BUAP

FACULTAD DE CIENCIAS DE LA COMPUTACIÓN

ING. EN TECNOLOGIAS
DE LA INFORMACIÓN

Tecnologías Web

“Tarea 4 ;Despliegue Profesional de aplicaciones web:Hosing,Docker y CI/CD”

PROFESOR: LUIS YAEL MENDEZ SANCHEZ

ALUMNO:LOZANO CRUZ JOSÉ ÁNGEL 202243427

Primavera 2025

Contenido

Introducción

En este presente documento se investigó acerca de tecnologías web las cuales serian en principal: Hosting, Docker y CI/CD , lo que me hizo recordar que hoy en día estas herramientas y servicios que usamos en internet cada vez son más comunes en la rutina diaria pero sin saber mucho del trans fondo de estas , ejemplos de estas tecnologías son las redes sociales o las plataformas de streaming , esto en cuestión de entretenimiento pero también en el apartado de compras en línea y con el tema de distancia lo que impulso la pandemia del COVID surgieron aplicaciones de trabajo y escolares , y bueno todo esto funciona gracias al despliego web.

Aunque en su mayoría indagar en estos temas requiere un conocimiento medio de tecnologías , todo se resume en manejar una aplicación en línea desde un dispositivo. Ya indagando más en el tema hay mucho más trabajo y aplicaciones detrás de todo esto que es la innovación de nuestra era que simplemente hacer una aplicación, en esta presenta investigación pude aprender temas importante del hosting por qué es necesario el dominio, como se manejan y conectan los servidores y como es que todo esto puede lograr que una aplicación o pagina puede subirse a lo que se llama red o internet y que funcione correctamente.

También es importante reconocer la gran ayuda que representan las aplicaciones actuales en todos estos procesos que nos orientan y podrían llegar a poder automatizar todo esto , uno de mis programas de los cuales me pareció más interesante fue Docker el cual permite simular entornos de trabajo en este caso como tipo servidores sin la necesidad de acudir a uno ya más profesional , esto con la finalidad de mantener funciones importantes de todas estas aplicaciones en línea , incluso con datos que mayormente necesitan más seguridad y eficiencia como lo serian aplicaciones de banco etc. Toda esta investigación me demostró la importancia de aplicar los métodos correctos y bien establecidos a programas

,aplicaciones o páginas, en cuanto a despliegue y funcionamiento me refiero. Esto también ayudo a comprender la importancia de esta conexión en estos casos el back o hablando más general de la programación con el frontend. A lo largo de esta investigación se explicará lo mejor posible todas estas tecnologías y los temas de los que se hablaron aquí y como se relacionan entre sí para lograr una aplicación web, bien estructurada y funcional

1.- ¿Qué es el hosting o alojamiento web?

Definición: El alojamiento web es un servicio en la nube en el que un proveedor de servicios almacena todos los archivos que componen un sitio web en un servidor y lo hace accesible en Internet.

Tipos de hosting: Existen diferentes tipos de hosting, cada uno con características específicas que se adaptan a distintas necesidades:

1. **Hosting compartido**: Varios sitios web comparten los recursos de un mismo servidor. Es una opción económica y adecuada para proyectos pequeños o personales.
2. **Servidor Privado Virtual (VPS):** Aunque varios sitios comparten el mismo servidor físico, cada uno tiene una porción dedicada de recursos, ofreciendo mayor control y rendimiento que el hosting compartido.
3. **Hosting dedicado**: Un servidor completo está dedicado a un solo sitio web, brindando el máximo rendimiento y control. Es ideal para sitios con alto tráfico o necesidades específicas de configuración.
4. **Hosting en la nube (cloud hosting):** Utiliza múltiples servidores interconectados para alojar sitios web, lo que permite una escalabilidad y disponibilidad superiores. Es adecuado para proyectos que requieren flexibilidad y capacidad de adaptación a picos de tráfico.

Empresas proveedoras poco comunes son:

- **Heroku**: Plataforma como servicio (PaaS) que facilita el despliegue y escalado de aplicaciones, especialmente para desarrolladores que buscan simplicidad en la gestión.
- **Amazon Web Services (AWS)**: Ofrece una amplia gama de servicios de infraestructura en la nube, incluyendo opciones de hosting escalables y seguras.
- **Render**: Proporciona una plataforma moderna para desplegar aplicaciones web y servicios, con un enfoque en la automatización y facilidad de uso.
- **Netlify y Vercel**: Especializadas en el alojamiento de sitios estáticos y aplicaciones frontend, ofreciendo integraciones con sistemas de control de versiones y flujos de trabajo de desarrollo modernos.

