Servicios de Red e Internet

Despliegue de Servidores Web con Nginx y Docker

Módulo: Flipped Classroom

ILERNA - 2º Curso

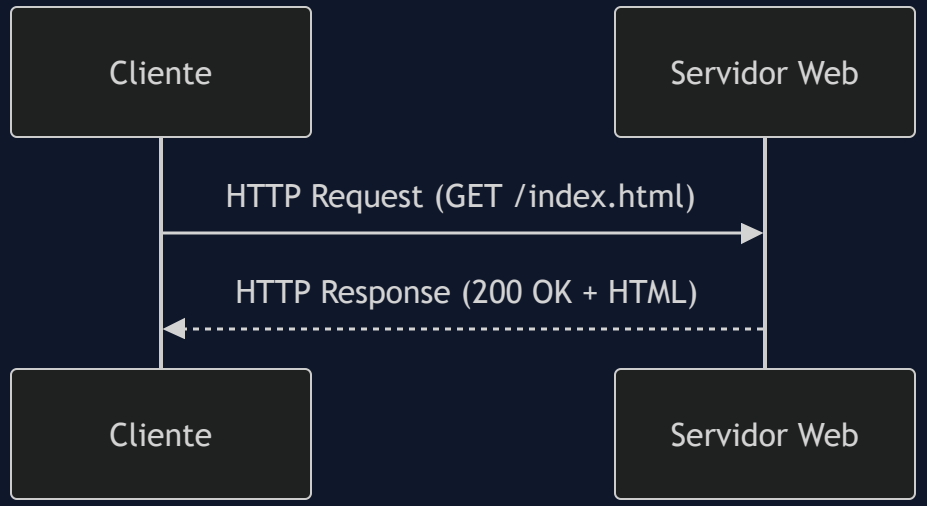
📋 Índice de Contenidos

BLOQUE 1

Fundamentos Web

¿Qué es un Servidor Web?

Un servidor web es software que **recibe peticiones HTTP** de clientes y devuelve respuestas (HTML, JSON, archivos...).



Servidores más usados:

- 🔵 **Apache** — El más antiguo y extendido
- 🟢 **Nginx** — Alto rendimiento, event-driven
- 🟡 **Caddy** — Moderno, HTTPS automático

Protocolo HTTP

HTTP (HyperText Transfer Protocol) es el protocolo de comunicación de la web.

Método	Uso
`GET`	Obtener recurso
`POST`	Enviar datos
`PUT`	Actualizar recurso
`DELETE`	Eliminar recurso

Códigos de estado más importantes:

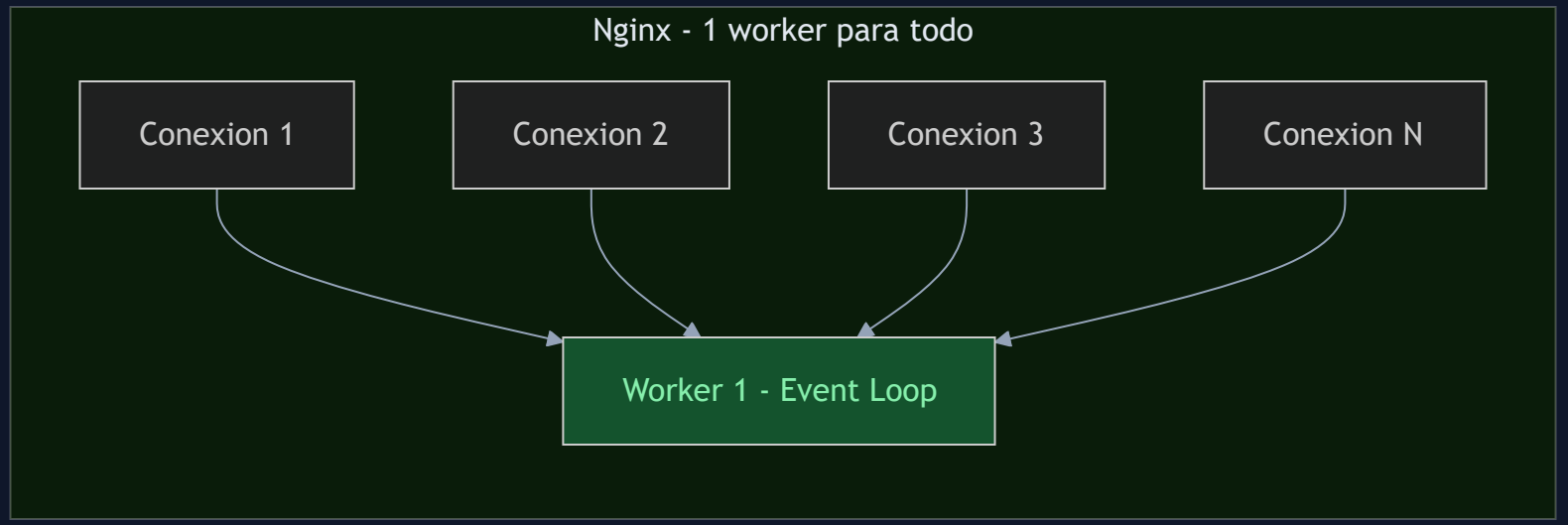
- `200 OK` — Éxito
- `301 Moved Permanently` — Redirección permanente
- `404 Not Found` — Recurso no encontrado
- `502 Bad Gateway` — Error de proxy

