**Proyecto de Organización de Archivos**

**Proyecto de Investigación y Prueba de Concepto (PoC): Testing de APIs con Docker y Automatización**

**Matthias Paniagua Rodriguez**

**Jonathan Morales Barrientos**

**Erick Martinez Brenes**

**Bryan Frazee west**

**Universidad Politécnica Internacional**

**Profesor Jose Javier Mata Guerrero**

**1 Cuatrimestre 2025**

**Introducción**

El presente proyecto tiene como finalidad demostrar la implementación de una solución automatizada para la prueba de APIs REST utilizando herramientas modernas como Postman, Newman, Docker, WordPress y Visual Studio Code. Esta Prueba de Concepto o (PoC) busca integrar una infraestructura sencilla pero robusta que permita validar de manera automática y visual el funcionamiento de distintos endpoints de APIs públicas. A través de la contenerización, se logra que la solución sea fácilmente replicable, portátil y escalable.

## **Objetivo del Proyecto**

Construir una prueba de concepto automatizada y contenerizada que permita probar múltiples APIs REST de forma visual (WordPress) y automatizada (Newman), usando Docker para orquestar todo el entorno.

## **Herramientas Utilizadas**

- Docker  
- Docker Compose  
- Postman  
- Newman  
- WordPress  
- Visual Studio Code  
- PowerShell

-Navegador de preferencia

**¿Qué es docker-compose?**

El comando docker-compose up --build es uno de los más utilizados para levantar entornos de desarrollo o pruebas que están definidos en un archivo docker-compose.yml. Este comando permite construir las imágenes necesarias y levantar múltiples contenedores de manera coordinada, conectándolos automáticamente dentro de una red virtual interna. A continuación, se detalla su funcionamiento "tras bambalinas" y se explica cómo actúa en cada paso.

**¿Qué hace docker-compose up --build?**

Este comando tiene dos acciones principales:

1. **--build**: Construye las imágenes necesarias a partir de los Dockerfile indicados en los servicios.
2. **up**: Levanta todos los servicios definidos en el archivo YAML, creando y ejecutando los contenedores.

**Proceso Detallado Paso a Paso**

**1. Lectura del archivo docker-compose.yml**

Docker Compose primero analiza y carga la estructura del archivo docker-compose.yml. En este archivo se definen:

* Los **servicios** (contenedores) que componen la aplicación.
* Las **rutas de construcción** de imágenes.
* Los **puertos** expuestos.
* Las **variables de entorno** requeridas.
* Las **redes internas** para conexión entre contenedores.
* Los **volúmenes** que permiten persistencia de datos.

Ejemplo básico de servicios definidos:

services:  wordpress:    image: wordpress:latest    ports:      - "8080:80"    environment:      WORDPRESS\_DB\_HOST: db:3306  db:    image: mysql:5.7    environment:      MYSQL\_ROOT\_PASSWORD: root  newman:    build: ./newman    command: ["newman", "run", "coleccion\_catfacts\_poc.json"]

**2. Construcción de imágenes (--build)**

Para cualquier servicio que incluya la propiedad build, Docker Compose:

* Ingresa a la ruta definida (por ejemplo, ./newman).
* Busca el archivo Dockerfile.
* Ejecuta paso a paso las instrucciones del Dockerfile.

Ejemplo de Dockerfile para Newman:

FROM node:18RUN npm install -g newmanWORKDIR /appCOPY coleccion\_catfacts\_poc.json .CMD ["newman", "run", "coleccion\_catfacts\_poc.json"]

Resultado: una imagen lista para ejecutar automáticamente las pruebas.

**3. Creación de contenedores**

Una vez las imágenes están listas, Docker Compose crea los contenedores. Cada contenedor es una instancia aislada del servicio definido, conectada a la red del proyecto y configurada con sus variables, puertos y volúmenes correspondientes.

**4. Configuración de redes y volúmenes**

Docker Compose crea una red interna por defecto que permite a los servicios comunicarse entre sí mediante sus nombres definidos. Por ejemplo, WordPress puede conectarse a db gracias a esa red.

También se crean y montan volúmenes para conservar datos persistentes, como los archivos de configuración de WordPress o las bases de datos de MySQL.

**5. Inicio de servicios**

Finalmente, todos los contenedores son iniciados. En el caso de esta PoC:

* WordPress arranca en el puerto 8080.
* MySQL queda disponible internamente para WordPress.
* El contenedor Newman ejecuta la colección de pruebas y finaliza tras imprimir los resultados en consola.

