Guía de Instalación - Sistema RMS-Shopify Integration

> **Documento técnico para BestBrands** Versión: 1.0 Fecha: Noviembre 2025

Tabla de Contenidos

---

1. Componentes del Sistema

1.1 Software que se Instala

El sistema RMS-Shopify Integration instala los siguientes componentes en el equipo local:

A. Aplicación Principal

```
Componente: FastAPI Application
Lenguaje: Python 3.13
Puerto: 8000 (configurable)
Tipo: Servicio web asíncrono
Estado: Debe estar corriendo 24/7
```

Funcionalidades:

- Motor de sincronización automática (APScheduler)
- API REST para monitoreo y control manual
- Sistema de detección de cambios cada 5 minutos
- Procesador de webhooks de Shopify
- Panel de administración web (Swagger UI)

B. Base de Datos y Cache

```
Componente: Redis
Versión: 7.x
Puerto: 6379
Propósito: Cache y cola de tareas asíncronas
Persistencia: Opcional (configurable)
```

Uso de Redis:

- Cache de productos para reducir queries a RMS
- Sistema de locks distribuidos (evita duplicación)
- Cola de trabajos en background
- Almacenamiento temporal de webhooks

C. Estructura de Archivos

```
C:\RMS-Shopify-Integration\  (o directorio de instalación)
│
├── app/                        # Código de la aplicación
│   ├── main.py                 # Punto de entrada
│   ├── api/                    # Endpoints REST
│   ├── core/                   # Configuración y scheduler
│   ├── db/                     # Acceso a RMS y Shopify
│   ├── services/               # Lógica de negocio
│   └── utils/                  # Utilidades
│
├── logs/                       # Logs de operación
│   ├── app.log                 # Log general
│   ├── app_errors.log          # Solo errores
│   └── app.json                # Formato JSON
│
├── checkpoint/                 # Estado de sincronización
│   └── checkpoint.json         # Última fecha de sync exitoso
│
├── checkpoints/                # Progreso de syncs en curso
│   └── sync_[id].json          # Checkpoint de progreso
│
├── .env                        # Configuración (CRÍTICO - contiene credenciales)
├── docker-compose.yml          # Configuración Docker
├── pyproject.toml              # Dependencias Python
└── README.md                   # Documentación
```

D. NO se Instala

```
 SQL Server (usa el RMS existente)
 Shopify (usa la tienda existente en la nube)
 Bases de datos adicionales
 Modificaciones a RMS
```

1.2 Métodos de Instalación

Opción A: Docker (Recomendado)

Ventajas:

- Instalación simplificada
- Aislamiento de dependencias
- Fácil actualización
- Portabilidad entre equipos

Requisitos:

- Docker Desktop para Windows
- 4 GB RAM disponible
- 10 GB espacio en disco

Opción B: Python Nativo

Ventajas:

- Mayor control
- Menor overhead de recursos
- Debug más directo

Requisitos:

- Python 3.13+ instalado
- Poetry (gestor de dependencias)
- Permisos de administrador

1.3 Servicios Opcionales

Servicio de Windows

Si se configura como servicio Windows:

```
Nombre: RMSShopifyIntegration
Tipo: Automático (inicio con Windows)
Usuario: Cuenta con acceso a RMS
Puerto: 8000
```

Ventajas:

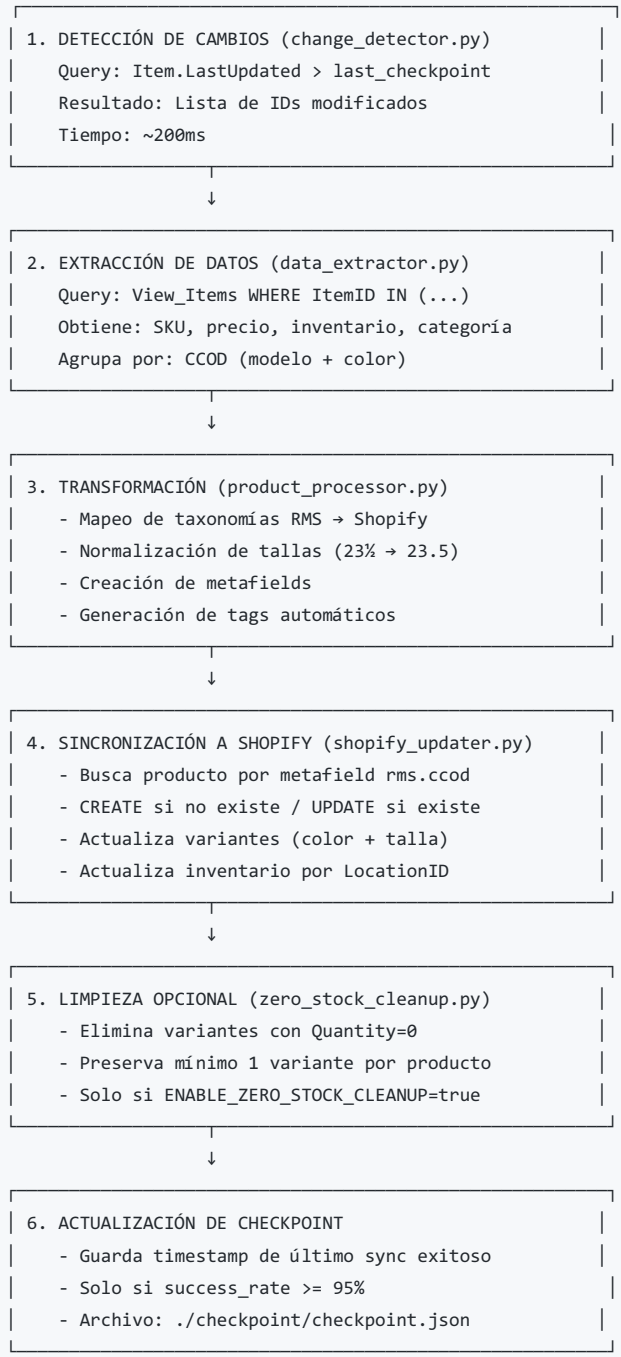
- Auto-inicio con el sistema
- Recuperación automática ante fallos
- Gestión vía Services.msc

---

2. ¿Qué Procesa el Sistema?

2.1 Flujo Principal: RMS → Shopify

Procesamiento Automático (Cada 5 Minutos)



Resultado del Procesamiento

Entrada (RMS):

```
ItemID: 12345
C_ARTICULO: 24YM05051-NEG-38
CCOD: 24YM05051
color: NEGRO
talla: 38
Price: 50.00
Quantity: 5
```

Salida (Shopify):

```
Product: "Zapato Deportivo Modelo 24YM05051"
├─ Variant: Negro / 38
│  ├─ SKU: 24YM05051-NEG-38
│  ├─ Price: $50.00
│  └─ Inventory: 5 units
├─ Metafields:
│  ├─ rms.ccod: "24YM05051"
│  ├─ rms.color: "NEGRO"
│  └─ rms.talla: "38"
└─ Tags: ["RMS-SYNC-2025-01-11", "24YM05051"]
```