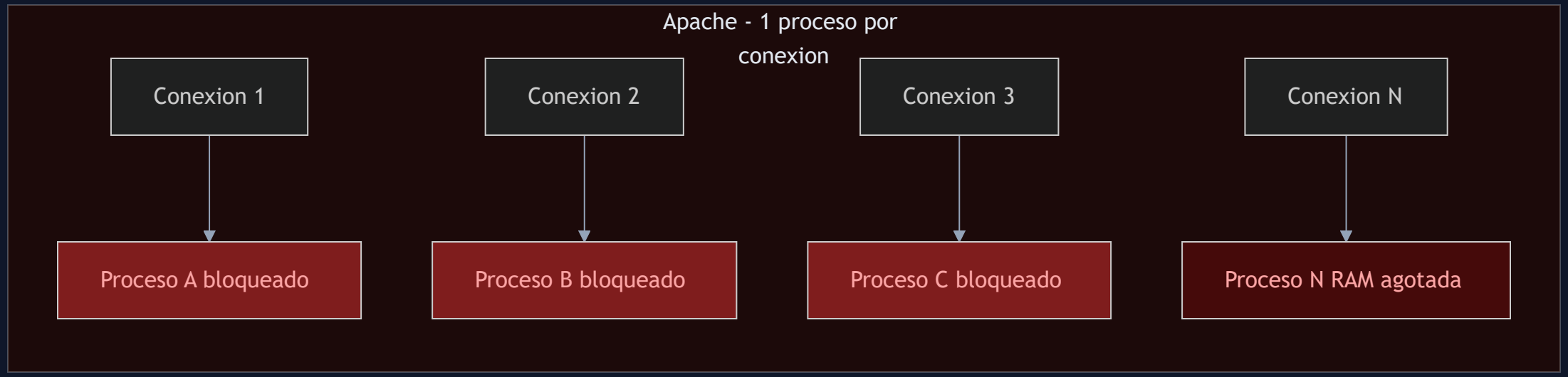
HTTP vs HTTPS

Característica	HTTP	HTTPS
Puerto	80	443
Cifrado	❌ No	✅ TLS/SSL
Seguridad	Datos en claro	Datos cifrados
SEO	Penalizado	Favorecido
Certificado	No necesario	Obligatorio

> ⚠️ **Nunca** enviar contraseñas o datos sensibles por HTTP

Apache vs Nginx: Arquitectura





El Problema C10K

C10K = manejar **10.000 conexiones concurrentes**

- Apache con 10.000 conexiones → **10.000 procesos** → GBs de RAM
- Nginx con 10.000 conexiones → **pocos workers** → ~10-20 MB RAM

¿Por qué importa?

- Un servidor de producción puede recibir miles de peticiones simultáneas
- Nginx fue diseñado específicamente para resolver este problema
- Es la razón principal por la que Nginx domina en producción

Headers HTTP Importantes

```
GET /api/users HTTP/1.1
Host: example.com
Accept: application/json
Authorization: Bearer eyJhbGc...
```

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: max-age=3600
X-Frame-Options: SAMEORIGIN
Strict-Transport-Security: max-age=31536000
```

Los headers controlan: caché, seguridad, tipo de contenido, autenticación, redirecciones...

BLOQUE 2

Docker e Infraestructura

¿Qué es Docker?

Docker permite **empaquetar aplicaciones** con todas sus dependencias en contenedores portables.

Sin Docker:

> "En mi máquina funciona..." 🤷

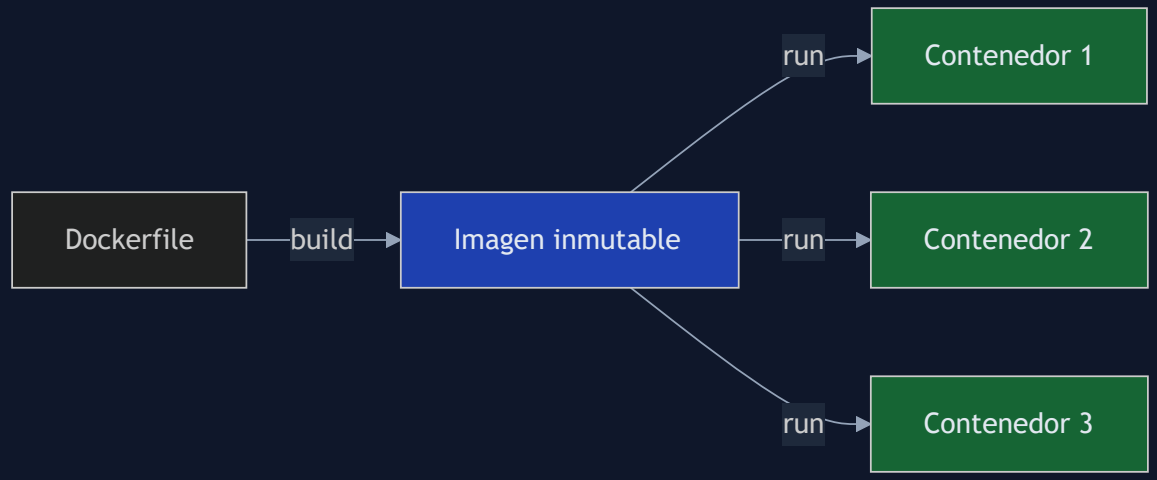
Con Docker:

> El contenedor funciona igual en desarrollo, staging y producción ✅

Conceptos clave:

- 📦 **Imagen** — Plantilla inmutable (como una clase)
- 🏃 **Contenedor** — Instancia en ejecución (como un objeto)
- 📋 **Dockerfile** — Receta para construir la imagen
- 🗂️ **Registry** — Repositorio de imágenes (Docker Hub)

Imagen vs Contenedor



Analogía:

- Imagen = Clase en POO
- Contenedor = Objeto instanciado

Comandos básicos:

