# RTPEngine

Instalación Nativa vs Docker

Guía Completa de Comparación y Mejores Prácticas

| | |
|---|---|
| **Fecha:** | 13 de November de 2025 |
| **Sistema:** | AlmaLinux 9 |
| **Versión:** | RTPEngine 11.5+ |
| **Audiencia:** | Ingenieros VoIP y DevOps |

# Tabla de Contenidos

# 1. Introducción

RTPEngine es un proxy RTP/RTCP de alto rendimiento utilizado en infraestructuras VoIP modernas. Desarrollado por Sipwise, es la solución preferida para manejar flujos de media en sistemas basados en Kamailio, OpenSIPS y otros servidores SIP.

Este documento proporciona un análisis exhaustivo de las dos formas principales de desplegar RTPEngine: instalación nativa en el sistema operativo versus contenedores Docker. El objetivo es ayudar a ingenieros VoIP y DevOps a tomar una decisión informada basada en sus necesidades específicas.

## Características Principales de RTPEngine:

- Forwarding de paquetes RTP/RTCP en kernel space (máxima performance)
- Soporte para WebRTC (ICE, DTLS-SRTP)
- Transcoding de códecs de audio y video
- Grabación de llamadas en tiempo real
- Soporte para IPv4/IPv6 dual-stack
- Integración nativa con Kamailio, OpenSIPS y FreeSWITCH
- Reescritura de SDP para NAT traversal
- Protocolo de control ng (bencode sobre UDP)

# 2. Comparación Técnica Detallada

## 2.1 Rendimiento (Performance)

**Instalación Nativa:**
La instalación nativa de RTPEngine ofrece el máximo rendimiento posible. El proceso se ejecuta directamente en el sistema operativo con acceso sin restricciones al módulo kernel xt_RTPENGINE, que realiza el forwarding de paquetes RTP en kernel space. Esto elimina cualquier overhead de virtualización o contenedorización.

**Métricas típicas:**
- Latencia: <1ms adicional por paquete RTP
- Capacidad: 5,000-10,000 llamadas concurrentes (hardware adecuado)
- CPU overhead: Mínimo
- Jitter: Mínimo y predecible

**Docker:**
Con Docker usando network_mode: host, el overhead es mínimo pero presente. El contenedor requiere privilegios elevados (--privileged) y montaje de /proc para acceder al módulo kernel. En pruebas reales, la diferencia de rendimiento es generalmente menor al 5% en la mayoría de escenarios.

**Métricas típicas:**
- Latencia: 1-2ms adicional por paquete RTP
- Capacidad: 4,500-9,500 llamadas concurrentes (hardware equivalente)
- CPU overhead: +3-5%
- Jitter: Ligeramente mayor pero aceptable

**Veredicto:** La instalación nativa gana en performance pura, especialmente en cargas muy altas (>5,000 llamadas concurrentes). Para cargas medias-bajas (<3,000 llamadas), Docker con network_mode: host ofrece rendimiento muy cercano al nativo.

## 2.2 Instalación y Configuración

**Instalación Nativa (AlmaLinux 9):**
El proceso de instalación nativa es complejo y requiere múltiples pasos:

1. Instalación de repositorios EPEL y dependencias de desarrollo
2. Compilación del código fuente desde el repositorio Git de Sipwise
3. Compilación del módulo kernel xt_RTPENGINE compatible con la versión del kernel
4. Configuración del servicio systemd
5. Creación y edición de archivos de configuración
6. Configuración de firewall y SELinux

7. Pruebas de funcionamiento

**Problemas comunes:** Dependencias rotas, incompatibilidad entre versión de kernel y módulo, errores de compilación por bibliotecas faltantes, actualizaciones de kernel que rompen el módulo.

**Instalación con Docker:**
El proceso con Docker es extremadamente simplificado. Requiere únicamente crear un archivo docker-compose.yml con la configuración deseada y ejecutar un comando. La imagen ya contiene todas las dependencias necesarias precompiladas.