2.2 Flujo Secundario: Shopify → RMS

Procesamiento en Tiempo Real (Webhooks)

```
┌─────────────────────────────────────────────────────┐
│ 1. RECEPCIÓN DE WEBHOOK (webhooks.py)               │
│    Shopify envía: POST /api/v1/webhooks/orders/create │
│    Evento: Nueva orden creada                       │
│    Validación: HMAC signature                       │
└─────────────────────┬───────────────────────────────┘
                      ↓
┌─────────────────────────────────────────────────────┐
│ 2. VALIDACIÓN (order_validator.py)                  │
│    - Verifica financial_status (PAID, AUTHORIZED)   │
│    - Valida productos existen en RMS                │
│    - Soporta pedidos de invitados                   │
└─────────────────────┬───────────────────────────────┘
                      ↓
┌─────────────────────────────────────────────────────┐
│ 3. RESOLUCIÓN DE CLIENTE (customer_resolver.py)     │
│    - Busca cliente por email en RMS                 │
│    - Auto-crea "SHOPIFY-GUEST" si no existe         │
│    - Maneja direcciones de envío                    │
└─────────────────────┬───────────────────────────────┘
                      ↓
┌─────────────────────────────────────────────────────┐
│ 4. CONVERSIÓN (order_converter.py)                  │
│    - Convierte estructura Shopify → RMS             │
│    - Discrimina impuestos (IVA 13%)                 │
│    - Mapea SKUs → ItemIDs de RMS                    │
└─────────────────────┬───────────────────────────────┘
                      ↓
┌─────────────────────────────────────────────────────┐
│ 5. INSERCIÓN EN RMS (order_creator.py)              │
│    INSERT INTO [Order] VALUES (...)                 │
│    INSERT INTO [OrderEntry] VALUES (...)            │
│    - Transacción atómica (rollback si falla)        │
└─────────────────────────────────────────────────────┘
```

Resultado del Procesamiento

Entrada (Shopify):

```
{
  "order": {
    "id": 5678901234,
    "email": "cliente@example.com",
    "total_price": "113.00",
    "line_items": [
      {
        "sku": "24YM05051-NEG-38",
        "quantity": 1,
        "price": "100.00"
      }
    ]
  }
}
```

Salida (RMS):

```
INSERT INTO [Order] (
  CustomerID, StoreID, Total, Tax, Comment
) VALUES (
  123,        -- ID del cliente
  40,         -- Tienda virtual
  100.00,     -- Total sin impuesto
  13.00,      -- Impuesto discriminado
  'Shopify Order #5678901234'
)

INSERT INTO [OrderEntry] (
  OrderID, ItemID, Quantity, Price
) VALUES (
  99999,   -- ID de orden generado
  12345,   -- ItemID de RMS
  1,       -- Cantidad
  100.00   -- Precio unitario sin impuesto
)
```

2.3 Full Sync Nocturna

```
Hora: 2:00 AM (configurable)
Frecuencia: Diaria
Propósito: Sincronización completa del catálogo
Duración: ~2-4 horas (según catálogo)

Proceso:
1. Ignora checkpoint (procesa TODO el catálogo)
2. Sincroniza todos los productos activos
3. Actualiza inventarios completos
4. Genera reporte de discrepancias
```

---

3. ¿Necesita Procesamiento Local?

Respuesta: SÍ - Procesamiento Local Significativo

El sistema **NO es solo un "lector"** de RMS. Es un **motor de sincronización bidireccional activo** que requiere procesamiento constante.

3.1 Razones para Procesamiento Local

A. Motor de Detección de Cambios

```
Ejecución: Cada 5 minutos, 24/7
Operación: Query activa a RMS
SQL: SELECT * FROM Item WHERE LastUpdated > :last_sync
Procesamiento: Comparación, filtrado, ordenamiento
```

NO es lectura pasiva, es **monitoreo activo permanente**.

B. Transformación de Datos

```
# Ejemplo de transformación
Entrada RMS:
  talla: "23½"
  color: "NEGRO"
  CCOD: "24YM05051"

Procesamiento Local:
  - Normaliza: "23½" → "23.5"
  - Agrupa por CCOD para crear variantes
  - Mapea categoría: "Tenis" → "Shoes > Sneakers"
  - Genera metafields estructurados

Salida Shopify:
  options: [
    {name: "Color", values: ["Negro"]},
    {name: "Talla", values: ["23.5"]}
  ]
```

C. Orquestación de Sincronización

```
Lógica Compleja:
✓ Batch processing (25-100 productos/vez)
✓ Manejo de errores y reintentos exponenciales
✓ Sistema de checkpoints para recuperación
✓ Control de rate limiting de Shopify
✓ Concurrencia y locks distribuidos
✓ Validación de datos pre-envío
```