```
docker build -t mi-app .
docker run -p 80:80 mi-app
docker ps
docker stop <id>
```

Dockerfile: Instrucciones Clave

```
FROM nginx:1.25-alpine3.19
LABEL maintainer="admin@example.com"
RUN apk add --no-cache curl
COPY nginx.conf /etc/nginx/nginx.conf
EXPOSE 80
USER nginx
CMD ["nginx", "-g", "daemon off;"]
```

Instrucción	Uso
`FROM`	Imagen base
`LABEL`	Metadatos
`RUN`	Ejecutar en build
`COPY`	Copiar archivos
`EXPOSE`	Documentar puerto
`USER`	Usuario de ejecución
`CMD`	Comando por defecto

Buenas Prácticas en Dockerfile

✅ **Versiones específicas**

`nginx:1.25-alpine3.19` no `:latest`

✅ **Minimizar capas**

Combinar RUN con `&&`

✅ **Imagen Alpine**

~5MB base vs ~80MB Debian

✅ **No ejecutar como root**

`USER nginx`

✅ **Multi-stage builds**

Separar build de runtime

```
# ❌ Mal: 3 capas
RUN apt-get update
RUN apt-get install -y curl
RUN apt-get clean

# ✅ Bien: 1 capa
RUN apt-get update && \
    apt-get install -y curl && \
    apt-get clean
```

Volúmenes: Persistencia de Datos

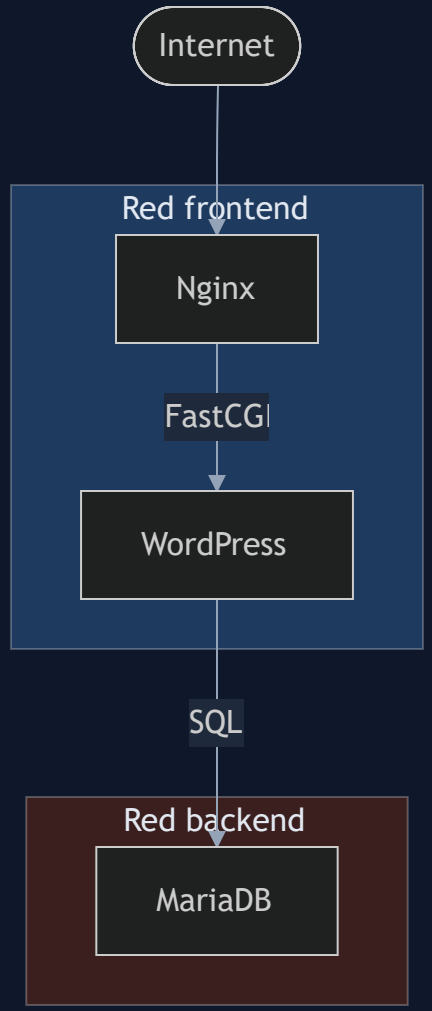
Los contenedores son **efímeros** — al eliminarse, los datos se pierden.

Tipos de montaje:

Tipo	Descripción	Uso
Named Volume	Gestionado por Docker	Bases de datos, producción
Bind Mount	Directorio del host	Desarrollo local
tmpfs	En RAM, temporal	Datos sensibles temporales

```
volumes:
  # Named volume (gestionado por Docker)
  - db_data:/var/lib/mysql
  # Bind mount de solo lectura
  - ./config:/etc/nginx:ro
```

Redes en Docker



Redes separadas = aislamiento de seguridad

Docker Compose

Docker Compose orquesta **múltiples contenedores** con un solo archivo.

