	Aplicación	app.linux.container	TPSIngSw3UCC2024	En ejecución
sa-crud-front-qa	Aplicación	app.linux.container	TPSIngSw3UCC2024	En ejecución

← → ↺ sa-crud-front-prod.azurewebsites.net

EmployeeCrudAngular

Employees:

Id	Name	Created Date		
1	Juan Perez	05/09/2024 00:00:00		
2	Carla Ruiz	05/09/2024 22:22:47		
3	Carlos Gomez	06/09/2024 09:16:50		
4	Joaquin Zarate	06/09/2024 09:19:37		
5	Luis Rodriguez	06/09/2024 09:23:31		

← → ↺ No es seguro sa-crud-front-qa.azurewebsites.net

EmployeeCrudAngular

Employees:

Id	Name	Created Date		
1	Juan Perez	05/09/2024 00:00:00		
2	Carla Ruiz	05/09/2024 22:22:47		
3	Carlos Gomez	06/09/2024 09:16:50		
4	Joaquin Zarate	06/09/2024 09:19:37		
5	Luis Rodriguez	06/09/2024 09:23:31		

← → ↺ sa-crud-api-qa.azurewebsites.net/api/Employee/GetAll

Dar formato al texto

```
[{"id":1,"name":"Juan Perez","createdDate":"2024-09-05T00:00:00"}, {"id":2,"name":"Carla Ruiz","createdDate":"2024-09-05T22:22:47.440572"}, {"id":3,"name":"Carlos Gomez","createdDate":"2024-09-06T09:16:50.070943"}, {"id":4,"name":"Joaquin Zarate","createdDate":"2024-09-06T09:19:37.898716"}, {"id":5,"name":"Luis Rodriguez","createdDate":"2024-09-06T09:23:31.324434"}]
```

← → ↺ sa-crud-api-prod.azurewebsites.net/api/Employee/GetAll

Dar formato al texto

```
[{"id":1,"name":"Juan Perez","createdDate":"2024-09-05T00:00:00"}, {"id":2,"name":"Carla Ruiz","createdDate":"2024-09-05T22:22:47.440572"}, {"id":3,"name":"Carlos Gomez","createdDate":"2024-09-06T09:16:50.070943"}, {"id":4,"name":"Joaquin Zarate","createdDate":"2024-09-06T09:19:37.898716"}, {"id":5,"name":"Luis Rodriguez","createdDate":"2024-09-06T09:23:31.324434"}]
```