Laboratorio: CI/CD con GitHub, Kubernetes (AKS) y Azure (Serverless) para un Web Service Node.js + Express + TypeScript.

Harold Carrillo, Andrés Díaz, Paola Feng, Isaac Gutiérrez, y Tonny Ortiz

I. Comandos de configuración para el webservice de nodejs, express y typescript

Inicializar proyecto con package.json por defecto

```
npm init -y
```

Instalar TypeScript y tipos para express (además de configuración para TS)

```
npm i -D typescript @types/node @types/express @tsconfig/node22
```

Crear archivo de configuración TypeScript

```
npx tsc --init
```

Instalar ejecutor TypeScript para desarrollo (alternativa a nodemon)

```
npm i -D tsx
```

Instala prettier para formatear código

```
npm i -D prettier eslint-config-prettier
```

Instalar linter para convenciones de código (con dependencias)

```
npm i -D eslint typescript-eslint @eslint/js eslint-plugin-perfectionist
```

Instalar jest para pruebas unitarias

```
npm i -D jest ts-jest @types/jest ts-node
```

Instalar plugin de jest para ESLint

```
npm i -D eslint-plugin-jest
```

Instalar supertest para realizar pruebas HTTP sobre el servidor

```
npm i -D supertest @types/supertest
```

Nota: La configuración inicial del proyecto se hizo basado en esta guía

Drouin, G. (2024, December 25). *Node.js 2025 Guide: How to Set Up Express.js with TypeScript, ESLint, and Prettier*. Medium. https://medium.com/@gabrieldrouin/node-js-2025-guide-how-to-setup-express-js-with-typescript-eslint-and-prettier-b342cd21c30c447

II. Implementación de pruebas con supertest

Se utilizó un file de pruebas para describir todos los test cases del web service implementado. Al ser un CRUD simple, se implemento las pruebas de get, post, put, delete. Las pruebas simulan al servidor en operación para probar el comportamiento con el mock data.

```
import request from 'supertest';

import app from '../src/index.js';
import { cleanData } from '../src/utils/jsonUtils.js';

describe('API Endpoints', () => {
  cleanData();

  it('health check endpoint working', async () => {
    const response = await request(app).get('/health');
    expect(response.status).toBe(200);
    expect(response.body).toEqual({ ok: true });
  });

  it('Create curso', async () => {
    const response = await request(app).post('/').send({
      creditos: 4,
      nombre: 'Cálculo I',
      sigla: 'MA1001',
    });
    expect(response.status).toBe(201);
    expect(response.body).toHaveProperty('id');
    expect(response.body.sigla).toBe('MA1001');
    expect(response.body.nombre).toBe('Cálculo I');
    expect(response.body.creditos).toBe(4);
  });

  it('Get all cursos', async () => {
    const response = await request(app).get('/');
    console.log(response.body);
    expect(response.status).toBe(200);
    expect(Array.isArray(response.body)).toBe(true);
    expect(response.body.length).toBeGreaterThan(0);
  });

  it('Get curso by id', async () => {
    const newCurso = await request(app).post('/').send({
      creditos: 4,
      nombre: 'Calculo 2',
      sigla: 'MA1002',
    });
    const cursoId = newCurso.body.id;
    const response = await request(app).get(`/${cursoId}`);
    expect(response.status).toBe(200);
    expect(response.body.id).toBe(cursoId);
    expect(response.body.sigla).toBe('MA1002');
    expect(response.body.nombre).toBe('Calculo 2');
    expect(response.body.creditos).toBe(4);
```

III. Continuous integration con Github Action

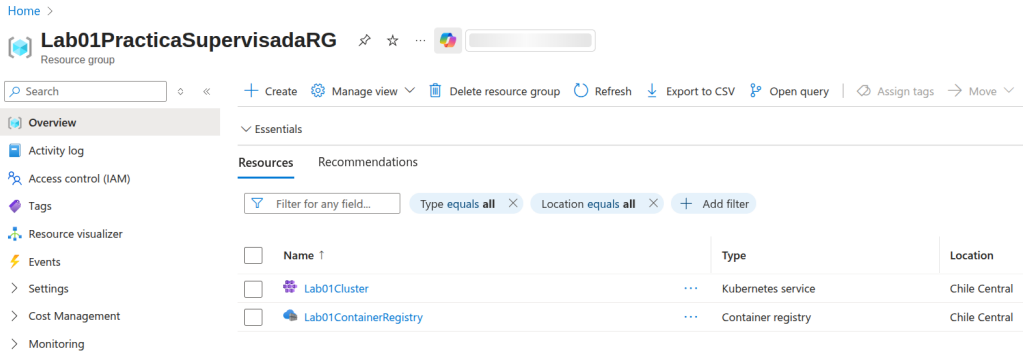
Se realizó un workflow para poder ejecutar diferentes scripts que se encuentran en el package json. El workflow ejecuta el chequeo de TypeScript, corre el linter para resolver problemas, formatea el código con prettier y corre las pruebas para hacer un check de calidad del código.

```yaml
name: CI

on:
  push:
    branches:
      - main

jobs:
  quality-checks:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Setup Node.js
        uses: actions/setup-node@v4
        with:
          node-version: '22'
          cache: 'npm'

      - name: Install dependencies
        run: npm ci

      - name: Run TypeScript type checking
        run: npm run type-check

      - name: Run ESLint and fix
        run: npm run lint:fix

      - name: Format with Prettier
        run: npm run format

      - name: Check for remaining ESLint errors
        run: npm run lint

      - name: Check for remaining formatting issues
        run: npm run format:check

      - name: Run tests
        run: npm run test
```