**Aplicación en la Prueba de Concepto**

En el proyecto de Prueba de Concepto (PoC), el comando docker-compose up --build permite:

* Construir la imagen personalizada de pruebas con Newman.
* Levantar simultáneamente WordPress y MySQL para la interfaz web.
* Ejecutar de manera automática todas las pruebas sobre las APIs seleccionadas.

Este enfoque garantiza un entorno reproducible, limpio y automatizado que puede ejecutarse en cualquier máquina con Docker instalado.

**¿Qué es una API?**

Una API (Application Programming Interface, o Interfaz de Programación de Aplicaciones) es un conjunto de definiciones y protocolos que permite que dos aplicaciones se comuniquen entre sí. En el contexto web, una API REST expone recursos a través de URLs y permite interactuar con ellos mediante métodos HTTP como GET, POST, PUT y DELETE. Las APIs son fundamentales para el desarrollo moderno, ya que permiten integrar servicios, automatizar procesos y habilitar la comunicación entre sistemas heterogéneos.

Por ejemplo, una API puede permitir que una aplicación móvil acceda a una base de datos de usuarios, que un sistema de pagos procese transacciones o que una página web consulte información meteorológica. En este proyecto se han utilizado APIs públicas que simulan comportamientos reales y permiten aplicar pruebas sin necesidad de infraestructura propietaria.

**Herramientas Utilizadas**

A continuación, se describen las herramientas utilizadas y su propósito específico dentro del proyecto:

* **Docker**: Utilizado para contenerizar los servicios necesarios para la PoC, permitiendo una fácil replicación y despliegue del entorno. Se implementaron tres contenedores: uno para WordPress, uno para la base de datos MySQL, y uno personalizado que ejecuta Newman para las pruebas automatizadas.
* **Postman**: Herramienta utilizada para diseñar y validar manualmente la colección de pruebas de las APIs seleccionadas. Permite crear peticiones, organizar colecciones y agregar scripts de validación.
* **Newman**: Utilidad de línea de comandos desarrollada por Postman que permite ejecutar colecciones de pruebas de manera automatizada. Ideal para integraciones en flujos CI/CD o ejecución dentro de contenedores Docker.
* **WordPress**: Sistema de gestión de contenido utilizado para construir una interfaz visual desde la cual los usuarios pueden probar las APIs de forma interactiva utilizando JavaScript y fetch.
* **Visual Studio Code**: Entorno de desarrollo utilizado para escribir, editar y organizar los archivos del proyecto, incluyendo archivos YAML, HTML, JSON y scripts de configuración Docker. Su integración con Git permite también el versionado del código.
* **PowerShell**: Consola de comandos en Windows utilizada para ejecutar peticiones HTTP manuales, así como para manejar los servicios de Docker y validar la operación de la PoC.

**Pasos Detallados de la Implementación**

**1. Estructuración del Proyecto**

Se definió una estructura de carpetas clara para separar la lógica de pruebas automatizadas del entorno de publicación web:

**2. Diseño de la Colección de Pruebas**

Se desarrolló una colección en Postman que contempla tres APIs públicas:

* **JSONPlaceholder**: API de prueba para operaciones CRUD simuladas. Se prueba la obtención de una publicación.
* **CatFacts API**: API que retorna curiosidades aleatorias sobre gatos. Reemplazó a BoredAPI debido a su mayor estabilidad.
* **ReqRes**: API que simula la creación de usuarios mediante una petición POST.

Cada una de estas pruebas incluye validaciones básicas como verificación de código de estado HTTP y existencia de propiedades clave en la respuesta JSON.

**3. Automatización con Newman y Docker**

Se construyó un contenedor personalizado que instala Newman a partir de una imagen base de Node.js. Este contenedor se configura para ejecutar automáticamente la colección de pruebas al iniciarse. El contenedor se integra dentro del archivo docker-compose.yml para levantarse junto con los servicios de WordPress y MySQL.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer screen

AI-generated content may be incorrect.

**4. Interfaz Visual con WordPress**

Dentro del entorno WordPress, se creó una página que permite probar las mismas APIs desde el navegador. Esto se logra utilizando bloques de HTML personalizados en los que se inserta código JavaScript que realiza llamadas a las APIs usando fetch() y muestra las respuestas en la misma página.

**5. Validación Manual con PowerShell**

Además de las pruebas automatizadas, se validaron los endpoints mediante comandos en PowerShell utilizando Invoke-RestMethod, lo que permite verificar las respuestas de manera directa desde la consola.

A screenshot of a computer

AI-generated content may be incorrect.