- **DigitalOcean**: Conocida por sus soluciones de infraestructura en la nube simples y accesibles, adecuada para desarrolladores y pequeñas empresas.

2.-Diferencia entre dominio, servidor y DNS

¿Qué es un dominio y cómo se registra?

Un dominio es el nombre que identifica a un sitio web en internet, como por ejemplo midominio.com. Sirve para que las personas no tengan que memorizar direcciones IP largas y difíciles, sino que puedan acceder fácilmente escribiendo un nombre sencillo en el navegador. Registrar un dominio implica comprarlo a través de registradores autorizados como GoDaddy, Namecheap o Google Domains, quienes permiten verificar su disponibilidad y realizar el pago para su uso exclusivo por un periodo determinado (generalmente un año o más) (ICANN, 2024).

¿Qué es un servidor web?

Un servidor web es una computadora (o una máquina virtual en la nube) que almacena los archivos de una página web, como imágenes, código HTML, hojas de estilo, bases de datos y demás elementos. Su función es responder a las solicitudes que hacen los usuarios cuando visitan el sitio desde sus navegadores. Es decir, cuando alguien entra a una página, el servidor "envía" la información necesaria para que se muestre correctamente en pantalla (Cloudflare, 2023).

¿Cómo funciona el DNS y qué papel tiene en la publicación de una app?

El Sistema de Nombres de Dominio (DNS, por sus siglas en inglés) actúa como una especie de directorio telefónico para internet. Cuando alguien escribe un dominio en su navegador, el DNS se encarga de traducir ese nombre en la dirección IP del servidor donde está alojada la web. De esta manera, permite que el navegador se conecte con el servidor correcto y muestre la página deseada. Sin el DNS, tendríamos que recordar direcciones IP numéricas en lugar de nombres fáciles de recordar (Google Developers, 2023).

Por ejemplo, si un usuario escribe www.ejemplo.com, el DNS busca qué dirección IP corresponde a ese dominio, y luego redirige al navegador para que se comunique con el servidor donde está alojada la aplicación. Este proceso suele tardar milisegundos, pero es crucial para que los sitios funcionen correctamente.

3.-Consideraciones para llevar una aplicación web a producción

Pasar una aplicación del entorno de desarrollo al entorno de producción no es simplemente “subirla a internet”. Este paso requiere seguir ciertas medidas para asegurar que funcione correctamente, que sea segura y que tenga buen rendimiento.

Diferencias entre entorno de desarrollo y producción

El entorno de desarrollo es donde los programadores crean y prueban nuevas funciones. Aquí no importa mucho el rendimiento, ni la seguridad, porque solo se usa de forma

interna. En cambio, el entorno de producción es donde la aplicación ya está disponible al público y necesita ser estable, rápida y segura. Por eso, en producción se desactivan herramientas de depuración, se optimiza el rendimiento y se controlan mejor los errores (Mozilla, 2023).

Variables de entorno y secretos

En lugar de escribir datos sensibles directamente en el código (como contraseñas o claves de acceso), se usan variables de entorno. Estas son configuraciones que se guardan aparte del código fuente y que el sistema puede leer al iniciar la aplicación. Esto mejora la seguridad, ya que evita exponer información delicada, sobre todo si se usa control de versiones como GitHub (12factor.net).

Por ejemplo, una aplicación puede tener la clave de acceso a su base de datos guardada como DB_PASSWORD=mi_clave_segura, y el código simplemente la lee al ejecutarse.

Seguridad

Al publicar una aplicación, debemos aplicar medidas básicas y más comunes de seguridad:

- Usar **HTTPS** en lugar de HTTP para cifrar la información entre el servidor y los usuarios (Let's Encrypt, 2024).
- Configurar **CORS** (Cross-Origin Resource Sharing) adecuadamente para controlar desde qué dominios se puede acceder a la aplicación (MDN Web Docs, 2023).
- Establecer sistemas de **autenticación y autorización** para proteger las rutas privadas.
- Usar bases de datos con contraseñas fuertes y restringir el acceso por IP o roles.

