# 🐳 Guía Completa: Instalación de Docker y Despliegue de EduStreaming

## 📚 Estudio de Caso: Implementación de Plataforma de Streaming Educativo

### 🎯 Contexto del Proyecto

Una universidad necesita implementar una plataforma de streaming para transmitir clases en vivo y bajo demanda a estudiantes remotos. La solución debe soportar hasta 500 conexiones simultáneas y ofrecer calidad adaptativa según el ancho de banda de cada usuario.

### 🏫 Información de Análisis del Proyecto de Aula

**Objetivos del Proyecto:**

- **Objetivo Principal**: Desarrollar una plataforma web de streaming educativo que permita la transmisión de clases en vivo y contenido bajo demanda
- **Objetivo Técnico**: Implementar una solución escalable usando tecnologías modernas (React, Docker, Nginx)
- **Objetivo Académico**: Demostrar competencias en desarrollo full-stack, containerización y despliegue de aplicaciones

**Requerimientos Funcionales:**

- ✅ **Sistema de Autenticación**: Login/registro de usuarios con roles (estudiante, profesor, admin)
- ✅ **Streaming en Vivo**: Transmisión de clases en tiempo real con chat interactivo
- ✅ **Contenido Bajo Demanda**: Biblioteca de clases grabadas con búsqueda avanzada
- ✅ **Sistema de Notificaciones**: Alertas para nuevas clases, tareas y recordatorios
- ✅ **Dashboard Administrativo**: Panel de control para profesores y administradores
- ✅ **Perfil de Usuario**: Gestión de información personal y progreso académico

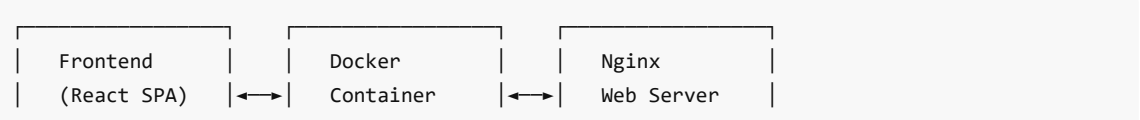
**Requerimientos No Funcionales:**

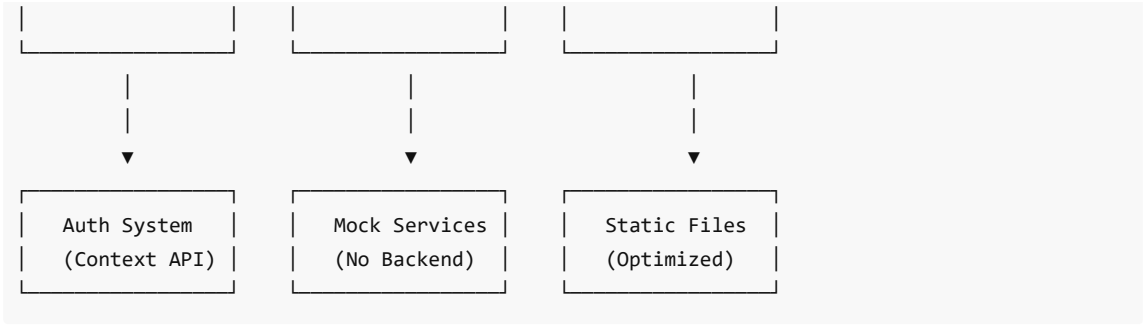
- **Escalabilidad**: Soporte para 500+ conexiones simultáneas
- **Rendimiento**: Tiempo de carga < 3 segundos
- **Disponibilidad**: 99.9% de uptime
- **Seguridad**: Autenticación segura y encriptación de datos
- **Usabilidad**: Interfaz intuitiva y responsive design
- **Compatibilidad**: Funcionamiento en múltiples navegadores y dispositivos

**Tecnologías Implementadas:**

- **Frontend**: React 18 + Vite + Material-UI + Styled Components
- **Containerización**: Docker + Docker Compose
- **Servidor Web**: Nginx con configuración optimizada
- **Estado Global**: Context API + React Query
- **Routing**: React Router DOM
- **Estilos**: Material-UI + Styled Components + CSS3