```
version: '3.8'
services:
  web:
    image: nginx:1.25-alpine3.19
    ports:
      - "80:80"
    depends_on: [app]
    networks: [frontend]
  app:
    image: wordpress:6.4-php8.2-fpm
    networks: [frontend, backend]
networks:
  frontend:
  backend:
    internal: true
```

Healthchecks en Docker

Permiten que Docker **sepa si el servicio está realmente listo**.

```
services:
  db:
    image: mariadb:11.2
    healthcheck:
      test: ["CMD", "healthcheck.sh", "--connect"]
      interval: 30s
      timeout: 10s
      retries: 3
      start_period: 60s
  wordpress:
    depends_on:
      db:
        condition: service_healthy
```

> ⚠️ `depends_on` sin `condition` solo espera que el contenedor **arranque**, no que esté **listo**

⚙ BLOQUE 3

Configuración de Nginx

Estructura de Configuración Nginx

```
/etc/nginx/
├── nginx.conf              # Configuración principal
├── conf.d/
│   ├── default.conf        # Virtual host por defecto
│   └── mi-sitio.conf       # Virtual hosts adicionales
└── sites-enabled/          # (alternativa en Debian/Ubuntu)
```

Jerarquía de contextos:

```
# Contexto global
events { }          # Configuración de conexiones

http {              # Contexto HTTP
    server {        # Virtual host
        location /  # Rutas específicas
        { }
    }
}
```

Virtual Hosts (Server Blocks)

```
server {
    listen 80;
    server_name example.com www.example.com;
    root /var/www/html;
    index index.html;

    # Servir archivos estáticos
    location / {
        try_files $uri $uri/ =404;
    }

    # Pasar PHP a PHP-FPM
    location ~ \.php$ {
        fastcgi_pass php-fpm:9000;
        fastcgi_param SCRIPT_FILENAME
            $document_root$fastcgi_script_name;
        include fastcgi_params;
    }
}
```

Location Blocks: Prioridad

```
# 1. Exacto (máxima prioridad)
location = /favicon.ico { }

# 2. Prefix preferente (evita regex)
location ^~ /static/ { }

# 3. Regex case-sensitive
location ~ \.php$ { }

# 4. Regex case-insensitive
location ~* \.(jpg|png|gif)$ { }

# 5. Prefix (menor prioridad)
location /api/ { }
```

Orden de evaluación: `=` → `^~` → `~` / `~*` → prefix más largo

Variables Útiles en Nginx

Variable	Valor
`$host`	Nombre del servidor (dominio)
`$remote_addr`	IP del cliente
`$request_uri`	URI completa con query string
`$scheme`	`http` o `https`
`$server_name`	Nombre del server block
`$uri`	URI sin query string
`$args`	Query string
`$http_user_agent`	User-Agent del cliente

```
# Ejemplo: log personalizado
log_format custom '$remote_addr - $host "$request_uri" $status';
```

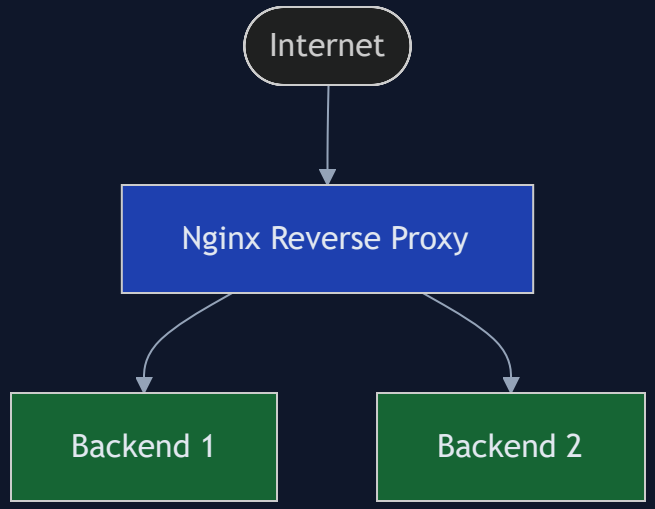
Servir Archivos Estáticos Eficientemente

```
server {
    # Caché agresiva para assets con hash en nombre
    location ~* \.(css|js|woff2)$ {
        expires 1y;
        add_header Cache-Control "public, immutable";
    }

    # Caché moderada para imágenes
    location ~* \.(jpg|png|gif|webp|svg)$ {
        expires 30d;
        add_header Cache-Control "public";
    }

    # Compresión Gzip
    gzip on;
    gzip_types text/plain text/css application/javascript;
    gzip_min_length 1000;
}
```

BLOQUE 4

Proxy Inverso y Balanceo de Carga

¿Qué es un Reverse Proxy?



Ventajas:

- 🔒 Oculta infraestructura interna
- ⚖️ Balanceo de carga
- 🔐 SSL Termination centralizado
- 💾 Caché de respuestas
- 🛡️ Protección DDoS

Sin reverse proxy:

Cada backend expuesto directamente a internet, sin control centralizado.

Con reverse proxy:

Un solo punto de entrada, backends protegidos en red interna.

Configuración Básica de Proxy

