Guía de Despliegue — De Pruebas a Producción

Esta guía explica paso a paso cómo llevar Hacienda API desde desarrollo/pruebas/staging hacia producción de forma segura y repetible. Cubre validaciones previas al lanzamiento, gestión de secretos, configuración del entorno, CI/CD, despliegue, monitoreo, rollback y validación post-lanzamiento.

Audiencia: Desarrolladores, DevOps, Ingenieros de Lanzamiento

---

Checklist (nivel alto)

- [ ] Todas las pruebas pasan en local y CI (unitarias, integración, e2e)
- [ ] Umbrales de cobertura alcanzados (proyecto requiere 80% global)
- [ ] Linting y formateo limpios
- [ ] Colecciones Postman / Newman pasan (pruebas smoke)
- [ ] Secretos (certificados, credenciales ATV) almacenados en gestor de secretos
- [ ] SDK real de ATV habilitado y probado en staging
- [ ] Backups y permisos de almacenamiento en lugar para `INVOICES_DIR`
- [ ] Monitoreo, logging y alertas configurados
- [ ] Plan de rollback documentado y probado

---

1) Pre-lanzamiento (local & CI)

1.1 Ejecutar pruebas y cobertura localmente (PowerShell):

```
cd "c:\Users\Christofer Brenes\Documents\PRACTICA\Hacienda_API"
npm ci
npm test
npm run test:coverage
```

- Corregir cualquier prueba fallida antes de continuar.
- Asegurar que la cobertura cumple los umbrales requeridos reportados por `jest`.

1.2 Verificaciones de linting y formateo:

```
npm run lint
npm run format:check
```

1.3 Ejecutar pruebas de integración / smoke (Supertest o Newman):

- Si tienes una colección Postman, ejecutar con Newman:

```
npx newman run postman/EnglishInvoices.postman_collection.json -e postman/env-prod.json
```

- O ejecutar suites de integración con Supertest:

```
npx jest tests/integration --runInBand
```

1.4 Verificar paridad de entornos: staging debe imitar producción (versión de Node, variables de entorno, ruta de almacenamiento, sim vs ATV real). Si `SIMULATE_IF_NO_KEYS=true` está configurado en staging, cambiar a modo REAL solo después de que los certificados estén en lugar.

---

2) Secretos y Certificados

2.1 Lista de secretos de producción (NO almacenar en el repositorio):

- `ATV_KEY_PATH` (ruta a clave privada / .p12)
- `ATV_CERT_PATH` (ruta a certificado / .p12)
- `ATV_CLIENT_ID` (ID de cliente)
- `ATV_USERNAME` , `ATV_PIN`
- `DATABASE_URL` u otras credenciales de servicios externos
- `NODE_ENV=production`

2.2 Elegir una solución de almacenamiento de secretos:

- GitHub Secrets (para Actions), AWS Secrets Manager, Azure Key Vault, HashiCorp Vault, o variables de entorno inyectadas por el orquestador.

2.3 Colocar archivos de certificado de forma segura en el servidor o usar montajes de secretos. Ejemplo (host Linux):

- Almacenar `/etc/secrets/hacienda/certificate.p12` y configurar `ATV_CERT_PATH` en consecuencia.
- Asegurar propiedad y permisos de archivo (solo root o usuario de la app):

```
chown root:appuser /etc/secrets/hacienda/certificate.p12
chmod 640 /etc/secrets/hacienda/certificate.p12
```

2.4 Validar certificado y SDK en staging antes de habilitar modo REAL.

---

3) Preparar la Aplicación para Producción

3.1 Variables de entorno (ejemplo `.env` para producción - pero usar gestor de secretos en despliegues reales):

```
PORT=3000
NODE_ENV=production
SIMULATE_IF_NO_KEYS=false
ATV_KEY_PATH=/etc/secrets/hacienda/private.key
ATV_CERT_PATH=/etc/secrets/hacienda/certificate.p12
ATV_CLIENT_ID=your-client-id
ATV_USERNAME=your-username
ATV_PIN=your-pin
LOG_LEVEL=warn
INVOICES_DIR=/var/lib/hacienda_api/invoices
MAX_FILE_SIZE=10MB
ALLOWED_MIME_TYPES=application/json,application/xml,application/pdf
```

3.2 Sistema de archivos: almacenamiento de facturas

- Asegurar que `INVOICES_DIR` existe y es escribible por el usuario del servicio.

```
mkdir -p /var/lib/hacienda_api/invoices
chown appuser:appgroup /var/lib/hacienda_api/invoices
chmod 750 /var/lib/hacienda_api/invoices
```

3.3 Logging

- Asegurar que `logs/` es rotado (logrotate) o configurar logging centralizado (CloudWatch, ELK, etc.).