**Arquitectura de la Solución:**






**Casos de Uso Principales:**

1. **Estudiante Accede a Clase en Vivo**
    - Autenticación → Navegación → Selección de clase → Streaming → Chat
2. **Profesor Inicia Transmisión**
    - Login → Dashboard → Configuración → Inicio de stream → Monitoreo
3. **Administrador Gestiona Contenido**
    - Login → Panel admin → Gestión de usuarios → Configuración → Reportes
4. **Usuario Busca Contenido**
    - Búsqueda → Filtros → Resultados → Reproducción → Favoritos

**Métricas de Rendimiento Objetivo:**

- **Tiempo de Carga Inicial**: < 3 segundos
- **Tiempo de Respuesta de API**: < 500ms
- **Throughput**: 500+ usuarios simultáneos
- **Disponibilidad**: 99.9% uptime
- **Tiempo de Recuperación**: < 5 minutos

**Consideraciones de Seguridad:**

- **Autenticación JWT**: Tokens seguros para sesiones
- **HTTPS**: Encriptación de datos en tránsito
- **CORS**: Configuración de políticas de origen cruzado
- **Validación**: Sanitización de inputs del usuario
- **Headers de Seguridad**: CSP, XSS Protection, etc.

**Estrategia de Despliegue:**

- **Desarrollo**: Hot reload con Docker Compose
- **Producción**: Multi-stage build optimizado
- **Monitoreo**: Health checks y logging centralizado
- **Escalabilidad**: Horizontal scaling con load balancer

**Beneficios de la Implementación:**

- **Para la Universidad**: Reducción de costos de infraestructura física
- **Para los Estudiantes**: Acceso flexible y contenido bajo demanda
- **Para los Profesores**: Herramientas avanzadas de enseñanza
- **Para la Institución**: Escalabilidad y mantenimiento simplificado

**Lecciones Aprendidas:**

- **Containerización**: Simplifica el despliegue y la escalabilidad
- **SPA Architecture**: Mejora la experiencia de usuario
- **Mock Services**: Permite desarrollo frontend independiente
- **Nginx Configuration**: Optimiza el rendimiento y la seguridad

**Próximos Pasos del Proyecto:**

1. **Fase 2**: Implementación de backend real (Node.js/Express)
2. **Fase 3**: Integración con base de datos (PostgreSQL/MongoDB)
3. **Fase 4**: Sistema de streaming real (WebRTC/RTMP)
4. **Fase 5**: Análisis de datos y machine learning

---

## 📋 Tabla de Contenidos



---

## 🖥️ Prerrequisitos del Sistema

### Requisitos Mínimos:

- **Sistema Operativo**: Windows 10/11, macOS 10.15+, o Linux
- **RAM**: Mínimo 4GB (Recomendado: 8GB+)
- **Espacio en Disco**: 2GB libres
- **Procesador**: 64-bit con soporte para virtualización
- **Conexión a Internet**: Para descargar imágenes de Docker

### Verificar Virtualización:

- **Windows**: Verificar que Hyper-V esté habilitado
- **macOS**: Verificar que VirtualBox o VMware estén instalados
- **Linux**: Verificar que KVM esté disponible

---

## 🚀 Instalación de Docker Desktop

### Para Windows:

#### Paso 1: Descargar Docker Desktop

1. Visita: https://www.docker.com/products/docker-desktop/
2. Haz clic en "Download for Windows"
3. Descarga el archivo `Docker Desktop Installer.exe`

#### Paso 2: Instalar Docker Desktop

1. **Ejecutar como Administrador**: Haz clic derecho en el instalador y selecciona "Ejecutar como administrador"
2. **Aceptar términos**: Marca la casilla "I accept the terms" y haz clic en "Install"

3. **Configuración inicial**:
    - ✅ Use WSL 2 instead of Hyper-V (recomendado)
    - ✅ Add shortcut to desktop
    - ✅ Use Windows containers for Linux containers

**Paso 3: Reiniciar el Sistema**

- Reinicia tu computadora cuando se solicite
- Esto es necesario para que los cambios de virtualización tomen efecto

**Paso 4: Configurar Docker Desktop**

1. **Abrir Docker Desktop**: Busca "Docker Desktop" en el menú inicio
2. **Aceptar términos de servicio**: Lee y acepta los términos
3. **Configuración de recursos**:
    - Ve a Settings (⚙️) → Resources
    - **Memory**: Asigna al menos 4GB (recomendado: 6-8GB)
    - **CPUs**: Asigna al menos 2 cores
    - **Disk image size**: Al menos 60GB