**Ventajas:** Instalación en 5-10 minutos, sin dependencias complejas, sin compilación manual, configuración declarativa, funcionamiento consistente entre diferentes sistemas.

**Veredicto:** Docker gana por amplio margen en facilidad de instalación y puesta en marcha inicial.

## 2.3 Mantenimiento y Actualizaciones

**Instalación Nativa:**
El mantenimiento de una instalación nativa requiere atención constante. Cada actualización del sistema operativo, especialmente del kernel, puede romper el módulo xt_RTPENGINE, requiriendo recompilación. Las actualizaciones de RTPEngine implican descargar nuevo código fuente, recompilar, detener el servicio, instalar los binarios nuevos y reiniciar.

**Docker:**
Las actualizaciones son atómicas y seguras. Con un simple 'docker compose pull' seguido de 'docker compose up -d', se descarga la nueva imagen y se reinicia el contenedor. Si algo falla, el rollback es inmediato ejecutando la versión anterior de la imagen. No hay riesgo de dejar el sistema en estado inconsistente.

**Veredicto:** Docker gana significativamente en mantenimiento, actualizaciones y capacidad de rollback.

## 2.4 Portabilidad y Reproducibilidad

La portabilidad es crítica en entornos modernos donde se necesita replicar configuraciones entre desarrollo, staging y producción, o migrar servicios entre servidores.

**Instalación Nativa:**
Replicar una instalación nativa requiere documentación exhaustiva de todas las versiones de dependencias, configuraciones del sistema, parámetros de compilación y ajustes del kernel. El síndrome "funciona en mi máquina" es común. Migrar a un servidor nuevo puede llevar horas o días.

**Docker:**
Con Docker, toda la configuración está contenida en archivos de texto (docker-compose.yml, Dockerfile). La misma imagen Docker funcionará idénticamente en cualquier servidor con Docker instalado. Se garantiza comportamiento consistente entre entornos. La migración es copiar archivos y ejecutar 'docker compose up'.

**Veredicto:** Docker gana totalmente en portabilidad y reproducibilidad de entornos.

## 2.5 Aislamiento y Seguridad

La seguridad es un aspecto complejo donde ambas opciones tienen ventajas y limitaciones.

**Instalación Nativa:**
El proceso RTPEngine ejecuta como root o con usuario dedicado, pero con acceso completo al sistema. Una vulnerabilidad en RTPEngine podría comprometer el servidor completo. Es difícil limitar recursos (CPU, memoria) sin herramientas adicionales como cgroups o systemd resource control.

**Docker:**
Los contenedores ofrecen aislamiento del filesystem y facilitan límites de recursos. Sin embargo, RTPEngine requiere el flag --privileged para acceder al módulo kernel, lo que reduce significativamente el aislamiento de seguridad. En la práctica, ambas opciones tienen riesgos similares.

**Veredicto:** Empate técnico. Docker ofrece mejor control de recursos pero RTPEngine necesita privilegios que eliminan parte del aislamiento.

## 2.6 Debugging y Troubleshooting

**Instalación Nativa:**
Ofrece acceso directo a todas las herramientas de debugging del sistema: strace, gdb, perf, systemtap, etc. Los logs están integrados con journald. Es posible hacer profiling detallado del proceso y análisis de rendimiento a bajo nivel. El acceso al módulo kernel para debugging es directo.

**Docker:**
Requiere entrar al contenedor para debugging profundo. Algunas herramientas de análisis no funcionan igual dentro de contenedores. Los logs se acceden vía 'docker logs'. Sin embargo, para debugging rutinario, Docker es suficiente y sus logs son más fáciles de centralizar en sistemas de logging modernos.

**Veredicto:** Nativa gana en debugging avanzado y herramientas de análisis de bajo nivel.

## 2.7 Recursos y Overhead

El consumo de recursos es importante, especialmente en servidores con múltiples servicios.