Logs y monitoreo

En producción, es fundamental llevar un registro de lo que ocurre dentro de la aplicación para poder detectar errores, caídas o comportamientos anómalos. Los logs son archivos donde se escriben eventos importantes, y el monitoreo permite visualizar en tiempo real el estado del sistema. Herramientas como Loggly, Datadog o Prometheus ayudan en este proceso (Loggly, 2024).

4.-Despliegue de aplicaciones en Angular

Cuando usamos angular todo el código que se hace debe ser preparado para subirlo a la red , a este proceso se le llama despliegue y esto aunque parezca fácil, no lo es necesita pasos específicos para que no cause errores y llegue bien al usuario final

Proceso de construcción (ng build --prod)

Una vez terminada la aplicación en Angular, el primer paso para desplegarla es ejecutando el comando ng build --configuration production (anteriormente --prod). Este comando genera una versión optimizada del proyecto: comprime los archivos, elimina

partes innecesarias y prepara todo para que cargue más rápido en producción (Angular CLI Docs, 2024).

El resultado de este comando se guarda en una carpeta llamada dist/, que contiene todos los archivos listos para ser subidos a un servidor web.

Estructura del directorio dist/ y su finalidad

La carpeta dist/ es el producto final del proyecto Angular. Contiene archivos como index.html, hojas de estilo (.css), archivos JavaScript minificados (.js) y recursos como imágenes. Esos son los archivos que se deben subir al servidor web o a una plataforma de hosting para que la aplicación esté disponible públicamente (Angular Docs, 2024).

Plataformas de hosting para aplicaciones estáticas

Como el frontend de Angular funciona como una app estática, existen plataformas muy populares y gratuitas para alojarla fácilmente:

- **Netlify**: Permite conectar el repositorio directamente desde GitHub y despliega automáticamente la app cuando detecta cambios. Ofrece HTTPS gratuito y dominio personalizado (Netlify Docs, 2024).
- **Vercel**: Similar a Netlify, ofrece despliegue automático y fácil integración con proyectos frontend. Es muy usado con frameworks modernos como React, Next.js o Angular (Vercel Docs, 2024).
- **GitHub Pages**: Sirve para alojar sitios estáticos desde un repositorio público de GitHub, ideal para proyectos personales, aunque tiene algunas limitaciones técnicas (GitHub Docs, 2024).

Servidores web compatibles (Nginx, Apache)

Para subir la app a tu propio servidor (por ejemplo, en una VPS o en la nube), puedes usar servidores como **Nginx** o **Apache**. Estos programas sirven los archivos estáticos (los del directorio dist/) al navegador del usuario. Nginx es especialmente recomendado por su velocidad y bajo consumo de recursos (NGINX Docs, 2024).

5.- Despliegue de aplicaciones Django

Django es uno de los framework más usado para el desarrollo de aplicaciones web del lado de Phyton para hacer uso de eso hay ajustes necesarios y configuraciones en este apartado hablaremos de ellas

Archivos importantes: manage.py, requirements.txt, settings.py

- manage.py: Es el archivo que permite ejecutar comandos de administración en Django, como migrar bases de datos o iniciar el servidor de desarrollo (Django Docs, 2024).
- requirements.txt: Contiene una lista de todas las dependencias del proyecto. Es fundamental al momento de desplegar, porque se usa para instalar los paquetes necesarios con el comando pip install -r requirements.txt (Python Packaging Authority, 2024).

- settings.py: Aquí se configuran aspectos clave del proyecto, como la conexión a la base de datos, las rutas estáticas, las variables de entorno y otros detalles que se deben ajustar al pasar a producción (Django Docs, 2024).

Configuración de producción

Al preparar el proyecto para producción, se deben modificar algunas variables del archivo settings.py:

- DEBUG=False: Desactiva los mensajes de error detallados que solo deben mostrarse en desarrollo.
- ALLOWED_HOSTS: Se deben agregar los dominios o IPs autorizadas para acceder a la app.
- Conexión a bases de datos reales como PostgreSQL o MySQL, en lugar de SQLite que se usa en desarrollo (Django Deployment Checklist, 2024).

Servidores WSGI: Gunicorn, uWSGI

Django no está diseñado para ejecutarse directamente en producción con su servidor incorporado. Por eso, se usa un servidor de aplicaciones WSGI como **Gunicorn** o **uWSGI**, que sirven de puente entre el servidor web y la aplicación Django (Gunicorn Docs, 2024).