D. Procesamiento de Webhooks

```
Disponibilidad: 24/7
Operaciones:
  - Recepción de webhooks HTTP POST
  - Validación HMAC signature
  - Conversión de formatos
  - Discriminación de impuestos
  - Escritura transaccional a RMS
  - Manejo de rollback ante errores
```

E. Estado y Cache

```
Redis (Local):
  - Cache de productos (TTL: 1 hora)
  - Locks distribuidos (previene duplicación)
  - Cola de trabajos en background

Checkpoint Files:
  - Estado de sincronización incremental
  - Progreso de syncs en curso
  - Histórico de operaciones

Logs:
  - 3 archivos de log rotados diariamente
  - Diagnóstico y auditoría
  - Métricas de performance
```

3.2 Recursos Necesarios

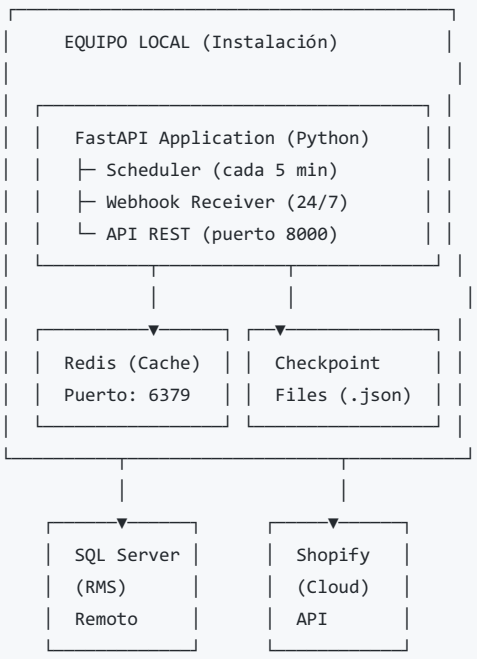
Mínimos

```
CPU: 2 cores
RAM: 4 GB
Disco: 20 GB disponible
Red: Conexión estable a RMS (LAN/VPN)
```

Recomendados

```
CPU: 4 cores
RAM: 8 GB
Disco: 50 GB SSD
Red: 10 Mbps uplink
```

3.3 Diagrama de Dependencias



3.4 Conclusión

El equipo local es **ESENCIAL** y debe:

- ✅ Estar encendido 24/7
- ✅ Tener conectividad constante a RMS
- ✅ Tener acceso a internet (Shopify)
- ✅ Ejecutar el servicio automáticamente

- ⚠️ Ser monitoreado regularmente

NO es opcional el procesamiento local porque:

1. RMS no puede enviar datos a Shopify directamente
2. Shopify no puede leer de RMS directamente
3. Se requiere transformación activa de datos
4. Los webhooks deben recibirse en tiempo real
5. La lógica de negocio es compleja y específica

---

4. Guía de Reinstalación

4.1 Escenario: Reemplazo de Equipo

Pre-requisitos del Nuevo Equipo

```
Sistema Operativo:
  - Windows 10/11 Professional
  - Windows Server 2016 o superior
  - Usuario con permisos de administrador

Conectividad:
  - Red con acceso a SQL Server RMS
  - Conexión a internet estable
  - Puerto 8000 disponible

Software Base:
  - Docker Desktop (Método A) o Python 3.13+ (Método B)
  - Git (opcional, para clonar código)
  - Acceso remoto (TeamViewer/AnyDesk) recomendado
```

