DEPARTAMENTO:	**CIENCIAS DE LA COMPUTACIÓN**	**CARRERA:**	**TECNOLOGÍAS DE LA INFORMACIÓN**		
ASIGNATURA:	Arquitectura de software	**PERÍODO LECTIVO:**	202551	**NIVEL:**	8
DOCENTE:	Armando Ortiz	**NRC:**	30798	**PRÁCTICA N°:**	6
LABORATORIO DONDE SE DESARROLLARÁ LA PRÁCTICA		LAB 3			
TEMA DE LA PRÁCTICA:	Implementación de un flujo CI/CD				

INTRODUCCIÓN:

CI/CD (Integración Continua y Despliegue Continuo)

En el desarrollo moderno de software, la velocidad y la calidad son factores críticos. Para lograr ambos de forma sostenida, las organizaciones adoptan prácticas de automatización que reduzcan errores humanos y aceleren la entrega de funcionalidades. En este contexto surge CI/CD, un conjunto de prácticas fundamentales dentro de DevOps. CI/CD significa Continuous Integration / Continuous Deployment (o Delivery) y se refiere a la automatización del proceso que va desde un cambio en el código fuente hasta su validación y puesta en producción.

La **Integración Continua** es la práctica de integrar frecuentemente los cambios de código de los desarrolladores en un repositorio central (por ejemplo, GitHub).
Cada integración activa automáticamente un proceso que:

- Compila el proyecto
- Ejecuta pruebas automáticas
- Verifica que el código cumple reglas básicas de calidad

El objetivo de la CI es "detectar errores lo antes posible", evitando que los problemas se acumulen y se descubran recién en etapas tardías del proyecto.
CI responde a la pregunta**:** *¿el código que acabo de subir funciona correctamente?*

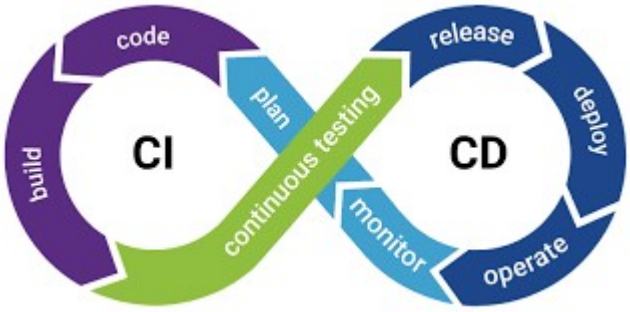
El **Despliegue Continuo** es el paso siguiente a la CI. Una vez que el código ha sido validado automáticamente, el sistema puede:

- Preparar el artefacto final (por ejemplo, una imagen Docker)
- Desplegarlo automáticamente en un servidor o plataforma en la nube
- Actualizar la aplicación sin intervención manual

Dependiendo del nivel de automatización, el CD puede ser:

- **Continuous Delivery**: el despliegue queda listo, pero requiere aprobación humana.
- **Continuous Deployment**: el despliegue se realiza automáticamente.

CD responde a la pregunta: *¿cómo hago que los cambios lleguen rápida y confiablemente al entorno donde se usa la aplicación?*



OBJETIVOS:

- Crear una API REST simple y versionada.
- Contenerizar la API con Docker.
- Configurar un pipeline CI (compilación + pruebas + artefacto).
- Configurar un pipeline CD (despliegue automático a servidor).
- Verificar despliegue y observabilidad mínima (health + logs).

MATERIALES:

REACTIVOS:	**INSUMOS:**
No aplica	No aplica

EQUIPOS:

- Computador: HP ProDesk 600 G2 DM (x64), Windows 10, Procesador Intel® Core™ i7-6700T 2.8 GHz, 8GB RAM, 480GB SSD, Intel HD Graphics 530.
- Periféricos: Monitor, teclado, mouse.

MUESTRA:
No aplica

INSTRUCCIONES:

CI/CD básico para construir y desplegar automáticamente una API

Parte A — Preparación del proyecto (API)

A1. Crear API en .NET
En una carpeta vacía:

```
dotnet new webapi -n Demo.CICD.Api
cd Demo.CICD.Api
dotnet build
dotnet test
```

Si tu plantilla crea Demo.CICD.Api.http, mantenlo.

A2. Endpoints mínimos obligatorios
Editar el archivo Program.cs y dejarlo con la siguiente estructura mínima:

```
var builder = WebApplication.CreateBuilder(args);

builder.Services.AddEndpointsApiExplorer();
```

```
builder.Services.AddSwaggerGen();

var app = builder.Build();

app.UseSwagger();
app.UseSwaggerUI();

app.MapGet("/api/v1/ping", () =>
        Results.Ok(new { ok = true, ts = DateTime.UtcNow })
);

app.MapGet("/health", () => Results.Ok("OK"));

app.MapGet("/version", () => Results.Ok(new
{
    service = "Demo.CICD.Api",
    env = app.Environment.EnvironmentName,
    build = Environment.GetEnvironmentVariable("GIT_SHA") ?? "local"
}));

app.Run();
```