Gunicorn, por ejemplo, se lanza con un comando como:

```
gunicorn myproject.wsgi:application --bind 0.0.0.0:8000
```

Reverse proxy con Nginx

Aunque Gunicorn sirve la aplicación, se recomienda usar **Nginx** como proxy inverso. Esto quiere decir que Nginx se encarga de recibir las peticiones externas y redirigirlas a Gunicorn, además de manejar archivos estáticos, compresión, seguridad HTTPS, y más (NGINX Docs, 2024).

Bases de datos en producción

En lugar de usar SQLite, en producción se suelen usar bases de datos más robustas como **PostgreSQL** o **MySQL**, que ofrecen mejor rendimiento, seguridad y opciones de respaldo (PostgreSQL Docs, 2024).

6.-Introduccion a Docker

Docker es una de las herramientas de contenedores más famosas y usadas esto ayuda muchas veces con los tediosos problemas de hardware o software lo que facilita muchísimo el uso de aplicaciones en distintas maquinas y no haya errores ya sean de configuración etc.

¿Qué es la virtualización por contenedores?

La virtualización por contenedores es una forma de ejecutar software de manera aislada, como si cada aplicación viviera en su propio espacio. A diferencia de las máquinas virtuales, que requieren un sistema operativo completo para cada instancia, los contenedores comparten el núcleo del sistema y consumen muchos menos recursos (Docker Docs, 2024).

Diferencia entre máquina virtual y contenedor

Una **máquina virtual** es la simula todo un sistema operativo y se ejecuta sobre un hipervisor. Es más pesada y lenta de iniciar, pero permite más aislamiento total. Un **contenedor**, en cambio, es más ligero porque comparte el sistema operativo del host y se inicia mucho más rápido. Esto los hace ideales para ambientes modernos de desarrollo y despliegue (IBM Cloud Docs, 2023).

Característica	Máquina Virtual	Contenedor
Arranque	Lento	Rápido
Tamaño	Grande (GBs)	Ligero (MBs)
Aislamiento	Fuerte	Ligero
Rendimiento	Menor	Mayor

Imagen vs Contenedor

Una **imagen** es como una receta: contiene las instrucciones para crear un contenedor. Un **contenedor** es la instancia en ejecución de esa imagen, es decir, es la app funcionando. Puedes tener varias copias de un mismo contenedor corriendo al mismo tiempo desde una sola imagen (Docker Docs, 2024).

¿Qué es un Dockerfile y un docker-compose.yml?

- El **Dockerfile** es un archivo de texto que define los pasos para construir una imagen de Docker. Por ejemplo, puedes indicar qué versión de Python usar, qué archivos copiar, qué paquetes instalar, y cómo iniciar tu app (Dockerfile Reference, 2024).
- El archivo **docker-compose.yml** permite definir múltiples contenedores y cómo deben interactuar entre ellos, como por ejemplo el backend, el frontend y la base de datos. Es muy útil cuando la aplicación tiene varios servicios funcionando al mismo tiempo (Docker Compose Docs, 2024).

Beneficios de contenerizar aplicaciones para producción

Al usar Docker, las aplicaciones se vuelven más portables, consistentes y fáciles de desplegar. No importa si estás en Windows, Linux o Mac: si usas la misma imagen Docker, la aplicación se comportará igual. Además, facilita la integración continua (CI/CD) y mejora la seguridad, ya que los contenedores se pueden aislar entre sí (Red Hat, 2024).

7.-Contenerización de una aplicación Angular y Django

Contener izar a una aplicación es como lo que se había hablado con Docker solo que en este caso con las aplicaciones de angular y Django ya que estas necesitan mucho de dependencias, servicios etc , para su correcto funcionamiento , y la versión de estas ya que aunque este instalado si su versión no es correcta no funcionará, contener izar permite que funcione en donde sea.

Docker para Angular: imagen de producción basada en Nginx

Para desplegar Angular con Docker, primero se debe construir el proyecto con el comando:

```
ng build --configuration production
```

Esto genera una carpeta dist/ con los archivos listos para producción. Luego, se crea un Dockerfile que copia esos archivos en una imagen de Nginx, un servidor web ligero y rápido. Un ejemplo de Dockerfile para Angular sería:

```
FROM nginx:alpine
COPY ./dist/mi-app /usr/share/nginx/html
```

Docker para Django: imagen con Gunicorn, base de datos externa, volúmenes y puertos

