

IaC con AWS + Github Actions

AWS Cdk

Cómo lanzar recursos o una Arquitectura?



consola

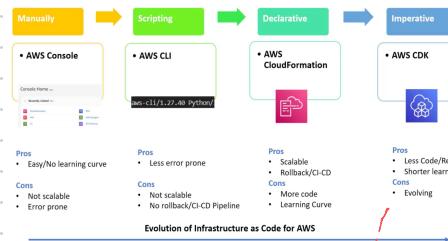
```
version: '2010-09-09'
resources:
  myLambdaFunction:
    Type: AWS::Lambda::Function
    Properties:
      Handler: index.handler
      Runtime: nodejs14.x
      Role: !GetAtt LambdaExecutionRole.Arn
    Code:
      S3Key: my-lambda-code.zip
  myLambdaFunction:
    Type: AWS::Lambda::Function
    Properties:
      Handler: index.handler
      MemorySize: 256
      Timeout: 10
      Policies:
        - LambdaExecutionPolicy
      Version: 2023-10-17
      TracingConfig:
        ExplicitHandler: true
      LogGroup:
        LogType: CloudWatchLogs
        LogGroupName: /aws/lambda/my-lambda-function
        LogStreamName: log-stream
  myLambdaFunctionArn: !GetAtt myLambdaFunction.Arn
  myLambdaFunctionLogGroup: !GetAtt myLambdaFunction.LogGroup
  myLambdaFunctionLogStream: !GetAtt myLambdaFunction.LogStream
```

CloudFormation

```
version: '2010-09-09'
resources:
  myLambdaFunction:
    type: AWS::Lambda::Function
    properties:
      handler: index.handler
      runtime: nodejs14.x
      memory_size: 256
      timeout: 10
      policies:
        - LambdaExecutionPolicy
      version: 2023-10-17
      tracing_config:
        explicit_handler: true
      log_group:
        type: CloudWatchLogs
        log_group_name: /aws/lambda/my-lambda-function
        log_stream_name: log-stream
  myLambdaFunctionArn: !GetAtt myLambdaFunction.Arn
  myLambdaFunctionLogGroup: !GetAtt myLambdaFunction.LogGroup
  myLambdaFunctionLogStream: !GetAtt myLambdaFunction.LogStream
```

CDK

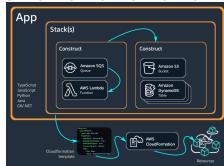
Evolución IaC en AWS



Yventajas
IaC

- Autonomía ✓
- Reproducibilidad ✓
- Yentorno ✓
- Herramientas ✓
- N lenguajes: JS, Python, Go...
- Definir lógica
- Integración en IDEs mediante Plugins
- Reutilización de componentes
- Open Source
- Varios niveles de abstracción (1/4/12/13)

Estructura y Despliegue con



CDK

- Contiene 1 o más constructs.
- Representa un Block de Cloud Formation
- Tiene un ciclo de vida indep
- extiende de la clase Construct

Constructs

- Representa la pieza mínima de construcción de un CDK
- Representa 1 o más componentes de CloudFormation
- y su configuración
- Se pueden importar constructs de CDK de terceros o crear los tuyos propios
- **Construct levels**
 - representan recursos de CloudFormation
 - L1: No proporcionan lógica o config por defecto. Toda la lógica debe ser codificada explícitamente. Llevan el prefijo "CDK"

- L2: Abstracciones de recursos de AWS
- Representa 1 o más recursos
- Proporcionan lógica por defecto

- L3: Llamados a "patrones"
- L3: Contienen un conjunto de constructs y lógica a facilitar el despliegue de infraestructura compleja y específica

Github Actions

CI/CD → Automatización del proceso de integración de código en un repositorio de forma frecuente, seguimiento de tests, y despliegues automáticos.

GitHub Actions es una funcionalidad de GitHub que permite automatizar tareas dentro del ciclo de vida de desarrollo de software, como pruebas, despliegues o integraciones continuas (CI/CD), directamente desde el repositorio. Se basa en workflows definidos en archivos YAML que se ejecutan al producirse eventos como push, pull request, etc.

Actions de la comunidad son acciones reutilizables creadas por otros desarrolladores y publicadas en GitHub. Pueden incluir tareas comunes como instalar dependencias, ejecutar tests o desplegar a servicios como AWS o Firebase.