```
server {
    listen 80;
    server_name api.example.com;

    location / {
        proxy_pass http://backend:3000;

        # Headers esenciales
        proxy_set_header Host             $host;
        proxy_set_header X-Real-IP        $remote_addr;
        proxy_set_header X-Forwarded-For  $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;

        # Timeouts
        proxy_connect_timeout 5s;
        proxy_read_timeout    30s;
    }
}
```

Upstream: Grupo de Backends

```
http {
    upstream api_backend {
        server backend1:3000;
        server backend2:3000;
        server backend3:3000;
        server backup-server:3000 backup;
    }

    server {
        location /api/ {
            proxy_pass http://api_backend;
        }
    }
}
```

Sin configuración adicional → Round Robin por defecto

Algoritmos de Balanceo

Round Robin (default)

```
upstream backend {
    server s1:3000 weight=2;
    server s2:3000 weight=1;
    server s3:3000 weight=1;
}
```

Least Connections

```
upstream backend {
    least_conn;
    server s1:3000;
    server s2:3000;
}
```

IP Hash (Sticky Sessions)

```
upstream backend {
    ip_hash;
    server s1:3000;
    server s2:3000;
}
```

💡 **IP Hash** garantiza que el mismo cliente siempre va al mismo backend (útil para sesiones)

⚠️ **Least Conn** es mejor cuando los requests tienen duración variable

Comparativa de Algoritmos

Algoritmo	Ideal para	Sticky Sessions	Considera carga real
Round Robin	Backends homogéneos	❌	❌
Weighted RR	Backends con diferente capacidad	❌	❌
Least Conn	Requests de duración variable	❌	✅
IP Hash	Apps con sesiones locales	✅	❌

> 💡 **Recomendación:** Evitar sticky sessions usando **sesiones compartidas** (Redis)

Health Checks y Failover

```
upstream backend {
    server s1:3000
        max_fails=3 fail_timeout=30s;
    server s2:3000
        max_fails=3 fail_timeout=30s;
    server s3:3000 backup;
}
server {
    location / {
        proxy_pass http://backend;
        proxy_next_upstream error timeout
            http_502 http_503;
        proxy_next_upstream_tries 3;
    }
}
```

Flujo de failover:

1. s1 falla 3 veces
   → marcado como **down** 30s
2. Tráfico redirigido a **s2**
3. Si s2 también falla
   → activa **servidor backup**

Parámetros clave:

- `max_fails` — intentos antes de marcar down
- `fail_timeout` — tiempo en estado down
- `backup` — solo activo si todos fallan

SSL Termination

```
Cliente
  | HTTPS
  ▼
Nginx (descifra)
  | HTTP interno
  ▼
Backend (privado)
```

Ventajas:

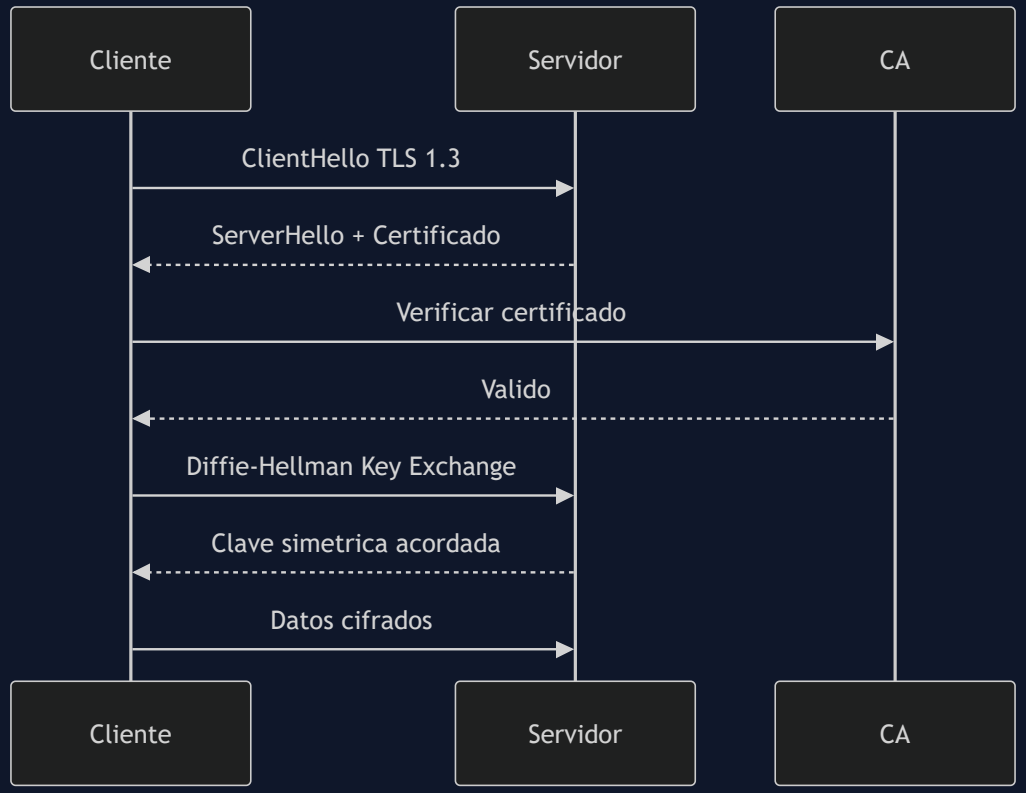
- Backends sin gestionar SSL
- Certificados centralizados
- HTTP interno más rápido