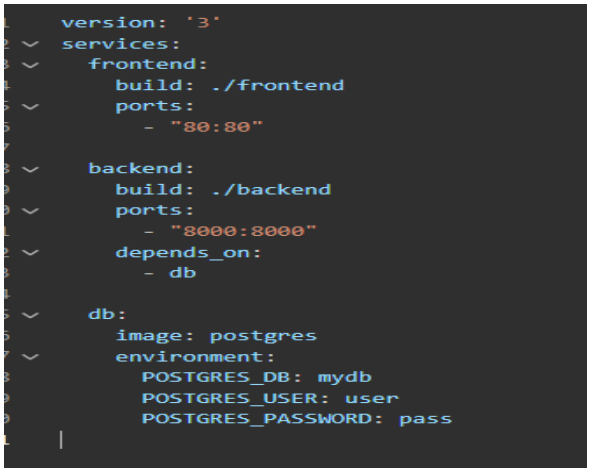
Para Django, se utiliza una imagen de Python que instala las dependencias y configura Gunicorn como servidor WSGI. Un ejemplo básico de Dockerfile sería:

```
FROM python:3.10
WORKDIR /app
COPY . .
RUN pip install -r requirements.txt
CMD ["gunicorn", "mi_proyecto.wsgi:application", "--bind", "0.0.0.0:8000"]
```

La base de datos (por ejemplo PostgreSQL) puede ir en otro contenedor, y se usan **volúmenes** para mantener los datos y **puertos** para comunicar los servicios entre sí

Ejemplo básico de docker-compose uniendo frontend y backend

Para que tanto Angular como Django funcionen juntos, se puede usar docker-compose, una herramienta que permite definir y ejecutar múltiples contenedores a la vez. Un ejemplo básico:



```yaml
version: '3'
services:
  frontend:
    build: ./frontend
    ports:
      - "80:80"

  backend:
    build: ./backend
    ports:
      - "8000:8000"
    depends_on:
      - db

  db:
    image: postgres
    environment:
      POSTGRES_DB: mydb
      POSTGRES_USER: user
      POSTGRES_PASSWORD: pass
```

8.- Conceptos de CI/CD (Integración continua y despliegue continuo)

Hoy en día, las aplicaciones se actualizan constantemente. Por eso, se necesita una forma automática y confiable de probar, construir y desplegar esas actualizaciones sin tener que hacerlo manualmente cada vez. Aquí es donde entran CI y CD.

¿Qué es CI/CD?

CI/CD significa *Integración Continua* (CI) y *Despliegue Continuo* (CD). Son prácticas que automatizan el proceso de desarrollo y publicación de software. Esto ayuda a entregar nuevas funciones, correcciones de errores y mejoras más rápido y con menos errores humanos (Red Hat, 2024).

- **Integración continua (CI)**: Es el proceso de automatizar las pruebas y validación del código cada vez que un desarrollador hace un cambio. Así se asegura que todo sigue funcionando como debe.
- **Despliegue continuo (CD)**: Es el proceso de llevar automáticamente el código probado hasta el entorno de producción o pruebas. Así, cada cambio aprobado puede publicarse sin pasos manuales.

Beneficios de automatizar el despliegue

- Se reduce el tiempo entre desarrollo y publicación.
- Aumenta la calidad del código, ya que se prueban los cambios constantemente.
- Permite detectar errores rápidamente.
- Mejora la colaboración entre desarrolladores y equipos.

Según GitHub, automatizar el despliegue también ayuda a mantener versiones estables y a minimizar los errores humanos durante la entrega de software (GitHub Docs, 2024).

Herramientas comunes

Existen varias plataformas que permiten implementar CI/CD de forma sencilla:

- **GitHub Actions**: Integrado en GitHub, permite crear flujos de trabajo que se activan con eventos como *push*, *pull request*, etc.
- **GitLab CI/CD**: Sistema de CI/CD completo integrado en GitLab.
- **CircleCI y Jenkins**: Plataformas independientes muy usadas en entornos empresariales (CircleCI Docs, 2024, Jenkins Docs, 2024).

Ejemplos de flujos de trabajo

Un flujo de CI/CD típico incluye:

1. Un desarrollador sube un cambio al repositorio.
2. GitHub Actions ejecuta pruebas automáticas.
3. Si todo funciona, se construye una imagen de Docker.
4. Se despliega automáticamente en un servidor o plataforma como Heroku, Render, AWS, etc.