**Explicación Técnica de docker-compose up –build**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Desafíos Encontrados y Soluciones Implementadas**

| **Desafío** | **Solución Implementada** |
| --- | --- |
| BoredAPI presentaba intermitencias de servicio | Se reemplazó por CatFacts API, la cual mostró mayor estabilidad y tiempos de respuesta |
| WordPress no cargaba en localhost:8080 | Se ajustó la configuración del puerto en el archivo docker-compose.yml |
| Docker no encontraba el archivo de configuración | Se aseguró la ejecución desde la carpeta raíz del proyecto donde reside el YAML |
| Errores visuales en scripts HTML de WordPress | Se confirmó que los errores eran visuales (subrayado) y no afectaban la ejecución real |

**Código y Configuración**

El código del proyecto está preparado para alojarse en un repositorio GitHub. Se incluye:

* Archivo docker-compose.yml con la definición de los tres servicios.
* Dockerfile que instala y configura Newman.
* Archivo .json con la colección de pruebas Postman.
* Archivo .dockerignore para evitar copiar archivos innecesarios.
* Página HTML personalizada dentro de WordPress que actúa como cliente visual de prueba.

## Paso a Paso Detallado para la implementación

### 1. Se debe crear la estructura de las carpetas

mkdir api-testing-poc  
cd api-testing-poc  
mkdir newman

### 2. Se debe crear el archivo docker-compose.yml

version: '3.9'  
services:  
 wordpress:  
 image: wordpress:latest  
 ports:  
 - "8080:80"  
 environment:  
 WORDPRESS\_DB\_HOST: db:3306  
 WORDPRESS\_DB\_USER: root  
 WORDPRESS\_DB\_PASSWORD: root  
 WORDPRESS\_DB\_NAME: wp\_db  
 volumes:  
 - wp\_data:/var/www/html  
 depends\_on:  
 - db  
  
 db:  
 image: mysql:5.7  
 environment:  
 MYSQL\_DATABASE: wp\_db  
 MYSQL\_ROOT\_PASSWORD: root  
 volumes:  
 - db\_data:/var/lib/mysql  
  
 newman:  
 build: ./newman  
 depends\_on:  
 - wordpress  
 command: ["newman", "run", "coleccion\_catfacts\_poc.json"]  
  
volumes:  
 wp\_data:  
 db\_data:

### 3. Se debe crear el Dockerfile para Newman

FROM node:18  
RUN npm install -g newman  
WORKDIR /app  
COPY coleccion\_catfacts\_poc.json .  
CMD ["newman", "run", "coleccion\_catfacts\_poc.json"]

### 4. Creamos el Archivo .dockerignore

node\_modules  
npm-debug.log

### 5. Crear la colección de Postman

Diseñar pruebas en Postman y exportarlas como JSON con un nombre como: coleccion\_catfacts\_poc.json. El mismo se debe guardar en /newman.

### 6. Crear página en WordPress para probar APIs

1. Ir a http://localhost:8080 una vez que el entorno esté levantado.

A screenshot of a computer

AI-generated content may be incorrect.

2. Crear el código en html para visualizar de manera grafica las pruebas.  
3. Crear una nueva página desde el panel de WordPress.  
4. Insertar un bloque de HTML personalizado.  
5. Pegar el código HTML con fetch() para consumir las APIs y mostrar resultados dinámicamente.  
6. Usar JavaScript dentro de <script> para hacer llamadas HTTP desde la interfaz.

### 7. Levantar todo el entorno con el siguiente comando dentro de la terminal

docker-compose up --build

Este comando construye la imagen personalizada de Newman y levanta los servicios de WordPress y MySQL. Newman ejecuta automáticamente la colección al iniciar.

### 8. Validar manualmente en consola (PowerShell)

Invoke-RestMethod -Uri "https://jsonplaceholder.typicode.com/posts/1" -Method GET  
Invoke-RestMethod -Uri "https://catfact.ninja/fact" -Method GET

### 9. Comando para detener los servicios

docker-compose down

**Instrucciones para Ejecutar la PoC en Cualquier Máquina**

1. Clonar el repositorio o descomprimir el proyecto en una carpeta local.
2. Asegurarse de tener instalado Docker y Docker Compose.
3. Abrir una terminal y ubicarse en la carpeta raíz del proyecto.
4. Ejecutar el comando:

docker-compose up --build

1. Acceder a WordPress en <http://localhost:8080> para interactuar con la interfaz visual.
2. Observar en consola la ejecución de pruebas automatizadas de Newman.