4.2 Paso 1: Backup del Equipo Antiguo

```
# Ejecutar en PowerShell como Administrador
# En el equipo ANTIGUO

# 1. Crear directorio de backup
New-Item -ItemType Directory -Path "C:\Backups\rms-shopify-$(Get-Date -Format 'yyyyMMdd')" -Force
$BackupDir = "C:\Backups\rms-shopify-$(Get-Date -Format 'yyyyMMdd')"

# 2. Detener servicios
docker-compose down
# O si es servicio Windows:
# Stop-Service "RMSShopifyIntegration"

# 3. Respaldar archivo de configuración (CRÍTICO)
Copy-Item ".env" "$BackupDir\.env"
Write-Host "✅ Configuración respaldada"

# 4. Respaldar docker-compose.yml
Copy-Item "docker-compose.yml" "$BackupDir\docker-compose.yml"

# 5. Respaldar checkpoint (estado de sincronización)
Copy-Item ".\checkpoint\checkpoint.json" "$BackupDir\checkpoint.json"
Write-Host "✅ Checkpoint respaldado"

# 6. Respaldar logs recientes (diagnóstico)
Copy-Item ".\logs\*" "$BackupDir\logs\" -Recurse -Force
Write-Host "✅ Logs respaldados"

# 7. Documentar configuración actual
docker-compose ps > "$BackupDir\service-status.txt"
ipconfig > "$BackupDir\network-config.txt"
Get-Date > "$BackupDir\backup-timestamp.txt"

Write-Host "✅ Backup completo en: $BackupDir"
Write-Host "⚠️  IMPORTANTE: Copiar esta carpeta al nuevo equipo"
```

Archivos Críticos a Respaldar:

Archivo	Importancia	Razón
`.env`	🔴 CRÍTICO	Contiene credenciales de RMS y Shopify
`checkpoint.json`	🟡 ALTA	Estado de sincronización (evita re-procesar todo)
`docker-compose.yml`	🟢 MEDIA	Configuración de servicios
`logs/`	🟢 BAJA	Solo para diagnóstico si hay problemas

4.3 Paso 2: Instalación en Equipo Nuevo

Método A: Instalación con Docker (Recomendado)

```
 # Ejecutar en PowerShell como Administrador
# En el equipo NUEVO

# 1. Instalar Docker Desktop
# Descargar de: https://www.docker.com/products/docker-desktop/
# Ejecutar instalador y REINICIAR el equipo

# 2. Verificar Docker instalado
docker --version
# Debe mostrar: Docker version 24.x.x

# 3. Clonar o copiar proyecto
# Opción 3a: Desde Git
git clone https://github.com/tu-usuario/rms-shopify-integration.git
cd rms-shopify-integration

# Opción 3b: Copiar desde USB/Red
# Copiar carpeta completa al equipo nuevo

# 4. Restaurar configuración desde backup
$BackupDir = "C:\Backups\rms-shopify-20250111"  # Ajustar fecha
Copy-Item "$BackupDir\.env" ".\.env"
Copy-Item "$BackupDir\checkpoint.json" ".\checkpoint\checkpoint.json"

Write-Host "✅ Configuración restaurada"

# 5. CRÍTICO: Actualizar IP de RMS si cambió
notepad .env
# Buscar línea: RMS_DB_HOST=190.106.75.222
# Actualizar con la IP ACTUAL del servidor RMS
# Guardar y cerrar

# 6. Verificar configuración
Get-Content .env | Select-String "RMS_DB_HOST"
# Verificar que la IP sea correcta

# 7. Construir imagen Docker
docker build -t rms-shopify-integration:latest .
Write-Host "✅ Imagen Docker construida"

# 8. Iniciar servicios
docker-compose up -d
Write-Host "✅ Servicios iniciados"

# 9. Verificar servicios corriendo
docker-compose ps
# Debe mostrar:
# NAME                    STATUS
# rms-shopify-api         Up
# rms-shopify-redis       Up

# 10. Verificar logs en vivo
docker-compose logs -f api
# Presionar Ctrl+C para salir
```