## Para macOS:

**Paso 1: Descargar Docker Desktop**

1. Visita: https://www.docker.com/products/docker-desktop/
2. Selecciona "Download for Mac"
3. Descarga el archivo `.dmg` apropiado para tu procesador (Intel o Apple Silicon)

**Paso 2: Instalar Docker Desktop**

1. **Montar la imagen**: Doble clic en el archivo `.dmg` descargado
2. **Arrastrar a Aplicaciones**: Arrastra el ícono de Docker a la carpeta Applications
3. **Ejecutar Docker Desktop**: Abre Docker Desktop desde Applications

**Paso 3: Configurar Docker Desktop**

1. **Aceptar términos**: Acepta los términos de servicio
2. **Configuración de recursos**:
    - Ve a Settings (⚙️) → Resources
    - **Memory**: Asigna al menos 4GB
    - **CPUs**: Asigna al menos 2 cores

## Para Linux (Ubuntu/Debian):

**Paso 1: Actualizar el sistema**

```
sudo apt update
sudo apt upgrade -y
```

**Paso 2: Instalar dependencias**

```
sudo apt install -y apt-transport-https ca-certificates curl gnupg lsb-release
```

**Paso 3: Agregar clave GPG de Docker**

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o
/usr/share/keyrings/docker-archive-keyring.gpg
```

**Paso 4: Agregar repositorio de Docker**

```
echo "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg]
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo tee
/etc/apt/sources.list.d/docker.list > /dev/null
```

**Paso 5: Instalar Docker**

```
sudo apt update
sudo apt install -y docker-ce docker-ce-cli containerd.io docker-compose-plugin
```

**Paso 6: Agregar usuario al grupo docker**

```
sudo usermod -aG docker $USER
newgrp docker
```

---

## ✅ Verificación de la Instalación

### Verificar Docker Engine:

```
docker --version
# Debería mostrar: Docker version 24.x.x, build xxxxx
```

### Verificar Docker Compose:

```
docker-compose --version
# Debería mostrar: Docker Compose version v2.x.x
```

### Verificar que Docker esté funcionando:

```
docker run hello-world
# Debería mostrar: "Hello from Docker!"
```

### Verificar Docker Desktop (Windows/macOS):

1. Abre Docker Desktop
2. Verifica que el estado sea "Running" (Verde)
3. Ve a la pestaña "Images" - debería estar vacía inicialmente

---

## 📁 Configuración del Proyecto

### Estructura del Proyecto:

```
edustreaming/
├── src/                    # Código fuente de la aplicación
├── public/                 # Archivos públicos
├── package.json           # Dependencias del proyecto
├── vite.config.js         # Configuración de Vite
├── Dockerfile             # Configuración para producción
├── Dockerfile.dev         # Configuración para desarrollo
├── docker-compose.yml     # Orquestación de servicios
├── nginx.conf             # Configuración del servidor web
├── deploy.sh              # Script de despliegue (Linux/Mac)
├── deploy.bat             # Script de despliegue (Windows)
└── .dockerignore          # Archivos a ignorar en Docker
```

### Archivos de Configuración Docker:

**Dockerfile (Producción):**

- **Multi-stage build** para optimización
- **Node.js 18 Alpine** para construcción
- **Nginx Alpine** para servidor web
- **Usuario no-root** para seguridad

**Dockerfile.dev (Desarrollo):**

- **Hot reload** habilitado
- **Puerto 5173** para Vite
- **Volúmenes montados** para cambios en tiempo real

**docker-compose.yml:**

- **Servicio de producción** en puerto 3000
- **Servicio de desarrollo** en puerto 5173
- **Health checks** configurados
- **Redes personalizadas**

---

## 🚀 Despliegue de la Aplicación

### Opción 1: Usando Scripts (Recomendado)

**En Windows:**

```
# Navegar al directorio del proyecto
cd C:\ruta\a\tu\proyecto\edustreaming

# Modo desarrollo (con hot reload)
deploy.bat dev

# Modo producción
deploy.bat prod

# Ver logs
```

```
deploy.bat logs

# Detener aplicación
deploy.bat stop
```

**En Linux/macOS:**

```
# Navegar al directorio del proyecto
cd /ruta/a/tu/proyecto/edustreaming

# Hacer ejecutable el script
chmod +x deploy.sh

# Modo desarrollo (con hot reload)
./deploy.sh dev

# Modo producción
./deploy.sh prod

# Ver logs
./deploy.sh logs

# Detener aplicación
./deploy.sh stop
```

### Opción 2: Usando Docker Compose Directamente

**Modo Desarrollo:**

```
# Construir y levantar en modo desarrollo
docker-compose --profile dev up -d edustreaming-dev