Este proceso puede definirse en un archivo. yml dentro del repositorio:

```yaml
name: Deploy Angular App
on: [push]
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - run: npm install && npm run build
```

Diferencias entre CI, CD y CD (Continuous Delivery vs Deployment)

Concepto	**Descripción**
CI Integración Continua	Automatiza pruebas y validación al subir nuevo código
CD - Entrega Continua	El código se prepara automáticamente para producción, pero requiere aprobación manual
CD - Despliegue Continuo	El código aprobado se despliega automáticamente a producción sin intervención

Tomado de (Atlassian, 2023)

9.-Plataformas para despliegue automatizado

Cuando ya se quiere lanzar una aplicación es muy importante saber como y donde quieres alojarla , aquí hablamos de herramientas que facilitan este proceso

Servicios para backend y bases de datos

Render

Render es una plataforma moderna que permite desplegar tanto servicios backend como bases de datos con tan solo conectar el repositorio. Es ideal para aplicaciones en Django, Node.js, Flask, entre otros. Además, incluye certificados HTTPS y escalado automático en los planes pagos (Render Docs, 2024).

Heroku

Heroku fue una de las primeras plataformas PaaS (Plataforma como Servicio) y sigue siendo muy popular por su facilidad de uso. Permite desplegar backend y bases de datos con comandos simples y tiene integración directa con GitHub. Aunque ha limitado su plan gratuito, sigue siendo excelente para proyectos pequeños o educativos (Heroku Dev Center, 2024).

Railway

Railway es una alternativa moderna similar a Render, con interfaz muy sencilla y gratuita para proyectos básicos. Permite levantar backends y bases de datos con un clic, y se configura automáticamente al conectar un repositorio (Railway Docs, 2024).

Servicios para frontend

Netlify

Netlify es ampliamente usado para desplegar aplicaciones frontend como Angular, React o sitios estáticos. Ofrece integración con GitHub, HTTPS automático, rutas personalizadas y funciones serverless si se desea ampliar. Ideal para proyectos modernos (Netlify Docs, 2024).

Vercel

Vercel se especializa en apps frontend y estáticas. Su enfoque principal es la velocidad y la simplicidad. Soporta frameworks modernos y tiene un sistema de previews automáticos por cada rama, ideal para trabajar en equipo (Vercel Docs, 2024).

Firebase Hosting

Desarrollado por Google, Firebase Hosting es una opción sólida para alojar sitios estáticos, especialmente si ya usas otras herramientas de Firebase como Firestore o Auth. Brinda seguridad, velocidad y se integra fácilmente con otras plataformas de Google (Firebase Docs, 2024).

Infraestructura personalizada: DockerHub + GitHub Actions + DigitalOcean / AWS

Para quienes necesitan más control, se puede construir una solución personalizada con herramientas como:

- **DockerHub**: Almacena imágenes Docker listas para ser desplegadas.
- **GitHub Actions**: Automatiza la construcción y envío de esas imágenes.
- **DigitalOcean o AWS**: Servidores donde se despliegan las imágenes y se configuran con mayor detalle según las necesidades del proyecto (Docker Docs, 2024, AWS Docs, 2024).

Este tipo de infraestructura es más avanzada, pero también ofrece más flexibilidad, especialmente en proyectos empresariales o que requieren recursos dedicados.

10.-Buenas prácticas en despliegue

Uso de .env para variables sensibles

Uno de los errores más comunes en desarrollo es dejar contraseñas o claves API escritas directamente en el código. Para evitar eso, se recomienda usar archivos .env, donde se colocan las variables sensibles, como:

```
SECRET_KEY=mi_clave_super_secreta
DATABASE_PASSWORD=mi_pass_bd
```

Separación de ambientes (dev, staging, prod)

Tener distintos ambientes ayuda a evitar errores catastróficos. Lo ideal es trabajar así:

- **Desarrollo (dev)**: para programar y probar.
- **Staging**: para pruebas más realistas antes de lanzar.
- **Producción (prod)**: el entorno real y público.

Monitoreo, backups y recuperación

Tener herramientas de monitoreo (como Datadog, Prometheus o New Relic) permite detectar errores o caídas antes de que afecten a los usuarios. También es importante hacer copias de seguridad automáticas y tener un plan de recuperación si algo falla.