**Usos Potenciales de Esta Prueba de Concepto**

Esta PoC puede ser útil como base para múltiples contextos en desarrollo de software:

* **Proyectos educativos**: Enseñar a estudiantes los fundamentos de APIs, pruebas automáticas y contenedores.
* **Validación de APIs internas o externas**: Adaptar la colección para probar APIs propias antes de ser publicadas.
* **Integración en pipelines de CI/CD**: Incluir Newman en un flujo de integración continua para validar endpoints después de cada despliegue.
* **Generación de documentación viva**: Ampliar la interfaz web para que sirva como demostración de uso para equipos de desarrollo o clientes.
* **Prototipado rápido**: Verificar que endpoints de APIs públicas o privadas respondan de forma esperada durante la fase de prototipo.
* **Testing visual para usuarios no técnicos**: Proporcionar a usuarios sin conocimientos técnicos una interfaz gráfica simple para validar el comportamiento de una API.

**Información del archivo JSON usado para el proyecto**

"info": {

"name": "API Testing PoC - JSONPlaceholder, CatFacts, ReqRes", (nombres de las API a usar)

"schema": "https://schema.getpostman.com/json/collection/v2.1.0/collection.json"

(esquema Postman que define el formato)

}

**Lista de peticiones (item)**

Este array contiene 3 objetos, cada uno es una petición individual.

**1. JSONPlaceholder - GET Post**

"url": {

"raw": "https://jsonplaceholder.typicode.com/posts/1", (pagina principal del placeholder)

"protocol": "https",

"host": ["jsonplaceholder", "typicode", "com"],

"path": ["posts", "1"] (el 1 indica el tipo de post que muestra la API según lo que indique el

Usuario, si pone un 2 el contenido es diferente y asi)

}

**Resumen**: Hace una petición GET a https://jsonplaceholder.typicode.com/posts/1 para obtener un post de prueba.

**Tests definidos**:

pm.test("Status code is 200", function () {

pm.response.to.have.status(200); (verifica que el estado de HTTP sea 200 que es OK)

});

pm.test("Response has userId and title", function () {

var jsonData = pm.response.json(); (Respuesta que enviamos desde el sitio web)

pm.expect(jsonData).to.have.property('userId'); (Verifica que la respuesta tenga el ‘userId’

pm.expect(jsonData).to.have.property('title'); (Verificar que la respuesta tenga el ‘title’

});

**2. CatFacts - Random Fact**

"url": {

"raw": "https://catfact.ninja/fact", (pagina de donde salen los factores)

"protocol": "https",

"host": ["catfact", "ninja"],

"path": ["fact"] (muestra un factorial aleatorio)

}

**Resumen**: Petición GET para obtener un dato curioso sobre gatos de https://catfact.ninja/fact.

**Tests definidos**:

pm.test("Status code is 200", function () {

pm.response.to.have.status(200); (verifica que el estado de HTTP sea 200 que es OK)

});

pm.test("Response has a cat fact", function () { (El nombre de la función que retrae un dato)

var jsonData = pm.response.json(); (Respuesta que enviamos desde el sitio web)

pm.expect(jsonData).to.have.property('fact'); (Verifica que la respuesta tenga la propiedad

fact)

});

**3. ReqRes - Create User**

"method": "POST",

"body": {

"mode": "raw",

"raw": "{\"name\": \"morpheus\", \"job\": \"leader\"}" (Datos que se esperan por defecto en la página web para crear un JSON de un usuario con un Id)

}

**Encabezado**:

{ "key": "Content-Type", "value": "application/json" } (básicamente dice que se envia un JSON)

**Tests definidos**:

pm.test("Status code is 201", function () {

pm.response.to.have.status(201); (Verifica que el estado de HTTP sea 201 que es Created)

});

pm.test("Response has id and createdAt", function () {

var jsonData = pm.response.json(); (Respuesta que enviamos desde el sitio web)

pm.expect(jsonData).to.have.property('id');

pm.expect(jsonData).to.have.property('createdAt');

});

(Verifica que la respuesta enviada tenga

las propiedades de ‘id’ (identificador de nuevo usuario) y ‘createdAt’ (hora de creación))

Este JSON enseña a:

* Definir y organizar **colecciones de pruebas API** en Postman.
* Crear pruebas automáticas para validar que las respuestas sean correctas.
* Trabajar con métodos **GET y POST**.
* Comprobar el contenido de respuestas JSON usando JavaScript.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

**Conclusiones**

En resumen, este proyecto ha demostrado cómo las herramientas modernas pueden ser integradas para crear una solución robusta y automatizada de prueba de APIs, ofreciendo una experiencia visual amigable y asegurando que la infraestructura sea fácilmente replicable y escalable. Está listo para ser presentado y replicado por cualquier compañero, proporcionando una base sólida para proyectos más complejos en el futuro.