# La aplicación estará disponible en: http://localhost:5173
```

**Modo Producción:**

```
# Construir y levantar en modo producción
docker-compose up -d edustreaming

# La aplicación estará disponible en: http://localhost:3000
```

### Proceso de Construcción:

1. **Descarga de imágenes base**: Docker descarga Node.js y Nginx
2. **Instalación de dependencias**: npm install ejecuta automáticamente
3. **Construcción de la aplicación**: Vite build genera archivos optimizados
4. **Configuración de Nginx**: Servidor web configurado para SPA
5. **Inicio del contenedor**: Aplicación lista para usar

---

## 🛠️ Comandos Útiles

### Comandos de Docker:

```
# Ver contenedores corriendo
docker ps

# Ver todas las imágenes
docker images

# Ver logs de un contenedor
docker logs <container_name>

# Detener un contenedor
docker stop <container_name>

# Eliminar un contenedor
docker rm <container_name>

# Eliminar una imagen
docker rmi <image_name>

# Limpiar sistema Docker
docker system prune -a
```

### Comandos de Docker Compose:

```
# Levantar servicios
docker-compose up -d

# Detener servicios
docker-compose down

# Ver logs
docker-compose logs -f

# Reconstruir servicios
docker-compose build --no-cache

# Ver estado de servicios
docker-compose ps
```

### Comandos del Script de Despliegue:

```
# Desarrollo
./deploy.sh dev          # Linux/Mac
deploy.bat dev           # Windows
```

```
# Producción
./deploy.sh prod          # Linux/Mac
deploy.bat prod           # Windows

# Construir solo
./deploy.sh build         # Linux/Mac
deploy.bat build          # Windows

# Detener
./deploy.sh stop          # Linux/Mac
deploy.bat stop           # Windows

# Limpiar
./deploy.sh clean         # Linux/Mac
deploy.bat clean          # Windows

# Ver logs
./deploy.sh logs          # Linux/Mac
deploy.bat logs           # Windows

# Verificar salud
./deploy.sh health        # Linux/Mac
deploy.bat health         # Windows
```

---

## 🔧 Solución de Problemas

### Problema: "Docker no está instalado"

**Solución:**

1. Verificar que Docker Desktop esté instalado
2. Reiniciar Docker Desktop
3. Verificar que el servicio esté corriendo

### Problema: "Puerto ya en uso"

**Solución:**

```
# Verificar qué está usando el puerto
netstat -tulpn | grep :3000    # Linux
netstat -an | findstr :3000    # Windows

# Detener contenedores y limpiar
docker-compose down
docker system prune -f
```

### Problema: "Error de permisos"

**Solución:**

```
# En Linux/Mac, dar permisos al script
chmod +x deploy.sh

# En Windows, ejecutar como administrador
# Clic derecho en PowerShell → "Ejecutar como administrador"
```

### Problema: "Imagen no se construye"

**Solución:**