Método B: Instalación Nativa (Sin Docker)

```
# Ejecutar en PowerShell como Administrador
# En el equipo NUEVO

# 1. Instalar Python 3.13
# Descargar de: https://www.python.org/downloads/
# Durante instalación: ☑ Add Python to PATH

# 2. Verificar Python instalado
python --version
# Debe mostrar: Python 3.13.x

# 3. Instalar Poetry (gestor de dependencias)
(Invoke-WebRequest -Uri https://install.python-poetry.org -UseBasicParsing).Content | python -

# 4. Clonar o copiar proyecto
git clone https://github.com/tu-usuario/rms-shopify-integration.git
cd rms-shopify-integration

# 5. Crear entorno virtual
python -m venv venv
.\venv\Scripts\Activate.ps1

# 6. Instalar dependencias
poetry install
Write-Host "✅ Dependencias instaladas"

# 7. Restaurar configuración
$BackupDir = "C:\Backups\rms-shopify-20250111"
Copy-Item "$BackupDir\.env" ".\.env"
Copy-Item "$BackupDir\checkpoint.json" ".\checkpoint\checkpoint.json"

# 8. Actualizar .env con IP de RMS
notepad .env
# Actualizar RMS_DB_HOST si cambió

# 9. Instalar y arrancar Redis (Windows)
# Descargar de: https://github.com/microsoftarchive/redis/releases
# O usar Memurai: https://www.memurai.com/

# 10. Iniciar aplicación
uvicorn app.main:app --host 0.0.0.0 --port 8000
# O en background:
# Start-Process "uvicorn" -ArgumentList "app.main:app --host 0.0.0.0 --port 8000" -WindowStyle Hidden
```

4.4 Paso 3: Configurar Firewall

```
# Permitir puertos necesarios

# Puerto 8000 (API REST)
New-NetFirewallRule `
  -DisplayName "RMS-Shopify API" `
  -Direction Inbound `
  -Protocol TCP `
  -LocalPort 8000 `
  -Action Allow

# Puerto 6379 (Redis)
New-NetFirewallRule `
  -DisplayName "RMS-Shopify Redis" `
  -Direction Inbound `
  -Protocol TCP `
  -LocalPort 6379 `
  -Action Allow

Write-Host "✅ Firewall configurado"
```

4.5 Paso 4: Configurar como Servicio Windows (Opcional)

```
# Solo si se requiere inicio automático con Windows

# 1. Instalar NSSM (Non-Sucking Service Manager)
choco install nssm
# O descargar de: https://nssm.cc/download

# 2. Crear servicio
$AppPath = "C:\rms-shopify-integration\venv\Scripts\python.exe"
$AppArgs = "C:\rms-shopify-integration\app\main.py"
nssm install RMSShopifyIntegration $AppPath $AppArgs

# 3. Configurar servicio
nssm set RMSShopifyIntegration AppDirectory "C:\rms-shopify-integration"
nssm set RMSShopifyIntegration DisplayName "RMS-Shopify Integration"
nssm set RMSShopifyIntegration Description "Sincronización bidireccional RMS-Shopify"
nssm set RMSShopifyIntegration Start SERVICE_AUTO_START

# 4. Configurar recuperación automática
nssm set RMSShopifyIntegration AppStdout "C:\rms-shopify-integration\logs\service-out.log"
nssm set RMSShopifyIntegration AppStderr "C:\rms-shopify-integration\logs\service-err.log"

# 5. Iniciar servicio
Start-Service RMSShopifyIntegration

# 6. Verificar estado
Get-Service RMSShopifyIntegration
# Debe mostrar: Status: Running

Write-Host "✅ Servicio Windows configurado"
```

4.6 Paso 5: Reconfigurar Webhooks en Shopify

Solo necesario si la IP pública del equipo cambió

```
Pasos en Shopify Admin:

1. Ir a: Settings → Notifications → Webhooks

2. Buscar webhook existente:
   Event: Order creation
   URL: https://[IP-ANTIGUA]:8000/api/v1/webhooks/shopify/orders/create

3. Actualizar URL a:
   https://[IP-NUEVA]:8000/api/v1/webhooks/shopify/orders/create

4. Guardar cambios

5. Probar webhook:
   - Crear orden de prueba en Shopify
   - Verificar recepción en logs
```