**Instalación Nativa:**
- Memoria: 50-100 MB base + memoria por llamada
- Disco: ~100 MB (binarios y bibliotecas)
- CPU: Dedicada 100% al proceso
- Overhead: Ninguno

**Docker:**
- Memoria: 50-100 MB (proceso) + 20-50 MB (overhead contenedor)
- Disco: 200-300 MB (imagen Docker)
- CPU: Similar con network_mode: host
- Overhead: ~5-10% memoria adicional

**Veredicto:** Nativa es ligeramente más eficiente en uso de recursos, pero la diferencia es marginal.

## 3. Tabla Comparativa Resumida

| Aspecto | Nativa | Docker | Ganador |
|---|---|---|---|
| **Performance** | ■■■■■ | ■■■■ | **Nativa** |
| **Instalación** | ■■ | ■■■■■ | **Docker** |
| **Mantenimiento** | ■■ | ■■■■■ | **Docker** |
| **Actualizaciones** | ■■ | ■■■■■ | **Docker** |
| **Portabilidad** | ■■ | ■■■■■ | **Docker** |
| **Debugging** | ■■■■■ | ■■■ | **Nativa** |
| **Seguridad** | ■■■ | ■■■ | **Empate** |
| **Recursos** | ■■■■■ | ■■■■ | **Nativa** |
| **Escalabilidad** | ■■■ | ■■■■ | **Docker** |
| **Reproducibilidad** | ■■ | ■■■■■ | **Docker** |

**Conclusión de la tabla:** Docker gana en 6 de 10 categorías, con énfasis en aspectos operacionales (instalación, mantenimiento, portabilidad). Nativa gana en aspectos de rendimiento puro y debugging avanzado.

# 4. Instalación Paso a Paso

## 4.1 Instalación con Docker en AlmaLinux 9

Esta guía cubre la instalación completa de RTPEngine usando Docker en AlmaLinux 9, desde la preparación del sistema hasta la verificación del funcionamiento.

### Paso 1: Preparar el Sistema

```
# Actualizar el sistema sudo dnf update -y # Instalar dependencias del kernel sudo dnf install
-y kernel-devel kernel-modules-extra # Cargar módulo xt_RTPENGINE sudo modprobe xt_RTPENGINE
echo "xt_RTPENGINE" | sudo tee /etc/modules-load.d/rtpengine.conf # Verificar módulo lsmod |
grep xt_RTPENGINE
```

### Paso 2: Instalar Docker

```
# Agregar repositorio Docker sudo dnf config-manager --add-repo
https://download.docker.com/linux/centos/docker-ce.repo # Instalar Docker sudo dnf install -y
docker-ce docker-ce-cli containerd.io docker-compose-plugin # Iniciar y habilitar Docker sudo
systemctl start docker sudo systemctl enable docker
```

### Paso 3: Configurar Firewall

```
# Puerto de control ng sudo firewall-cmd --permanent --add-port=22222/udp # Rango de puertos
RTP sudo firewall-cmd --permanent --add-port=10000-20000/udp # Recargar firewall sudo
firewall-cmd --reload
```

### Paso 4: Crear docker-compose.yml

```
mkdir -p ~/rtpengine-docker && cd ~/rtpengine-docker cat > docker-compose.yml << 'EOF'
services: rtpengine: image: drachtio/rtpengine:latest container_name: rtpengine restart:
unless-stopped network_mode: host privileged: true volumes: - /proc:/proc command: - rtpengine
- --interface=eth0 - --listen-ng=0.0.0.0:22222 - --port-min=10000 - --port-max=20000 -
--log-level=6 - --num-threads=4 EOF
```

### Paso 5: Iniciar RTPEngine

```
# Iniciar contenedor sudo docker compose up -d # Ver logs sudo docker compose logs -f #
Verificar estado sudo docker compose ps
```

### Paso 6: Verificar Funcionamiento

```
# Verificar puerto ng sudo ss -tulpn | grep 22222 # Probar conectividad echo -n "12345
d4:ping=1e" | nc -u -w1 127.0.0.1 22222 # Respuesta esperada: 12345 d4:pong=1e
```

## 4.2 Instalación Nativa en AlmaLinux 9

La instalación nativa requiere compilación desde el código fuente. Este proceso es más complejo pero ofrece máximo control y rendimiento.