```
# Limpiar cache de Docker
docker system prune -a

# Reconstruir sin cache
docker-compose build --no-cache
```

### Problema: "Aplicación no responde"

**Solución:**

```
# Verificar logs
docker-compose logs

# Verificar estado de contenedores
docker-compose ps

# Reiniciar servicios
docker-compose restart
```

### Problema: "Error de memoria insuficiente"

**Solución:**

1. Aumentar memoria asignada a Docker Desktop
2. Cerrar otras aplicaciones que consuman memoria
3. Reiniciar Docker Desktop

### Problema: "Error de virtualización"

**Solución:**

1. **Windows**: Habilitar Hyper-V o WSL 2
2. **macOS**: Verificar que VirtualBox esté instalado
3. **Linux**: Verificar que KVM esté disponible

---

## 🌐 Acceso a la Aplicación

### URLs de Acceso:

**Modo Desarrollo:**

- **URL Principal**: http://localhost:5173

- **Características**:
  - Hot reload automático
  - Herramientas de desarrollo
  - Debugging habilitado
  - Cambios en tiempo real

**Modo Producción:**

- **URL Principal**: http://localhost:3000
- **Características**:
  - Optimizada para producción
  - Compresión gzip
  - Cache optimizado
  - Configuración de seguridad

### Endpoints Adicionales:

**Health Check:**

- **URL**: http://localhost:3000/health
- **Respuesta**: "healthy"
- **Propósito**: Verificar que la aplicación esté funcionando

**Archivos Estáticos:**

- **CSS**: http://localhost:3000/assets/[filename].css
- **JavaScript**: http://localhost:3000/assets/[filename].js
- **Imágenes**: http://localhost:3000/assets/[filename].png

### Navegación en la Aplicación:

1. **Página Principal**: Catálogo de cursos y streams
2. **Búsqueda**: Funcionalidad de búsqueda avanzada
3. **Notificaciones**: Sistema de notificaciones
4. **Perfil**: Gestión de usuario
5. **Dashboard**: Panel de administración (para profesores/admin)

---

## 📊 Monitoreo y Mantenimiento

### Verificar Estado de la Aplicación:

```
# Estado de contenedores
docker-compose ps

# Uso de recursos
docker stats

# Logs en tiempo real
docker-compose logs -f
```

### Actualizar la Aplicación:

```
# Detener servicios
docker-compose down

# Actualizar código
git pull origin main

# Reconstruir y levantar
docker-compose up -d --build
```

### Backup y Restauración:

```
# Crear backup de volúmenes
docker run --rm -v edustreaming_data:/data -v $(pwd):/backup alpine tar czf
/backup/backup.tar.gz -C /data .

# Restaurar backup
docker run --rm -v edustreaming_data:/data -v $(pwd):/backup alpine tar xzf
/backup/backup.tar.gz -C /data
```

---

## 🎯 Próximos Pasos

### Para Desarrollo:

1. **Configurar IDE**: Instalar extensiones de Docker
2. **Debugging**: Configurar breakpoints en el código
3. **Testing**: Implementar tests automatizados
4. **CI/CD**: Configurar pipeline de integración continua

### Para Producción:

1. **Dominio**: Configurar dominio personalizado
2. **SSL**: Implementar certificados SSL
3. **Load Balancer**: Configurar balanceador de carga
4. **Monitoring**: Implementar herramientas de monitoreo

### Para Escalabilidad:

1. **Kubernetes**: Migrar a orquestación con Kubernetes
2. **Microservicios**: Dividir en microservicios
3. **Base de Datos**: Implementar base de datos externa
4. **Cache**: Implementar sistema de cache distribuido

---

## 📞 Soporte y Recursos

### Documentación Oficial:

- **Docker**: https://docs.docker.com/
- **Docker Compose**: https://docs.docker.com/compose/
- **Vite**: https://vitejs.dev/
- **React**: https://reactjs.org/

### Comunidad:

- **Docker Community**: https://forums.docker.com/
- **Stack Overflow**: https://stackoverflow.com/questions/tagged/docker
- **GitHub Issues**: Reportar problemas en el repositorio del proyecto

### Recursos Adicionales:

- **Docker Hub**: https://hub.docker.com/
- **Best Practices**: https://docs.docker.com/develop/dev-best-practices/
- **Security**: https://docs.docker.com/engine/security/

---

## ✅ Checklist de Verificación

### Antes de Empezar:

- [ ] Docker Desktop instalado y funcionando
- [ ] Docker Compose disponible
- [ ] Proyecto clonado localmente
- [ ] Puertos 3000 y 5173 disponibles
- [ ] Al menos 4GB de RAM disponible

### Después del Despliegue:

- [ ] Aplicación accesible en http://localhost:3000 (producción)
- [ ] Aplicación accesible en http://localhost:5173 (desarrollo)
- [ ] Health check respondiendo correctamente
- [ ] Logs sin errores críticos
- [ ] Navegación funcionando correctamente

### Para Producción:

- [ ] Configuración de seguridad implementada
- [ ] Certificados SSL configurados
- [ ] Monitoreo configurado
- [ ] Backup strategy implementada
- [ ] Documentación actualizada

---

## 🎉 ¡Felicitaciones!

Has configurado exitosamente Docker y desplegado la aplicación EduStreaming.

**Recuerda:**

- Usa `deploy.bat dev` para desarrollo
- Usa `deploy.bat prod` para producción
- Monitorea los logs regularmente
- Mantén Docker Desktop actualizado

¡Disfruta de tu aplicación! 🚀

---