```
server {
    listen 443 ssl http2;
    ssl_certificate
        /etc/nginx/certs/cert.pem;
    ssl_certificate_key
        /etc/nginx/certs/key.pem;
    ssl_protocols TLSv1.2 TLSv1.3;

    location / {
        proxy_pass http://backend;
        proxy_set_header
            X-Forwarded-Proto https;
    }
}
```

🔒 BLOQUE 5

Seguridad y Hardening

HTTPS: Cómo Funciona TLS



Tipos de certificados:

- 🔴 **Autofirmado** — Solo para desarrollo
- 🟡 **Let's Encrypt** — Gratuito, automático
- 🟢 **CA Comercial** — Pago, mayor garantía

Fases del handshake:

1. ClientHello — versión y cifrados
2. ServerHello + Certificado
3. Verificación con CA
4. Diffie-Hellman → clave simétrica
5. Comunicación cifrada

Configuración HTTPS en Nginx

```
server {
    listen 80;
    server_name example.com;
    # Redirección permanente a HTTPS
    return 301 https://$server_name$request_uri;
}

server {
    listen 443 ssl http2;
    server_name example.com;

    ssl_certificate     /etc/nginx/certs/fullchain.pem;
    ssl_certificate_key /etc/nginx/certs/privkey.pem;

    # Solo protocolos seguros
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_prefer_server_ciphers on;
    ssl_ciphers 'ECDHE-ECDSA-AES128-GCM-SHA256
                :ECDHE-RSA-AES128-GCM-SHA256';
}
```

HSTS: HTTP Strict Transport Security

Problema: Primera visita puede ir por HTTP (vulnerable a MITM)

Solución: HSTS le dice al navegador "recuerda usar HTTPS siempre"

```
add_header Strict-Transport-Security
    "max-age=31536000; includeSubDomains; preload" always;
```

Funcionamiento:

```
1ª visita:  HTTP → Nginx redirige a HTTPS
            → envía header HSTS
2ª visita:  Navegador fuerza HTTPS
            antes de enviar request
Siempre:    Certificado inválido
            → navegador BLOQUEA
```

Previene: SSL Stripping, Downgrade Attacks, MITM

Headers de Seguridad

```
add_header X-Frame-Options
    "SAMEORIGIN" always;

add_header X-Content-Type-Options
    "nosniff" always;

server_tokens off;

add_header Referrer-Policy
    "strict-origin-when-cross-origin"
    always;

add_header Content-Security-Policy
    "default-src 'self'" always;
```

Header	Protege contra
`X-Frame-Options`	Clickjacking
`X-Content-Type-Options`	MIME sniffing
`server_tokens off`	Fingerprinting
`Referrer-Policy`	Fuga de URLs
`Content-Security-Policy`	XSS

Autenticación HTTP Básica

```
location /admin {
    # Activar autenticación básica
    auth_basic "Área Restringida";
    auth_basic_user_file /etc/nginx/.htpasswd;

    proxy_pass http://backend;
}
```

Generar archivo de contraseñas:

```
# Instalar herramienta
apt-get install apache2-utils

# Crear usuario (pide contraseña)
htpasswd -c /etc/nginx/.htpasswd admin

# Añadir más usuarios
htpasswd /etc/nginx/.htpasswd usuario2
```

> ⚠ **Siempre** usar auth_basic **con HTTPS** — las credenciales van en Base64

Rate Limiting

Protege contra: fuerza bruta, DDoS capa 7, scraping abusivo

```
http {
    # Zonas de rate limiting
    limit_req_zone $binary_remote_addr
        zone=login:10m rate=5r/m;
    limit_req_zone $binary_remote_addr
        zone=api:10m   rate=100r/m;
}

server {
    # Login: máximo 5 intentos por minuto
    location /wp-login.php {
        limit_req zone=login burst=2 nodelay;
        limit_req_status 429;
    }

    # API general: 100 req/min con tolerancia de 20
    location /api/ {
        limit_req zone=api burst=20 nodelay;
    }
}
```

Protección de Archivos Sensibles

```
server {
    # Bloquear acceso a wp-config.php
    location = /wp-config.php {
        deny all;
        return 404;
    }

