TEST AUTOMATION ENGINEER – FORMACIÓN INTEGRAL

{desafío} latam_

CURSO:

TEST AUTOMATION ENGINEER – FORMACIÓN INTEGRAL

- Módulo 1: FUNDAMENTOS DEL TESTING DE SOFTWARE
- Módulo 2: CONTROL DE VERSIONES Y ENTORNOS DE DESARROLLO
- Módulo 3: FUNDAMENTOS DE PROGRAMACIÓN APLICADOS AL TESTING (JAVASCRIPT)
- Módulo 4: CARACTERISTICAS DE CYPRESS – AUTOMATIZACIÓN WEB I
- Módulo 5: CARACTERISTICAS DE CYPRESS – AUTOMATIZACIÓN WEB II
- Módulo 6: PRUEBAS DE APIS CON POSTMAN Y SUPERTEST

{desafío} latam_

CURSO:

TEST AUTOMATION ENGINEER – FORMACIÓN INTEGRAL

- Módulo 7: HERRAMIENTAS DE PLAYWRIGHT Y PRUEBAS CON MÚLTIPLES NAVEGADORES
- Módulo 8: DESARROLLO GUIADO EN EL COMPORTAMIENTO (BDD) CON CUCUMBER.JS
- Módulo 9: HERRAMIENTAS DE AUTOMATIZACIÓN MÓVIL CON APPIUM
- **Módulo 10: HERRAMIENTAS DE INTEGRACIÓN DE PRUEBAS EN CI/CD**
- Módulo 11: HERRAMIENTAS DE DOCKER, ENTORNOS VIRTUALIZADOS Y PRUEBAS EN LA NUBE
- HERRAMIENTAS DE AUTOMATIZACIÓN DE UN FLUJO COMPLETO WEB + API + CI/CD

Te encuentras aquí

{desafío} latam_

Módulo 10: Herramientas de integración de pruebas en ci/cd.

{desafío} latam_

OBJETIVO ESPECÍFICO DEL MÓDULO

- EXPLICAR HERRAMIENTAS DE INTEGRACIÓN DE PRUEBAS EN CI/CD, DE ACUERDO A LAS APLICACIONES WEB, MÓVILES Y APIS.

{desafío} latam_

¿Por qué cree que es cada vez más necesario automatizar los procesos de prueba, construcción y despliegue en proyectos de desarrollo de software?

{desafío} latam_

¿QUÉ ES CI/CD?

- **CI (Integración Continua):**
- Proceso de integrar cambios de código en un repositorio compartido varias veces al día.
- Cada integración dispara pruebas automáticas para detectar errores rápidamente.

- **CD (Entrega/Despliegue Continuo):**
- Automatiza el empaquetado, validación y despliegue de la aplicación.
- Permite entregar versiones estables al entorno de producción sin intervención manual.

- **Beneficios:**
- Menos errores en producción
- Feedback rápido sobre cambios
- Ahorro de tiempo en procesos repetitivos

- CI/CD es la columna vertebral de la automatización moderna: mejora calidad, velocidad y confianza en cada entrega.

AUTOMATIZACIÓN CON HERRAMIENTAS CI

- **GitHub Actions:**
- Usa archivos .yml dentro de .github/workflows/
- Ejecuta flujos en eventos como push, pull_request, schedule, etc.

- **Ejemplo básico:**

```yaml
name: Run Tests
on: [push]
jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - run: npm install
      - run: npm test
```

{desafío} latam_

- **GitLab CI:**
- Configuración en .gitlab-ci.yml
- Define stages (build, test, deploy) y jobs por entorno
- Compatible con runners locales o en la nube

- Ambas herramientas permiten ejecutar pruebas Cypress, Playwright, Jest, Supertest o Postman en entornos controlados y auditables.