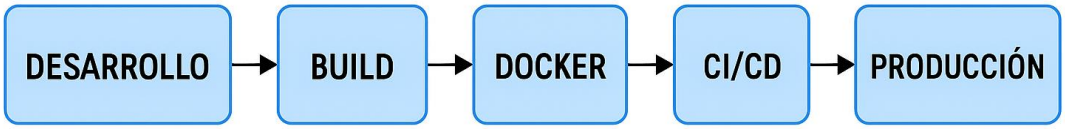
Las plataformas como Heroku, Render y Railway ya ofrecen algunas funciones integradas, pero en servidores propios es esencial configurarlas manualmente (Datadog Docs, 2024, DigitalOcean Monitoring, 2024).

Actualización continua sin downtime

Es una buena práctica automatizar el despliegue sin tener que detener el sistema. Para eso se puede usar blue-green deployment o rolling updates, donde una nueva versión se

publica mientras la anterior sigue funcionando, y solo se hace el cambio cuando todo está listo (Red Hat, 2024).

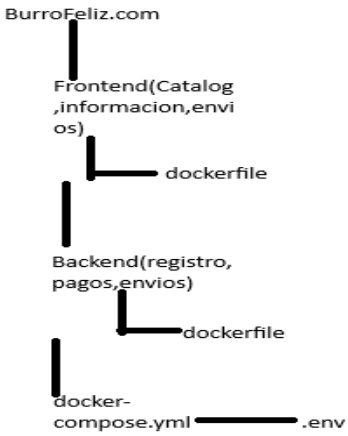
11 Diagrama explicativo de flujo de despliegue



12-Ejemplo hipotético

Supongamos que haremos una tienda online de disfraces para burros ,para que el comprador pueda ver los difraces para burros se necesita angular y para que se registren y hagan sus compras se necesitan bases de datos y una conexión para manejarlas Django y PostgreSQL

La estructura del ejemplo hipotético presentado sería algo como esto:



Ejemplo de Docker file para angular:

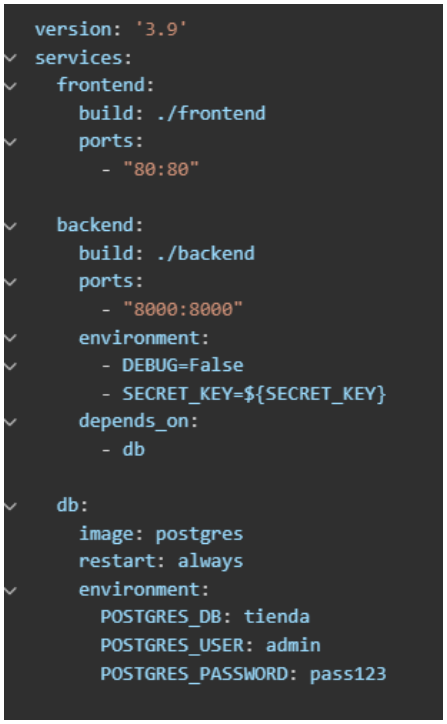
```
FROM node:18 AS builder
WORKDIR /app
COPY . .
RUN npm install && npm run build

FROM nginx:alpine
COPY --from=builder /app/dist/burro-tienda /usr/share/nginx/html
```

Ejemplo de DockerFile para Django:

```
FROM python:3.10
WORKDIR /app
COPY . .
RUN pip install -r requirements.txt
CMD ["gunicorn", "burrotienda.wsgi:application", "--bind", "0.0.0.0:8000"]
```

docker-compose.yml



```yaml
version: '3.9'
services:
  frontend:
    build: ./frontend
    ports:
      - "80:80"

  backend:
    build: ./backend
    ports:
      - "8000:8000"
    environment:
      - DEBUG=False
      - SECRET_KEY=${SECRET_KEY}
    depends_on:
      - db

  db:
    image: postgres
    restart: always
    environment:
      POSTGRES_DB: tienda
      POSTGRES_USER: admin
      POSTGRES_PASSWORD: pass123
```

13 conclusiones

Realizar esta investigación sobre el despliegue profesional de aplicaciones web fue una experiencia muy buena, porque me permitió entender que una aplicación no termina cuando se termina de programar que es lo que en principio uno entendería que es lo que pasa . De hecho, llevar una app al entorno de producción es un proceso igual de importante que escribir su código y laborioso sobre todo pero que al existir tantas herramientas nos permite una compresión más grande de estas aplicaciones y de la información que se maneja en nuestra era, simplemente es mucha.