    # Archivos ocultos (.htaccess, .env, .git)
    location ~ /\. {
        deny all;
        return 404;
    }

    # xmlrpc.php (vector de ataques WP)
    location = /xmlrpc.php {
        deny all;
        return 404;
    }

    # Control de acceso por IP
    location /admin {
        allow 192.168.1.0/24;
        deny all;
    }
}
```

Checklist de Seguridad Nginx

- ✅ `server_tokens off` — Ocultar versión
- ✅ `ssl_protocols TLSv1.2 TLSv1.3` — Solo protocolos seguros
- ✅ Redirección HTTP → HTTPS
- ✅ HSTS con `max-age=31536000`
- ✅ `X-Frame-Options: SAMEORIGIN`
- ✅ `X-Content-Type-Options: nosniff`
- ✅ Rate limiting en endpoints sensibles
- ✅ Bloquear archivos sensibles (`.env`, `.git`, `wp-config.php`)
- ✅ Autenticación en paneles de administración
- ✅ Certificados SSL válidos (no autofirmados en producción)

BLOQUE 6

Actividades Prácticas

Actividad 1: Stack WordPress con Seguridad

Objetivo: Desplegar WordPress con Nginx, MariaDB y PHP-FPM en Docker con seguridad.

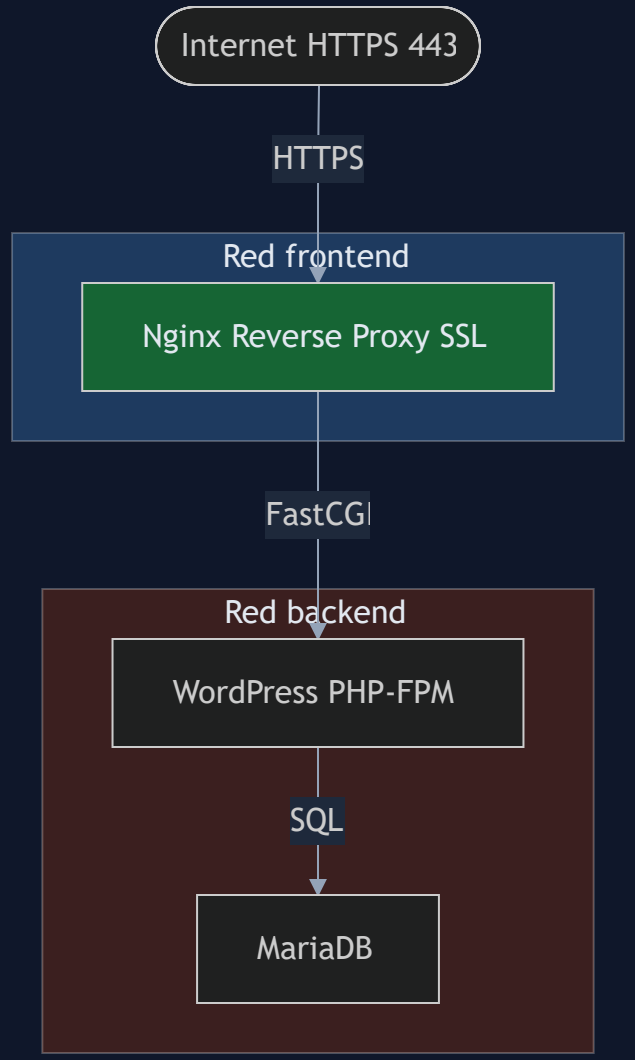
Tecnologías:

- 🐋 Docker + Docker Compose
- 🟢 Nginx (reverse proxy + SSL)
- 🗄️ MariaDB
- 🐘 PHP-FPM

Requisitos de seguridad:

1. HTTPS con certificado autofirmado
2. Redirección HTTP → HTTPS
3. HSTS habilitado
4. Auth básica en `/wp-admin`
5. Rate limiting en `/wp-login.php`
6. Bloqueo de archivos sensibles

Actividad 1: Arquitectura



Actividad 2: Balanceo de Carga

Objetivo: Implementar un sistema de balanceo de carga con alta disponibilidad usando Nginx y múltiples backends Node.js.

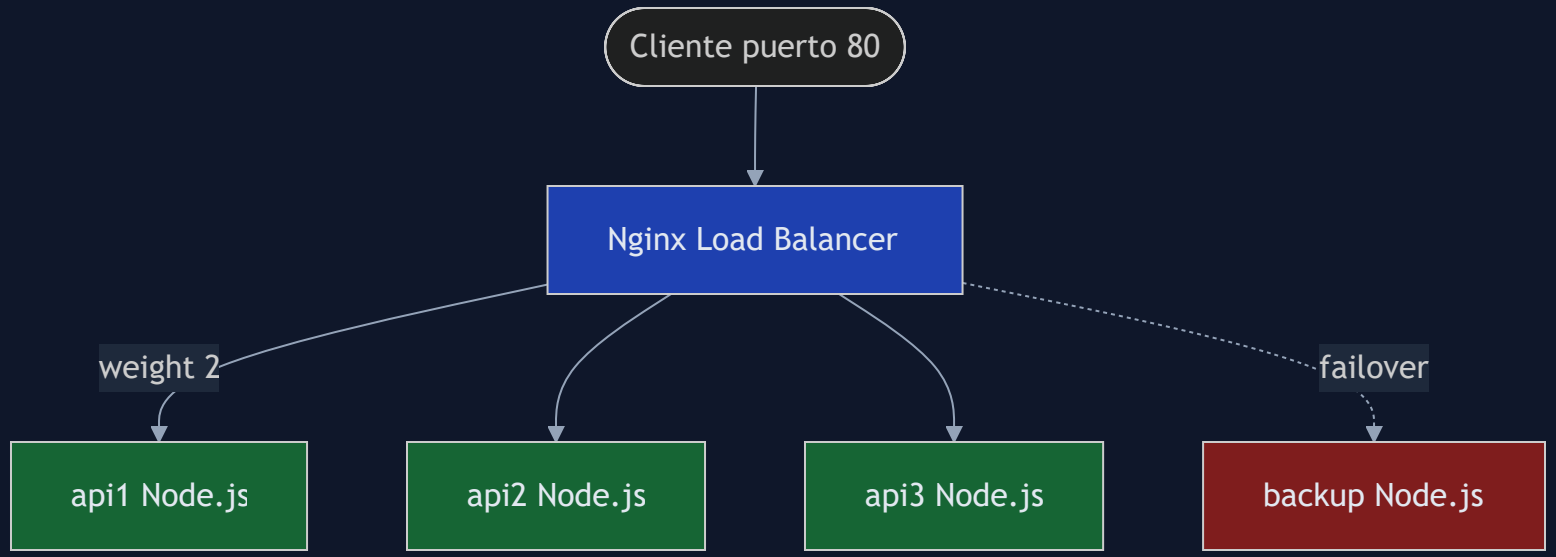
Tecnologías:

- 🐳 Docker + Docker Compose
- 🟢 Nginx (load balancer)
- 🟡 Node.js (3 instancias backend)

Requisitos:

1. 3 algoritmos de balanceo (Round Robin, Least Conn, IP Hash)
2. Health checks con failover automático
3. Servidor backup
4. Logs personalizados mostrando qué backend responde
5. Script de pruebas para verificar distribución

Actividad 2: Arquitectura



Comandos Útiles para las Actividades