CONTROL Y FLEXIBILIDAD EN LOS PIPELINES

- **Variables de entorno (env):**
- Guardan valores como tokens, URLs o claves API
- Se accede con ${{ secrets.MY_TOKEN }} (GitHub) o $VARIABLE (GitLab)

- **Condicionales:**

- **Ejecutar pasos solo si se cumplen condiciones:**

```
if: github.ref == 'refs/heads/main'
```

- **Orquestación de tareas:**
- Definir el orden: build → test → deploy
- Ejecutar en paralelo o secuencial
- Permite dividir tareas pesadas y reutilizar pasos

- Un pipeline bien orquestado garantiza ejecución eficiente y controlada, incluso ante múltiples ramas y entornos.

MANEJO DE FALLOS Y ALERTAS AUTOMÁTICAS

- Los pipelines generan logs detallados ante errores (fallas de test, instalación, permisos)
- Es clave leer los mensajes para identificar el paso exacto que falló

- **Notificaciones automáticas:**
- Enviar alertas por Slack, Discord, email, etc.

- **GitHub:** con actions/slack, actions/send-email

- **GitLab:** nativo o usando curl para integraciones

- **Buenas prácticas:**
- Mostrar errores amigables
- Registrar artefactos (screenshots, videos de fallas)
- Reintentar pruebas inestables (retry)

- La retroalimentación inmediata permite al equipo corregir errores sin demoras, manteniendo la calidad del producto.

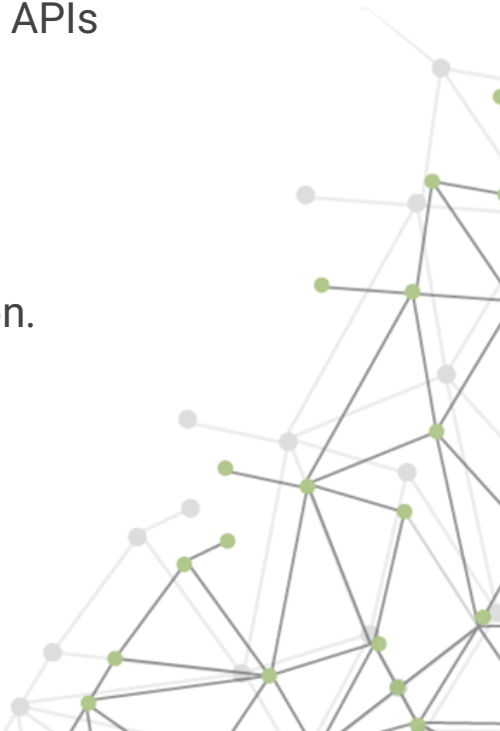
UN PIPELINE COMPLETO DE AUTOMATIZACIÓN

- Estructura típica de CI/CD en QA Automation:

1. **Test:** Ejecutar pruebas unitarias (Jest), E2E (Cypress/Playwright), y APIs (Supertest/Postman).

2. **Build:** Compilar o empaquetar la aplicación (React, Node.js, etc.).

3. **Deploy:** Enviar la app a un servidor de pruebas, staging o producción.



- **Ejemplo de archivo ci-cd.yml para GitHub Actions:**

```
name: CI/CD Pipeline
on:
  push:
   job:name: ↵ Teš Test
    runs uses: actions/cheıkoot@3
    steps:
      name 🔧 Test
       runs:on: ıbuncl
       -run: npm install
       -run: npm test
    build: npnc cypressrun
   jobs:
    name: 🔨🌲 Build
     needs: build
     sets: actıonśons/checkout@v3
      nues: test
       needs: build
       run: echo "Desplegan
    text:  "✅ Desplüegue exiıso QA ..."
   env:
   deploy: naı 🚀 Deploy
    name: ⚠ Deley
     uses: slackapi/s lack-github-action@v1.25.0
     types:
      run: echo "Desplegando
          aplicación a QA..."
     env:
     slack: secrets.SLACK_WEBHOOK_URL }}
```

{desafío} latam_

No olvide desarrollar los ejercicios que contiene el Módulo…

{desafío} latam_

¿Cómo contribuyen los pipelines automatizados, el uso de variables de entorno y la notificación de fallos al aseguramiento de calidad y la entrega continua en entornos de desarrollo reales?

{desafío} latam_

Éxito en la evaluación parcial y en la Prueba Final...

{desafío}
latam_

Academia de
talentos digitales