Alternativa (si usa dominio):

```
Si se usa un dominio (ej: rms-sync.bestbrands.com):
- No es necesario reconfigurar webhooks
- Solo actualizar DNS A record con nueva IP
- Esperar propagación DNS (5-30 minutos)
```

---

5. Verificación Post-Instalación

5.1 Checklist de Verificación

```
# Script de verificación automática

Write-Host "🔍 Verificando instalación..."

# 1. Verificar servicios corriendo
$api = Get-Process -Name "python" -ErrorAction SilentlyContinue
$redis = Get-Process -Name "redis-server" -ErrorAction SilentlyContinue

if ($api) { Write-Host "✅ API corriendo" } else { Write-Host "❌ API NO corriendo" }
if ($redis) { Write-Host "✅ Redis corriendo" } else { Write-Host "❌ Redis NO corriendo" }

# 2. Verificar puerto 8000 abierto
$port8000 = Test-NetConnection -ComputerName localhost -Port 8000 -InformationLevel Quiet
if ($port8000) { Write-Host "✅ Puerto 8000 abierto" } else { Write-Host "❌ Puerto 8000 cerrado" }

# 3. Verificar acceso web
try {
    $response = Invoke-WebRequest -Uri "http://localhost:8000/docs" -UseBasicParsing
    if ($response.StatusCode -eq 200) {
        Write-Host "✅ Swagger UI accesible"
    }
} catch {
    Write-Host "❌ Swagger UI NO accesible"
}

# 4. Verificar health check
try {
    $health = Invoke-RestMethod -Uri "http://localhost:8000/api/v1/health"
    if ($health.status -eq "healthy") {
        Write-Host "✅ Sistema saludable"
        Write-Host "   - RMS: $($health.services.rms_database)"
        Write-Host "   - Shopify: $($health.services.shopify_api)"
        Write-Host "   - Redis: $($health.services.redis)"
    }
} catch {
    Write-Host "❌ Health check falló"
}

# 5. Verificar motor de sincronización
try {
    $status = Invoke-RestMethod -Uri "http://localhost:8000/api/v1/sync/monitor/status"
    if ($status.motor_activo) {
        Write-Host "✅ Motor de sincronización activo"
        Write-Host "   - Último sync: $($status.ultimo_sync)"
    } else {
        Write-Host "⚠️  Motor de sincronización INACTIVO"
    }
} catch {
    Write-Host "❌ No se puede verificar motor"
}

# 6. Verificar archivos críticos
$files = @(".env", "checkpoint\checkpoint.json")
foreach ($file in $files) {
    if (Test-Path $file) {
        Write-Host "✅ Archivo existe: $file"
    } else {
        Write-Host "❌ Archivo faltante: $file"
    }
}

Write-Host "`n🎉 Verificación completada"
```

5.2 Tests Manuales

Test 1: Health Check

```
# En navegador:
http://localhost:8000/api/v1/health

# Respuesta esperada:
{
  "status": "healthy",
  "timestamp": "2025-01-11T15:30:00Z",
  "services": {
    "rms_database": "connected",
    "shopify_api": "connected",
    "redis": "connected"
  }
}
```

Test 2: Estado del Motor

```
# En navegador o Postman:
GET http://localhost:8000/api/v1/sync/monitor/status

# Respuesta esperada:
{
  "motor_activo": true,
  "ultimo_sync": "2025-01-11T15:25:00Z",
  "proximo_sync": "2025-01-11T15:30:00Z",
  "intervalo_minutos": 5
}
```

Test 3: Sincronización Manual

```
# Trigger sync manual
POST http://localhost:8000/api/v1/sync/monitor/trigger

# Respuesta esperada:
{
  "status": "iniciado",
  "sync_id": "sync_20250111_153000",
  "mensaje": "Sincronización manual iniciada"
}
```

Test 4: Verificar Logs

```
# Ver últimas 50 líneas del log
Get-Content .\logs\app.log -Tail 50

# Buscar errores
Get-Content .\logs\app_errors.log -Tail 20

# Debería verse:
# [INFO] Sistema iniciado correctamente
# [INFO] Motor de sincronización activo
# [INFO] Conexión a RMS exitosa
# [INFO] Conexión a Shopify exitosa
```