IV. Creación de servicios de Azure

Para esta parte se mostrará como se crearon los recursos necesarios en Azure junto con las configuraciones necesarias para que todo el pipeline de CI/CD funcione correctamente

A. Creacion de recursos



Se proceden a crear dentro de la región Chile Central el Resource Group, Azure Container Registry y el Azure Kubernetes Service.

Una vez creados los recursos se procede a configurar lo necesario desde el Azure CLI.

B. Creación de App registration para OIDC

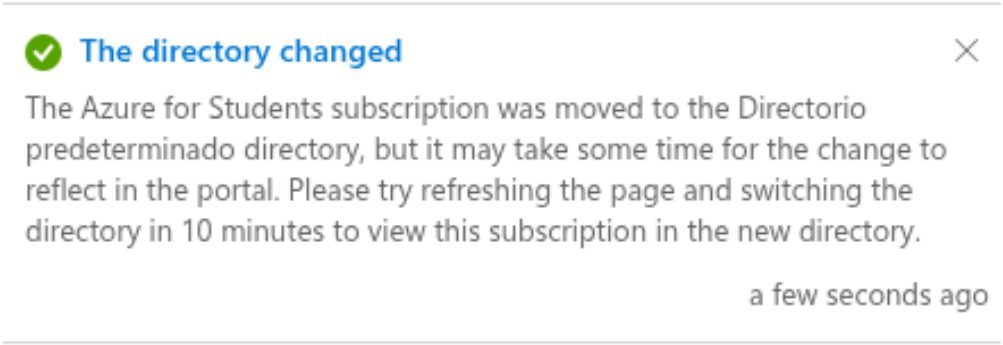
Para crearlo, utilizamos el siguiente script:

```
   # Variables a usar (ajstalas a tu entorno)
SUBSCRIPTION_ID="00099c50-1fa9-4420-9a16-ccecce110185"
RESOURCE_GROUP="Lab01PracticaSupervisadaRG"
APP_NAME="lab01-be-app"

# Selecciona la suscripcin
az account set --subscription $SUBSCRIPTION_ID

# Crea la app registration
az ad app create --display-name $APP_NAME
```

Por problemas de políticas, encontramos el siguiente error:

```
   az ad app create --display-name $APP_NAME
Directory permission is needed for the current user to register the application. For how to
    configure, please refer 'https://learn.microsoft.com/azure/azure-resource-manager/resource-
group-create-service-principal-portal'. Original error: Insufficient privileges to complete
    the operation.
```

Esto hace que sea necesario mover la suscripción a un tenant de otra cuenta personal para permitir activar los privilegios de que nuestro usuario pueda hacer app registrations. Se toma esta decisión debido a que de otra forma el administrador de licencias de la UCR es quien debe darnos los permisos en el tenant de la cuenta UCR.



Una vez hecho esto si se permite con normalidad el comando.