Antes de hacer este trabajo, pensaba que desplegar una aplicación era simplemente “subirla” a internet como muchos cursos te lo venden , en su mayoría solo hacer la parte visual y listo, pero ahora sé que implica muchas decisiones técnicas y buenas prácticas para asegurar que funcione bien, sea segura y pueda escalarse sin complicaciones.

Aprendí que existen diferentes tipos de servicios de hosting, cada uno con sus ventajas y limitaciones, y que el concepto de dominio, servidor y DNS es fundamental para que una app esté disponible en la web. También entendí por qué es tan importante diferenciar entre desarrollo y producción, y cómo herramientas como los archivos .env, los logs y el monitoreo juegan un papel esencial para mantener una aplicación funcional y sin errores.

Una de las partes que más me interesó fue la contenerización con Docker. Me parece increíble que podamos “empaquetar” una app con todo lo necesario para que funcione en cualquier lugar, sin preocuparnos por diferencias entre computadoras o sistemas operativos. Junto con CI/CD, estas herramientas hacen posible que el proceso de actualización y despliegue sea rápido, automático y confiable. Eso es algo que cualquier empresa necesita si quiere trabajar de manera eficiente.

Además, al investigar ejemplos de plataformas como Heroku, Netlify, Vercel o Render, vi que hoy en día hay muchas opciones para desarrollar y desplegar proyectos de forma profesional, incluso con pocos recursos. Sin duda, todo lo que aprendí en esta tarea me será útil en el futuro, tanto para proyectos personales como para trabajos más grandes o profesionales y más ya que ahora puedo hacer mis proyectos funcionales sin necesidad de estar checando las maquinas o sufriendo más de este tipo de problemas que antes tenia.

Fue muy interesante aprender estas herramientas y pienso que son el futuro de las aplicaciones y las paginas web.

14.- Referencias

Despliegue Angular y Django

- Angular CLI. (2024). *ng build documentation*. https://angular.io/cli/build
- Angular Team. (2024). *Deployment overview*. https://angular.io/guide/deployment
- Docker Inc. (2024). *Django Sample App*. https://docs.docker.com/samples/django/
- Gunicorn. (2024). *Gunicorn documentation*. https://docs.gunicorn.org/en/stable/
- NGINX. (2024). *Reverse Proxy Guide*. https://docs.nginx.com/nginx/admin-guide/web-server/reverse-proxy/
- PostgreSQL Global Development Group. (2024). *Documentation*. https://www.postgresql.org/docs/

Docker y contenerización

- IBM Cloud. (2023). *Containers vs VMs*. https://www.ibm.com/cloud/blog/containers-vs-vms
- Docker Inc. (2024). *Dockerfile Reference*. https://docs.docker.com/engine/reference/builder/
- Docker Inc. (2024). *Docker Compose Documentation*. https://docs.docker.com/compose/

- Red Hat. (2024). *What is Docker?* https://www.redhat.com/en/topics/containers/what-is-docker

CI/CD y flujos de trabajo

- Atlassian. (2023). *CI/CD explained*. https://www.atlassian.com/continuous-delivery/ci-vs-ci-vs-cd
- CircleCI. (2024). *Documentation*. https://circleci.com/docs/
- Jenkins. (2024). *User Documentation*. https://www.jenkins.io/doc/

Plataformas de despliegue

- Render. (2024). *Render Docs*. https://render.com/docs
- Heroku. (2024). *Dev Center*. https://devcenter.heroku.com/
- Railway. (2024). *Docs*. https://docs.railway.app/
- Netlify. (2024). *Docs*. https://docs.netlify.com/
- Vercel. (2024). *Documentation*. https://vercel.com/docs
- Firebase. (2024). *Firebase Hosting*. https://firebase.google.com/docs/hosting
- Amazon Web Services (AWS). (2024). *AWS Documentation*. https://docs.aws.amazon.com/

Buenas prácticas y monitoreo

- Dotenv. (2024). *Dotenv GitHub Repository*. https://github.com/motdotla/dotenv
- Microsoft Azure. (2024). *DevOps environments best practices*. https://learn.microsoft.com/en-us/azure/architecture/best-practices/devops-environments
- Datadog. (2024). *Monitoring Docs*. https://docs.datadoghq.com/
- DigitalOcean. (2024). *Monitoring Documentation*. https://docs.digitalocean.com/products/monitoring/
- Red Hat. (2024). *Blue-Green Deployment*. https://www.redhat.com/en/topics/devops/what-is-blue-green-deployment