### Paso 1: Instalar Dependencias

```
# Repositorios sudo dnf install -y epel-release # Herramientas de desarrollo sudo dnf groupinstall -y "Development Tools" # Bibliotecas necesarias sudo dnf install -y kernel-devel glib2-devel zlib-devel \ openssl-devel pcre-devel libcurl-devel xmlrpc-c-devel \ hiredis-devel json-glib-devel libevent-devel iptables-devel \ libpcap-devel mariadb-devel ffmpeg-devel spandsp-devel
```

### Paso 2: Descargar y Compilar RTPEngine

```
# Clonar repositorio git clone https://github.com/sipwise/rtpengine.git cd rtpengine # Compilar daemon cd daemon make sudo make install # Compilar módulo kernel cd ../kernel-module make sudo make install # Cargar módulo sudo modprobe xt_RTPENGINE
```

### Paso 3: Configurar Servicio

```
# Crear archivo de configuración sudo mkdir -p /etc/rtpengine sudo vi /etc/rtpengine/rtpengine.conf # Contenido básico: # interface = eth0 # listen-ng = 0.0.0.0:22222 # port-min = 10000 # port-max = 20000 # log-level = 6 # Configurar systemd sudo cp rtpengine.service /etc/systemd/system/ sudo systemctl daemon-reload sudo systemctl enable rtpengine sudo systemctl start rtpengine
```

# 5. Escalado de RTPEngine

RTPEngine presenta desafíos únicos para el escalado debido a su naturaleza de procesamiento de media en tiempo real y su dependencia del módulo kernel.

## Limitaciones de Escalado:

- Requiere network_mode: host - no puede usar redes bridge de Docker
- Cada instancia necesita su propio rango de puertos RTP/RTCP
- El puerto de control ng (22222) debe ser único por instancia
- Mantiene estado de sesiones - no es completamente stateless
- Dependencia del módulo kernel xt_RTPENGINE

## Estrategias de Escalado:

**1. Escalado Vertical:** Aumentar CPU, RAM y ancho de banda del servidor donde corre RTPEngine. Es la opción más simple y efectiva hasta cierto punto.

**2. Múltiples Instancias en el Mismo Host:** Ejecutar 2-3 instancias de RTPEngine en el mismo servidor, cada una con su puerto ng y rango RTP único. Kamailio balancea usando hash-callid.

**3. Escalado Horizontal:** Desplegar RTPEngine en múltiples servidores físicos o VMs. Usar Kamailio dispatcher para distribuir llamadas entre instancias. Cada servidor ejecuta una instancia de RTPEngine en el puerto estándar.

**4. Docker Swarm:** Usar modo global con placement constraints para una instancia por nodo. Agregar nodos al swarm para escalar horizontalmente.

**5. Kubernetes:** DaemonSet con hostNetwork=true para despliegue en cada nodo del cluster. Más complejo pero ofrece orchestration avanzada.

## Ejemplo: Múltiples Instancias con Docker

Para ejecutar 3 instancias de RTPEngine en el mismo host, cada una debe tener configuración única:

```
# docker-compose.yml con 3 instancias services: rtpengine-1: image: drachtio/rtpengine:latest
container_name: rtpengine-1 network_mode: host privileged: true volumes: ["/proc:/proc"]
command: ["rtpengine", "--interface=eth0", "--listen-ng=0.0.0.0:22222", "--port-min=10000",
"--port-max=15000"] rtpengine-2: image: drachtio/rtpengine:latest container_name: rtpengine-2
network_mode: host privileged: true volumes: ["/proc:/proc"] command: ["rtpengine",
"--interface=eth0", "--listen-ng=0.0.0.0:22223", # Puerto diferente "--port-min=15001",
"--port-max=20000"] # Rango diferente rtpengine-3: image: drachtio/rtpengine:latest
container_name: rtpengine-3 network_mode: host privileged: true volumes: ["/proc:/proc"]
command: ["rtpengine", "--interface=eth0", "--listen-ng=0.0.0.0:22224", # Puerto diferente
```

```
"--port-min=20001", "--port-max=25000"] # Rango diferente
```

# 6. Integración con Kamailio

Kamailio es el servidor SIP más común para integrar con RTPEngine. La integración se realiza mediante el módulo rtpengine.so que implementa el protocolo ng.