3.4 Habilitar modo REAL en `atvAdapter.js`

- Instalar la dependencia del SDK real en `package.json` si no está ya presente: `npm install @facturacr/atv-sdk`.
- En `src/services/atvAdapter.js` descomentar o implementar la lógica de `_initRealMode()` para crear la instancia del SDK usando `ATV_KEY_PATH`, `ATV_CERT_PATH` y credenciales. Hacer esto solo en un entorno seguro.

3.5 Endurecimiento de seguridad

- Ejecutar con usuario con privilegios mínimos
- Mantener secretos fuera de logs
- Usar TLS para todas las comunicaciones externas

---

4) CI/CD — Build, Test, Publish

4.1 Ejemplo de job en GitHub Actions (mínimo) — `/.github/workflows/ci.yml`:

```
name: CI
on: [push, pull_request]
jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v4
        with:
          node-version: '18'
      - run: npm ci
      - run: npm test
      - run: npm run test:coverage
  build-and-push:
    runs-on: ubuntu-latest
    needs: test
    if: github.ref == 'refs/heads/main'
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v4
        with:
          node-version: '18'
```

```
      - run: npm ci --production
      - run: docker build -t ghcr.io/${{ github.repository }}:v${{ github.run_number }} .
      - uses: docker/login-action@v2
        with:
          registry: ghcr.io
          username: ${{ github.actor }}
          password: ${{ secrets.GITHUB_TOKEN }}
      - run: docker push ghcr.io/${{ github.repository }}:v${{ github.run_number }}
```

4.2 Etiquetado y lanzamientos

- Usar etiquetas de versión semántica y crear Lanzamientos en GitHub para despliegues de producción.

4.3 Artefactos de build

- Enviar imágenes Docker construidas a un registro (DockerHub, GHCR, ECR).

---

5) Desplegar a Staging

5.1 Desplegar el mismo artefacto/imagen a un entorno de staging primero.

- Usar la misma configuración que producción pero con certificados de prueba o `SIMULATE_IF_NO_KEYS=true` si deseas comportamiento simulado.

5.2 Ejecutar pruebas smoke (Postman/Newman) contra staging:

```
npx newman run postman/EnglishInvoices.postman_collection.json -e postman/env-staging.json --bail
```

5.3 Ejecutar prueba de carga básica (Artillery o ApacheBench):

```
artillery quick --count 20 -n 100 http://staging.example.com/health
```

5.4 Validar logs, almacenamiento y conectividad del SDK de ATV (si está en REAL).

---

6) Estrategia de Lanzamiento en Producción

6.1 Opciones de modelo de despliegue

- Host único (systemd): bueno para configuraciones pequeñas
- Contenedor Docker: recomendado
- Kubernetes: recomendado para escala y actualizaciones progresivas
- Despliegues Blue/Green o Canary recomendados para riesgo mínimo

6.2 Despliegue progresivo (ejemplo estrategia Kubernetes)

- Usar `Deployment` con `maxUnavailable: 1` y `maxSurge: 1` para rollout seguro.
- Sondeos de liveness y readiness: asegurar que la sonda readiness verifica `/health` y que `atvAdapter` está inicializado cuando está en modo REAL.

6.3 Ejemplo de unidad systemd (simple):

```
[Unit]
Description=Hacienda API
After=network.target

[Service]
User=appuser
Group=appgroup
WorkingDirectory=/opt/hacienda_api
ExecStart=/usr/bin/node src/server.js
Restart=on-failure
Environment=NODE_ENV=production
Environment=ATV_CERT_PATH=/etc/secrets/hacienda/certificate.p12

[Install]
WantedBy=multi-user.target
```

6.4 Pasos para lanzar a producción (mínimos):

1. Etiquetar el lanzamiento en Git (ej: `v1.2.0`) y hacer push
2. CI construye e envía imagen Docker
3. Sacar la imagen en el orquestador de producción o desplegar usando tu pipeline
4. Ejecutar pruebas smoke inmediatamente después del despliegue
5. Monitorear logs y métricas durante 10-30 minutos
6. Si hay problemas, rollback a la etiqueta de imagen anterior