5.3 Checklist Final de Migración

```
✓ Docker/Python instalado en nuevo equipo
✓ Código clonado y configurado
✓ Archivo .env restaurado con credenciales
✓ Checkpoint.json restaurado (estado de sincronización)
✓ RMS_DB_HOST actualizado si cambió IP
✓ Servicios iniciados (docker-compose up -d o python app/main.py)
✓ Health check: RMS connected
✓ Health check: Shopify connected
✓ Health check: Redis connected
✓ Motor de sincronización activo (motor_activo: true)
✓ Sincronización manual de prueba exitosa
✓ Logs sin errores críticos
✓ Firewall configurado (puertos 8000, 6379)
✓ Webhooks actualizados en Shopify (si IP cambió)
✓ Servicio Windows configurado (opcional)
✓ Monitoreo configurado (opcional)
```

---

6. Troubleshooting

6.1 Problemas Comunes

Problema: "Cannot connect to SQL Server"

```
Error: [SQL Server] Login failed for user 'sa'

Soluciones:
1. Verificar IP de RMS en .env:
   notepad .env
   RMS_DB_HOST=190.106.75.222  # ¿Es correcta?

2. Verificar credenciales:
   RMS_DB_USER=sa
   RMS_DB_PASSWORD=tu_password  # ¿Es correcta?

3. Verificar conectividad de red:
   Test-NetConnection -ComputerName 190.106.75.222 -Port 1433

4. Verificar firewall de SQL Server
5. Verificar SQL Server acepta conexiones remotas
```

Problema: "Shopify API rate limit exceeded"

```
Error: 429 Too Many Requests

Soluciones:
1. Reducir frecuencia de sync:
   PUT http://localhost:8000/api/v1/sync/monitor/interval
   {"interval_minutes": 10}

2. Reducir batch size en .env:
   SYNC_BATCH_SIZE=25  # En vez de 100

3. Esperar 60 segundos (rate limit se resetea)
```

Problema: "Redis connection refused"

```
Error: Connection refused [Errno 111]

Soluciones (Docker):
docker-compose ps  # Verificar redis corriendo
docker-compose restart redis

Soluciones (Nativo):
# Iniciar Redis manualmente
redis-server

# O instalar como servicio
```

Problema: "Port 8000 already in use"

```
Error: Address already in use

Soluciones:
1. Verificar qué proceso usa el puerto:
   netstat -ano | findstr :8000

2. Matar proceso:
   taskkill /PID [PID_NUMBER] /F

3. O cambiar puerto en .env:
   API_PORT=8080
```

6.2 Comandos de Diagnóstico

```
# Ver logs en tiempo real
docker-compose logs -f api

# Ver solo errores
docker-compose logs api | Select-String "ERROR"

# Verificar uso de recursos
docker stats

# Reiniciar servicios
docker-compose restart

# Reiniciar solo API
docker-compose restart api

# Verificar variables de entorno
docker-compose exec api env | Select-String "RMS"
```

6.3 Contacto de Soporte

```
Si los problemas persisten:

Email: enzo@oneclick.cr
Incluir:
  - Logs: .\logs\app.log y .\logs\app_errors.log
  - Configuración: docker-compose ps
  - Error exacto y pasos para reproducir
```

Tiempo Estimado de Instalación

Actividad	Tiempo
Backup equipo antiguo	15 min
Instalación software base (Docker/Python)	30 min
Configuración y restauración	15 min
Inicio de servicios	10 min
Verificación y testing	20 min
Configuración opcional (Servicio Windows)	10 min
TOTAL	**1-2 horas**

Downtime estimado: 30-60 minutos (mientras se configura nuevo equipo)

Documento generado: Noviembre 2025 **Versión:** 1.0 **Autor:** OneClick Development **Contacto:** enzo@oneclick.cr