## Configuración Básica:

```
# kamailio.cfg # Cargar módulo loadmodule "rtpengine.so" # Configurar socket de RTPEngine
(instancia única) modparam("rtpengine", "rtpengine_sock", "udp:127.0.0.1:22222") # Timeouts
modparam("rtpengine", "rtpengine_timeout", "1") modparam("rtpengine", "rtpengine_retr", 3) #
Routing para manejo de media route[RTPENGINE_MANAGE] { if (is_method("INVITE|UPDATE")) { #
Ofrecer SDP modificado rtpengine_offer("replace-origin replace-session-connection"); } if
(is_method("ACK")) { # Procesar respuesta SDP rtpengine_answer("replace-origin
replace-session-connection"); } if (is_method("BYE|CANCEL")) { # Liberar recursos
rtpengine_delete(); } }
```

## Configuración con Múltiples Instancias:

```
# Configurar múltiples sockets de RTPEngine modparam("rtpengine", "rtpengine_sock",
"udp:127.0.0.1:22222") modparam("rtpengine", "rtpengine_sock", "udp:127.0.0.1:22223")
modparam("rtpengine", "rtpengine_sock", "udp:127.0.0.1:22224") # Algoritmo de balanceo #
round-robin: rotación simple # hash-callid: mismo RTPEngine para mismo call-id (recomendado) #
hash-from-tag: hash basado en from-tag modparam("rtpengine", "rtpengine_algo", "hash-callid")
# Kamailio automáticamente distribuirá llamadas entre las 3 instancias
```

## Configuración Avanzada para WebRTC:

```
route[RTPENGINE_WEBRTC] { if (is_method("INVITE")) { if ($ua =~ "WebRTC") { # Cliente es
WebRTC rtpengine_offer("RTP/AVP replace-origin replace-session-connection ICE=remove
DTLS=off"); } else { # Cliente es SIP tradicional rtpengine_offer("replace-origin
replace-session-connection"); } } if (is_method("ACK") && has_body("application/sdp")) {
rtpengine_answer("replace-origin replace-session-connection"); } }
```

# 7. Recomendaciones Finales

## Usa Docker cuando:

✓ Instalación rápida es prioritaria (minutos vs horas)

✓ Infraestructura moderna/cloud-native

✓ Múltiples entornos (dev/staging/prod) que deben ser idénticos

✓ Actualizaciones frecuentes del software

✓ Equipo con experiencia en contenedores pero no en compilación

✓ Integración con CI/CD pipelines

✓ Testing constante de nuevas versiones

✓ Carga de trabajo: <3,000 llamadas concurrentes

✓ Facilidad de rollback es importante

✓ Portabilidad entre diferentes sistemas

## Usa Instalación Nativa cuando:

✓ Servidor dedicado exclusivamente para RTPEngine

✓ Máximo rendimiento es crítico (>5,000 llamadas concurrentes)

✓ Equipo domina compilación y administración de Linux a bajo nivel

✓ Infraestructura tradicional (no cloud-native)

✓ Debugging profundo es frecuentemente necesario

✓ No se planean cambios de versión frecuentes

✓ Servidor de producción crítica donde cada milisegundo cuenta

✓ Políticas organizacionales contra contenedores

✓ Necesitas personalización extrema del código fuente

✓ Hardware especializado o optimizaciones específicas del kernel

## Recomendación Específica para AlmaLinux 9:

Para entornos basados en AlmaLinux 9, se recomienda **Docker** por las siguientes razones específicas:

• AlmaLinux 9 tiene dependencias más recientes que pueden causar conflictos de compilación
• Las actualizaciones de kernel en RHEL-based distributions pueden romper módulos compilados
• Docker abstrae estas complejidades del sistema operativo

• La mayoría de casos de uso VoIP no requieren el último 5% de performance que ofrece nativa
• El overhead de Docker con network_mode: host es prácticamente imperceptible

**Excepción:** Si ya tienes experiencia previa compilando RTPEngine en CentOS/RHEL y tu carga supera las 5,000 llamadas concurrentes, considera instalación nativa.