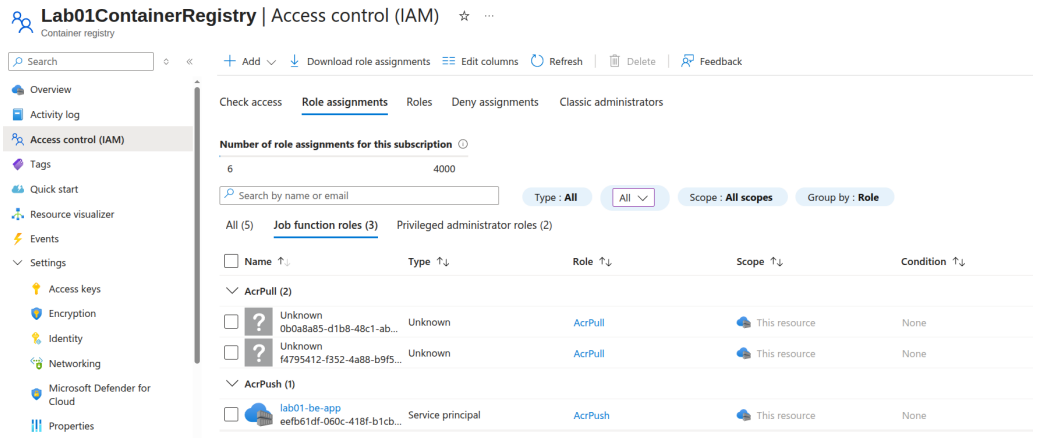
```
Your Cloud Shell session will be ephemeral so no files or system changes will persist beyond your current session.
tonny [ ~ ]$ SUBSCRIPTION_ID="00099c50-1fa9-4420-9a16-ccecce110185"
tonny [ ~ ]$ RESOURCE_GROUP="Lab01PracticaSupervisadaRG"
tonny [ ~ ]$ APP_NAME="lab01-be-app"
tonny [ ~ ]$ az account set --subscription $SUBSCRIPTION_ID
tonny [ ~ ]$ az ad app create --display-name $APP_NAME
{
  "@odata.context": "https://graph.microsoft.com/v1.0/$metadata#applications/$entity",
  "addIns": [],
  "api": {
    "acceptMappedClaims": null,
    "knownClientApplications": [],
    "oauth2PermissionScopes": [],
    "preAuthorizedApplications": [],
    "requestedAccessTokenVersion": null
  },
  "appId": "eefb61df-060c-418f-b1cb-6ad11c91fbdf",
  "appRoles": [],
  "applicationTemplateId": null,
  "certification": null,
  "createdDateTime": "2025-09-15T04:15:37.7276463Z",
  "defaultRedirectUri": null,
  "deletedDateTime": null,
  "description": null,
  "disabledByMicrosoftStatus": null,
```

```
  "servicePrincipalLockConfiguration": null,
  "signInAudience": "AzureADMyOrg",
  "spa": {
    "redirectUris": []
  },
  "tags": [],
  "tokenEncryptionKeyId": null,
  "uniqueName": null,
  "verifiedPublisher": {
    "addedDateTime": null,
    "displayName": null,
    "verifiedPublisherId": null
  },
  "web": {
    "homePageUrl": null,
    "implicitGrantSettings": {
      "enableAccessTokenIssuance": false,
      "enableIdTokenIssuance": false
    },
    "logoutUrl": null,
    "redirectUriSettings": [],
    "redirectUris": []
  }
}
tonny [ ~ ]$
```

https://learn.microsoft.com/en-us/azure/aks/use-oidc-issuer

Posteriormente se procede a dar permisos de Accesos Control:

- Permisos de AcrPush al App registration que creamos antes
- Permisos de AcrPull al AKS para que pueda hacer pull de las imagenes en el ACR.



Y se procede a configurar los workflows en la carpeta .github/workflow del backend. Adjunto podemos leer el código para el action que se encargara de hacer push de las imagenes al Container Registry y de ahí hacer deployment al Kubernetes Service.

Además para que esto funcione se deben definir los secrets en la configuración del repo.

Estos secrets no son privados ya que corresponden a los nombres de resource group, el nombre del cluster, el nombre del Conteiner registry, entre otros.

```
name: CD Staging

on:
  push:
    branches:
      - main
  workflow_dispatch:
```

```
permissions:
  id-token: write
  contents: read

jobs:
  build-and-deploy:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      # 1. Login to Azure using OIDC
      - name: Azure Login (OIDC)
        uses: azure/login@v2
        with:
          client-id: ${{ secrets.AZURE_CLIENT_ID }}
          tenant-id: ${{ secrets.AZURE_TENANT_ID }}
          subscription-id: ${{ secrets.AZURE_SUBSCRIPTION_ID }}

      # 2. Login to Azure Container Registry
      - name: Azure Container Registry login (OIDC)
        run: |
          TOKEN=$(az acr login --name ${{ secrets.ACR_NAME }} --expose-token --output tsv --query
              accessToken)
          echo $TOKEN | docker login ${{ secrets.ACR_NAME }}.azurecr.io \
            --username 00000000-0000-0000-0000-000000000000 \
            --password-stdin


      # 3. Build and push image to ACR
      - name: Build and push image to ACR
        run: |
          IMAGE_TAG=${{ secrets.ACR_NAME }}.azurecr.io/${{ secrets.IMAGE_NAME }}:${{ github.sha }}
          docker build -f lab-be.dockerfile -t $IMAGE_TAG .
          docker push $IMAGE_TAG
          echo "IMAGE_TAG=$IMAGE_TAG" >> $GITHUB_ENV

      # 4. Get AKS credentials
      - name: Get AKS credentials
        run: |
          az aks get-credentials \
            --name ${{ secrets.AKS_NAME }} \
            --overwrite-existing \
            --resource-group ${{ secrets.RESOURCE_GROUP }}

      # 5. Deploy to AKS
      - name: Deploy to AKS
        run: |
          # Replace placeholder and apply manifests
          sed "s|PLACEHOLDER_IMAGE|${{ env.IMAGE_TAG }}|" k8s/deployment.yaml | kubectl apply -f -
          kubectl apply -f k8s/service.yaml
          kubectl apply -f k8s/ingress.yaml
          kubectl rollout status deployment/lab-be
```

Una vez hecho push del workflow vemos la ejecución desde el actions.

Lamentablemente el job que falla es al hacer deployment al AKS.

Se logra hacer build del backend y push correctamente de la imagen al Container registry, sin embargo parece que algo falla desde el Kubernetes Service al intentar hacer pull de la imagen en el container registry. Más pruebas serán necesarias para resolver el problema.

Aqui podemos ver que si se hizo push correctamente de las imagenes cada vez que se hace un nuevo commit