---

7) Validación Post-despliegue & Pruebas Smoke

7.1 Pruebas smoke después del despliegue (PowerShell):

```
# Health check e info básicos
Invoke-RestMethod -Uri "https://api.example.com/health"
Invoke-RestMethod -Uri "https://api.example.com/info"

# Emitir una factura ligera (datos simulados o de staging) y validar
$payload = Get-Content -Raw .\docs\COMPLETE_PAYLOADS.md | ConvertFrom-Json # o construir un
payload pequeño
Invoke-RestMethod -Uri "https://api.example.com/api/english-invoices/emit" -Method POST -
Body ($payload | ConvertTo-Json) -ContentType 'application/json'
```

7.2 Verificar almacenamiento: asegurar que los archivos de factura se guardan y los permisos son correctos.

7.3 Verificar logs: sin errores repetidos en `logs/error.log`.

7.4 Monitorear métricas: uso de CPU, memoria, disco, latencia de request, tasa de error.

---

8) Monitoreo y Alertas

8.1 Métricas importantes a monitorear

- Tasa de requests (RPS)
- Tasa de error (5xx, 4xx)
- Latencia promedio y p95/p99
- Uso de disco de `INVOICES_DIR`
- Fallos de conectividad con ATV (timeouts, errores de auth)
- Logs de aplicación para excepciones no capturadas

8.2 Alertas a configurar

- Tasa de error > 1% durante 5m
- Latencia promedio > 1s durante 5m (ajustar por endpoint)
- Fallos de conectividad con ATV repetidos N veces
- Uso de disco > 80%

8.3 Logging

- Agregar logs a un sistema centralizado (ELK, CloudWatch, Datadog)
- Mantener `error.log` y `combined.log` rotados

---

9) Backup y Retención

- Hacer backup de `./src/data/consecutivo.json` periódicamente (diariamente) y antes de lanzamientos.
- Hacer backup de `INVOICES_DIR` a almacenamiento de objetos (S3, Azure Blob) con reglas de ciclo de vida.
- Retener logs críticos e invoices según requisitos regulatorios.

---

10) Plan de Rollback

10.1 Opciones de rollback rápido:

- Re-desplegar etiqueta anterior de imagen Docker
- Si se ejecuta en un host único, iniciar unidad systemd anterior con binario más antiguo

10.2 Pasos para rollback:

1. Pausar tráfico (si está detrás de load balancer)
2. Desplegar artefacto anterior (imagen o código)
3. Ejecutar pruebas smoke
4. Verificar integridad de almacenamiento de facturas
5. Notificar a stakeholders

10.3 Post-rollback: abrir incidente, recolectar logs, realizar análisis de causa raíz.

---

11) Notas de Seguridad & Cumplimiento

- Mantener credenciales de ATV offline y rotar regularmente.
- Asegurar TLS para todos los endpoints externos.
- Encriptar backups si contienen datos sensibles.
- Evitar loguear PII o contenido de certificados.
- Revisar políticas de retención regulatorias para invoices.

---

12) Comandos Quick-Start de Ejemplo (PowerShell)

```
# Construir y ejecutar localmente en modo producción
npm ci
$env:NODE_ENV='production'
$env:ATV_CERT_PATH='C:\secrets\hacienda\certificate.p12'
$env:ATV_KEY_PATH='C:\secrets\hacienda\private.key'
$env:ATV_CLIENT_ID='CLIENT_ID'
$env:SIMULATE_IF_NO_KEYS='false'
node src/server.js

# Ejecutar pruebas smoke (ejemplo de una línea)
Invoke-RestMethod -Uri 'http://localhost:3000/health'
```

---

13) Solución de Problemas

- `ATV adapter falla al inicializar` : verificar ruta del certificado, permisos y versión del SDK. Habilitar logs debug temporalmente.
- `Facturas no se guardan` : verificar ruta de `INVOICES_DIR` y permisos de escritura.
- `Tasa de error alta después del despliegue` : hacer rollback inmediatamente e investigar logs.

---

14) Plantilla de Aprobación de Lanzamiento

- Pruebas: ✅ unitarias ✅ integración ✅ e2e
- Certificados: ✅ disponibles en gestor de secretos
- Backups: ✅ verificados
- Monitoreo: ✅ dashboards + alertas
- Revisión de seguridad: ✅ secretos/permisos verificados
- Aprobación: [Nombre], [Fecha], [Notas]

Apéndice: Enlaces Útiles

- `docs/TESTING.md` — instrucciones de prueba
- `docs/COMPLETE_PAYLOADS.md` — ejemplos de payloads
- `docs/MANUAL_TECNICO.md` — manual técnico

Fin de guía