# 8. Arquitectura Híbrida

Una estrategia óptima para muchas organizaciones es combinar ambos enfoques según el entorno y las necesidades específicas.

## Patrón Recomendado:

**Desarrollo y Staging: Docker**
• Instalación rápida para developers
• Entornos consistentes y reproducibles
• Fácil testing de nuevas versiones
• Rollback instantáneo si algo falla
• Ideal para CI/CD

**Producción: Evaluar según carga**
• Carga baja-media (<3,000 llamadas): Docker
• Carga alta (>5,000 llamadas): Nativa
• Carga media (3,000-5,000): Probar ambas y medir

Esta estrategia permite desarrollo ágil mientras se optimiza la producción según necesidades reales.

## Ejemplo de Arquitectura:

**Equipo de Desarrollo:**
Laptop con Docker → RTPEngine containerizado → Pruebas locales rápidas

**Servidor Staging:**
Docker Compose con Kamailio + RTPEngine → Testing de integración

**Producción Pequeña (<2,000 llamadas):**
Docker con docker-compose → Fácil mantenimiento

**Producción Grande (>5,000 llamadas):**
Instalación nativa en servidores dedicados → Máximo performance

**Ventaja:** El código y configuración se prueban en Docker durante desarrollo, luego se trasladan las configuraciones a instalación nativa en producción cuando es necesario.

## Migración Gradual:

Si empiezas con Docker y tu carga crece hasta requerir instalación nativa:

1. Mantén Docker en staging para continuar desarrollo ágil
2. Despliega instalación nativa en servidor nuevo de producción
3. Migra tráfico gradualmente (porcentaje de llamadas)
4. Monitorea métricas de rendimiento durante la migración
5. Mantén Docker como backup o para disaster recovery

Esta transición es más suave que migrar de nativa a Docker o viceversa partiendo de cero.

# Conclusión

La elección entre instalación nativa y Docker de RTPEngine no tiene una respuesta universal. Depende del contexto específico: tamaño de la organización, carga de tráfico, experiencia del equipo, y prioridades operacionales.

**Para la mayoría de casos de uso modernos, Docker es la opción recomendada** por su balance entre facilidad de uso, mantenibilidad y rendimiento suficiente. El overhead es mínimo con network_mode: host, y las ventajas operacionales superan ampliamente la pequeña pérdida de performance.

La instalación nativa mantiene su lugar en escenarios de alto rendimiento donde cada milisegundo cuenta y se tienen los recursos humanos y técnicos para gestionar su complejidad.

La arquitectura híbrida (Docker para dev/staging, evaluación caso por caso para producción) ofrece lo mejor de ambos mundos y es probablemente la estrategia más pragmática para organizaciones que están creciendo o evolucionando su infraestructura VoIP.

## Recursos Adicionales:

- Repositorio oficial: https://github.com/sipwise/rtpengine
- Documentación: https://github.com/sipwise/rtpengine/blob/master/README.md
- Imagen Docker: https://hub.docker.com/r/drachtio/rtpengine
- Kamailio rtpengine module: https://www.kamailio.org/docs/modules/stable/modules/rtpengine.html
- Lista de correo: https://lists.sipwise.com/listinfo/rtpengine-devel

*Este documento ha sido creado como guía técnica para la evaluación e implementación de RTPEngine en infraestructuras VoIP modernas. Para casos de uso específicos o configuraciones avanzadas, consulte la documentación oficial y la comunidad de Sipwise.*