```
# Levantar stack
docker compose up -d

# Ver logs en tiempo real
docker compose logs -f nginx

# Verificar HTTPS
curl -k https://localhost -I

# Probar rate limiting (debe dar 429 al 6º intento)
for i in {1..6}; do curl -s -o /dev/null -w "%{http_code}\n" \
  https://localhost/wp-login.php; done

# Verificar balanceo de carga
for i in {1..9}; do curl -s http://localhost/api | jq .server; done

# Ver estado de contenedores
docker compose ps

# Entrar en contenedor
docker compose exec nginx sh
```

Errores Comunes y Soluciones

Error	Causa	Solución
`502 Bad Gateway`	Backend no disponible	Verificar que el backend está corriendo
`403 Forbidden`	Permisos incorrectos	`chmod 755` en directorio web
`SSL_ERROR`	Certificado inválido	Regenerar certificado o usar `-k` en curl
Puerto ya en uso	Otro proceso en :80/:443	`netstat -tulpn | grep :80`
`connection refused`	Nombre de servicio incorrecto	Verificar nombres en docker-compose.yml
`413 Request Entity Too Large`	Archivo muy grande	`client_max_body_size 10M;` en Nginx

Recursos para Seguir Aprendiendo

Documentación oficial:

- 🟦 nginx.org/en/docs — Referencia completa Nginx
- 🟩 docs.docker.com — Documentación Docker
- 🟧 ssl-config.mozilla.org — Configuración TLS recomendada

Herramientas útiles:

- 🔍 ssllabs.com/ssltest — Auditar configuración SSL
- 🔒 hstspreload.org — HSTS Preload List
- 📊 securityheaders.com — Analizar headers de seguridad

¡Gracias!

🧪 Ahora toca practicar

Actividad 1: Stack WordPress con seguridad

Actividad 2: Balanceo de carga con Node.js

Recuerda: la documentación oficial es tu mejor aliada 📖

📝 Preguntas de Repaso Rápido

1. ¿Qué diferencia hay entre `CMD` y `ENTRYPOINT`?
2. ¿Por qué no usar `:latest` en producción?
3. ¿Qué hace `server_tokens off`?
4. ¿Cuándo usar `ip_hash` en lugar de `round_robin`?
5. ¿Qué previene HSTS exactamente?
6. ¿Qué hace `proxy_next_upstream`?
7. ¿Diferencia entre named volume y bind mount?
8. ¿Qué significa que HTTP es stateless?