Ejemplo de configuración de un pipeline simple (CI):

Archivo .github/workflows/ci.yml:

```
name: CI Pipeline
on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]
jobs:
  build-and-test:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v3 # Acción de la comunidad
      - name: Set up Node.js
        uses: actions/setup-node@v3 # Otra acción de la comunidad
        with:
          node-version: '18'
      - name: Install dependencies
        run: npm install
      - name: Run tests
        run: npm test
```

Este pipeline se ejecuta al hacer push o pull request a la rama main, instala Node.js, las dependencias del proyecto y corre los tests.

QA y Autom de Casos de Prueba

Conceptos

QA Quality Assurance

- Naturaleza: más preventiva y orientada al control de procesos
- Todos los canales y así dentro de un ciclo de calidad → poco fiable

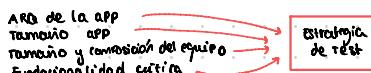
QC Quality Control

- Control de calidad orientado al producto final

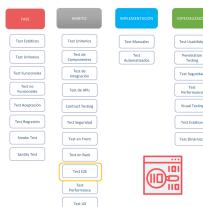
Testing Quality Control

- Todo lo relacionado con el testing (manual o autom)

Estrategias y Tipos de Test



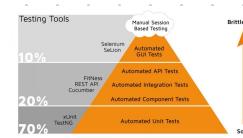
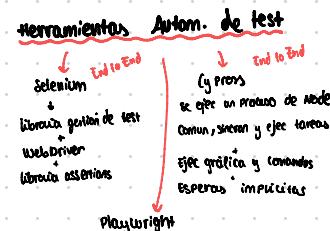
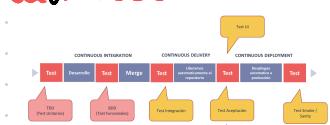
Clasificaciones de los tipos de test



Especificación de los test



Integración Q&A en el Ciclo desarrollo



POM: Page Object Model

Cuando hacemos E2E testing → Imp. localizar tus elem para hacer una acción

click botones rellenar un cuadro de texto ...

Problema...

Debemos tener en cuenta un punto muy importante al automatizar: la UI (Interfaz de usuario) puede sufrir cambios. Si nuestros test manipulan directamente los elementos de la página, estos serán muy frágiles y requerirán de mucho más mantenimiento.

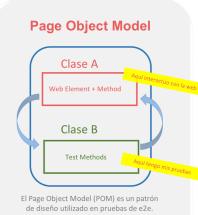
Solución:

El patrón POM encapsula elementos y el comportamiento de las páginas, o una parte de estas, lo que nos permite escribir tests y manipular elementos de la página sin tener que lidiar con el HTML.

Ejemplo:

Por cada página podemos crear una clase con los componentes (input, checkboxes, radios, etc) de la página definidos como variables y sus funciones/métodos. Esto permitirá reutilizar el acceso a los elementos si por ejemplo por alguna razón un elemento cambia de nombre, solo lo cambiaremos en la clase de la página en lugar de cambiarlo en todos los tests.

Posteriormente definiremos otra clase donde estaran implementados los propios test y que se validará de las clases anteriores para focalizar, acceder e interactuar y ejecutar el test.



LoginPage
User
Password
EnvrMessage
LoginButton

Ejercicios: Gherkin y Cypress

Gherkin (como herramienta de automatización de pruebas):

- Qué es:

Gherkin no es una herramienta en si, sino un lenguaje de especificación legible por humanos utilizado para definir casos de prueba en formato Given-When-Then. Se usa junto con herramientas como Cucumber, Behave o SpecFlow para ejecutar pruebas automatizadas basadas en esos escenarios.

- Conceptos principales:

Facilita la colaboración entre técnicos y no técnicos (por ejemplo, QA, negocio y desarrollo), alineando los tests con los requisitos funcionales.

- Ejemplo de escenario en Gherkin:

```
gherkin
    Scenario: Log in
        Given the user logs in successfully
        When the user is on the login page
        Then the user enters valid credentials
        Then the user should be redirected to the dashboard
```

- Uso en CI/CD:
Se ejecuta junto con pruebas automatizadas en el pipeline para validar que el software cumple con los requisitos de negocio definidos.

¿Cuál usar en CI/CD y por qué?