Parte B — Dockerización

B1. Crear Dockerfile

En la raíz del proyecto (Demo.CICD.Api/) crea un archivo Dockerfile:

```
# Build
FROM mcr.microsoft.com/dotnet/sdk:10.0 AS build
WORKDIR /src
COPY . .
RUN dotnet restore
RUN dotnet publish -c Release -o /app/publish

# Runtime
FROM mcr.microsoft.com/dotnet/aspnet:10.0 AS runtime
WORKDIR /app
EXPOSE 8080
ENV ASPNETCORE_URLS=http://+:8080

ARG GIT_SHA=local
ENV GIT_SHA=$GIT_SHA

COPY --from=build /app/publish .
ENTRYPOINT ["dotnet", "Demo.CICD.Api.dll"]
```

B2. Crear docker-compose (para el servidor)

En la raíz del repositorio, crear el archivo docker-compose.yml:

```
services:
  api:
    image: ghcr.io/USUARIO_GH/demo-cicd-api:latest
    container_name: demo-cicd-api
    ports:
      - "8080:8080"
    environment:
      - ASPNETCORE_ENVIRONMENT=Development
      - GIT_SHA=local
```

El valor USUARIO_GH se reemplazará luego por el usuario de GitHub del grupo.

Parte C — Repositorio y CI (Build + Tests)

C1. Subir a GitHub

1. Crea repo: demo-cicd-api
2. Sube el código:

```
git init
git add .
git commit -m "init api"
git branch -M main
git remote add origin https://github.com/USUARIO/demo-cicd-api.git
git push -u origin main
```

C2. Configurar GitHub Container Registry (GHCR)

La idea: el pipeline construye imagen y la publica en GHCR.

Crea archivo: .github/workflows/ci-cd.yml

```
name: CI-CD API

on:
  push:
    branches: [ "main" ]

permissions:
  contents: read
  packages: write

jobs:
  build_test_and_publish_image:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout
        uses: actions/checkout@v4

      - name: Setup .NET
        uses: actions/setup-dotnet@v4
        with:
```

```
        dotnet-version: "10.0.x"

    - name: Restore
      run: dotnet restore

    - name: Build
      run: dotnet build -c Release --no-restore

    - name: Test
      run: dotnet test -c Release --no-build

    - name: Log in to GHCR (GitHub Container Registry)
      uses: docker/login-action@v3
      with:
        registry: ghcr.io
        username: ${{ github.actor }}
        password: ${{ secrets.GITHUB_TOKEN }}

    - name: Build and push Docker image
      run: |
        IMAGE=ghcr.io/${{ github.repository_owner }}/demo-cicd-api:latest
        docker build -t $IMAGE ./Demo.CICD.Api
        docker push $IMAGE
```

Con esto ya tienes CI: compila, prueba y publica imagen en GitHub Container Registry.

PARTE D — CD local (Despliegue Continuo sin VPS)

D1. Requisitos locales del estudiante

- Windows 10/11
- Docker Desktop instalado
- WSL 2 habilitado
- Sesión iniciada en GitHub

Verificar Docker:
docker --version
docker compose version

D2. Autenticarse en GitHub Container Registry
docker login ghcr.io
Usuario: usuario de GitHub
Contraseña: Personal Access Token

D3. Despliegue local de la API
Desde la raíz del proyecto:
docker compose pull
docker compose up -d

D4. Verificación del despliegue
Probar en el navegador o Postman:

- http://localhost:8080/health
- http://localhost:8080/api/v1/ping
- http://localhost:8080/version

El campo build debe cambiar cuando haya nuevos commits.

ACTIVIDADES POR DESARROLLAR:

Ejecutar Despliegue Continuo (CD) local

- Autenticarse en GitHub Container Registry.
- Descargar la imagen generada por el pipeline CI.
- Desplegar la API localmente usando Docker Compose.
- Verificar que la API se encuentre en ejecución.

Generar un cambio y validar el ciclo CI/CD

- Realizar un cambio controlado en el código (por ejemplo, modificar el mensaje o agregar un campo en el endpoint /version).
- Subir el cambio al repositorio (git commit y git push).
- Verificar que el pipeline CI se ejecute nuevamente de forma automática.
- Actualizar la aplicación local ejecutando el despliegue.
- Confirmar que el cambio se refleja en la respuesta del endpoint /version.

Analizar y documentar el flujo CI/CD

- Describir brevemente el flujo completo desde el cambio en el código hasta la ejecución de la aplicación.
- Identificar el rol de GitHub Actions en la Integración Continua.
- Explicar cómo se realiza el Despliegue Continuo en el entorno local.

RESULTADOS OBTENIDOS:

1. Realizar el informe utilizando el formato general de informes.
2. Incluir en el informe las capturas de pantalla que evidencien su trabajo.
3. Subir a la plataforma el informe en formato "PDF".

CONCLUSIONES:

Incluir al menos 2 conclusiones propias.

RECOMENDACIONES:

Incluir al menos 2 recomendaciones propias.

FIRMAS

Nombre y Apellido: Armando Ortiz **DOCENTE**	**Nombre y Apellido:** Paulo Galarza **COORDINADOR DE ÁREA DE CONOCIMIENTO**	**Nombre y Apellido:** Javier Cevallos **COORDINADOR/JEFE DE LABORATORIO**