- Usa Gherkin (con Cucumber/Behave...):
- Necesita colaboración entre equipos técnicos y no técnicos.
- Queremos los tests sean ejecutables y orientados al negocio.
- Trabaja con BDD (Behavior-Driven Development).

- Usa Cypress:
- Trabaja en validar el comportamiento de la interfaz de usuario en tiempo real.
- Trabaja principalmente con aplicaciones web modernas.
- Preferes una herramienta rápida, con feedback visual y fácil de integrar.

Cypress (como herramienta de automatización de pruebas):

- Qué es:

Cypress es una herramienta de testing end-to-end para aplicaciones web. Permite simular la interacción de un usuario en un navegador real y verificar que la interfaz funciona correctamente.

- Características clave:

- Fácil de configurar.
- Muy buena integración con JavaScript/TypeScript.
- Ideal para pruebas UI rápidas y confiables.
- Soporte visual e inspección del DOM durante los tests.

- Ejemplo básico de test en Cypress:

```
describe('Login', () => {
  it('should log in successfully', () => {
    cy.visit('https://www.example.com')
    cy.get('#username').type('user@example.com')
    cy.get('#password').type('password123')
    cy.get('#login').click()
    cy.url().should('include', '/dashboard')
  })
})
```

- Uso en CI/CD:

Cypress se ejecuta fácilmente en pipelines (por ejemplo, GitHub Actions o GitLab CI) para ejecutar pruebas E2E automáticamente con cada punto o despliegue.

Angular

Conceptos Básicos

Web Application Framework

Single Page Application

open-source

TypeScript based

Características Principales

- Component-based ARQ: Cada componente →
- Data binding
 - Sync entre el modelo → Vista
 - Cada modif hecha → Vista → Actualiza → Modelo
 - Cada modif hecha → Modelo → Actualiza → Vista
- Dependency injection

Es un patrón de diseño en el que las dependencias (por ejemplo, servicios, objetos necesarios para una clase) no se crean dentro de la clase, sino que se inyectan desde fuera, generalmente por el framework (en este caso, Angular).

Ejemplo simple:
En lugar de hacer esto dentro de una clase:

```
ts                                ⚡ Copiar
const myService = new MyService();
```

Angular lo hace así:

```
ts                                ⚡ Copiar
constructor(private myService: MyService) { }
```

Angular se encarga de crear y proporcionar MyService automáticamente.

El framework debería manejar dependencias, no los programadores.

ROUTING

En Angular, el routing es el sistema que permite navegar entre diferentes páginas o vistas de una aplicación de una sola página (SPA, Single Page Application), sin recargar la página completa.

Angular CLI

Instalación básica

Prerrequisitos < node.js < Angular

- npm install -g @angular/cli
- ng new app-name

APP COMPONENT · L ↗ CSS
HTML TS SPEC.TS

Por ejemplo:

```
bash                                ⚡ Copiar
ng new tienda-angular
```

Angular CLI te hará algunas preguntas:

- ¿Quieres usar routing? (di que sí siquieres navegar entre páginas)
- ¿Qué estilo quieres usar? (CSS, SCSS, etc.)

3. Moverte al directorio del proyecto y ejecutarlo

```
bash                                ⚡ Copiar
cd tienda-angular
ng serve
```

Esto levanta el servidor local. Abres tu navegador en <http://localhost:4200> y verás tu nueva app.

4. ¿Qué son los App Components que se generan?

Al crear el proyecto, Angular crea una estructura básica, con los siguientes componentes principales:

app.module.ts

Es el módulo raíz de tu aplicación.

- Aquí se declaran todos los componentes.
- Importa otros módulos necesarios (como FormsModule, HttpClientModule, etc.)
- Define qué componente se carga al inicio (bootstrap).

app.component.ts

Es el componente raíz, el primero que se renderiza.

Contiene:

- La lógica del componente (class AppComponent)
- Un selector que se usa en el HTML principal (index.html)
- Referencia a sus HTML y CSS.

```
ts                                ⚡ Copiar
```

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'tienda-angular';
}
```

app.component.html

Es la vista del componente raíz. Aquí puedes usar otros componentes, poner navegación (router-outlet), etc.

app.component.css

Estilos locales del componente raíz. Solo aplican a este componente.

Estructura de carpetas básica:

```
src/
  app/
    app.component.ts    --> Componente raíz
    app.component.html  --> Vista del componente raíz
    app.component.css   --> Estilos
    app.module.ts       --> Módulo raíz
```