Análisis de Scripts - Repositorio Escalafín

Proyecto: Aurum Control Center (ERP Holding + 11 Satélites)
Fuente: https://github.com/qhosting/escalafin
Fecha: 2025-12-12

Scripts Encontrados

Total: 23 archivos en `/scripts`

1. Scripts de Verificación Pre-Build/Deploy

`pre-build-check.sh`

Propósito: Verificación antes de build Docker. Detecta problemas que causarían fallo en el build.

Funcionalidad:
- Verifica existencia de Dockerfile y archivos críticos
- Valida package.json y yarn.lock/package-lock.json
- Verifica coherencia entre lockfile y package.json
- Verifica estructura de directorios requerida (components/, lib/, prisma/, public/)
- Valida scripts de startup (start-improved.sh, emergency-start.sh)
- Verifica .dockerignore
- Test opcional de build del stage 'deps'

Aplicable a Aurum: ✅ SÍ - CRÍTICO
- Aurum usará Docker/contenedores para deployment
- Previene errores comunes de build antes de gastar tiempo/recursos
- Adaptable a Next.js/monorepo structure

`pre-deploy-check.sh`

Propósito: Verificación exhaustiva antes de deploy en EasyPanel/producción.

Funcionalidad:
- Verifica todos los archivos críticos (Dockerfile, package.json, next.config.js, schema.prisma)
- Valida configuración de output: 'standalone' en next.config.js (CRÍTICO para Docker)
- Verifica git status y commits
- Valida dependencias críticas (next, react, @prisma/client, next-auth)
- Verifica configuración Docker (stages, comandos críticos, EXPOSE)
- Valida template de variables de entorno
- Verifica estructura de directorios

Aplicable a Aurum: ✅ SÍ - CRÍTICO
- Validación completa antes de deployment
- Evita deployments rotos que afecten producción
- Adaptable a arquitectura de Aurum

`pre-deploy-verification.sh`

Propósito: Verificación rápida antes de push y rebuild.

Funcionalidad:
- Verifica archivos críticos y scripts de producción
- Valida directorios esenciales
- Verifica contenido de Dockerfile (WORKDIR, COPY scripts)
- Valida .dockerignore (scripts no deben estar ignorados)
- Verifica sincronización de dependencias
- Verifica estado de git (commits sin push)
- Valida permisos de scripts

Aplicable a Aurum: ✅ SÍ - RECOMENDADO
- Validación rápida antes de cada push
- Previene errores simples que rompen builds
- Útil para equipo de desarrollo

`post-deploy-check.sh`

Propósito: Verificación después del deploy para confirmar que la aplicación funciona.

Funcionalidad:
- Verifica conectividad básica (DNS, HTTP response, response time)
- Valida certificado SSL
- Verifica contenido de la página (Next.js detectado, sin errores)
- Prueba endpoints críticos (/api/health, /api/auth/signin, /login)
- Verifica recursos estáticos (_next/static)
- Valida headers de seguridad (X-Frame-Options, HSTS)

Aplicable a Aurum: ✅ SÍ - CRÍTICO
- Validación automática post-deployment
- Detecta problemas inmediatamente después del deploy
- Permite rollback rápido si hay problemas
- Adaptable a múltiples satélites (validar cada uno)

---

2. Scripts de Diagnóstico

`cache-diagnostics.sh`

Propósito: Diagnóstico de problemas causados por cache de construcción.

Funcionalidad:
- Verifica timestamps de archivos críticos
- Detecta si package-lock.json es más antiguo que package.json
- Verifica sincronización con GitHub (commits sin push)
- Detecta archivos críticos modificados sin commitear
- Verifica coherencia de Dockerfile con archivos existentes
- Busca síntomas de uso de cache antiguo
- Genera hashes de verificación de archivos críticos

Aplicable a Aurum: ✅ SÍ - MUY ÚTIL
- Problemas de cache son comunes en CI/CD
- Detecta discrepancias entre local y remoto
- Ahorra tiempo de debugging

`diagnose-db.sh`

Propósito: Diagnóstico de conectividad y estado de base de datos PostgreSQL.

Funcionalidad:
- Verifica conectividad de red a la BD
- Valida credenciales de acceso
- Lista tablas existentes
- Verifica migraciones de Prisma
- Muestra estadísticas de la base de datos

Aplicable a Aurum: ✅ SÍ - CRÍTICO
- Aurum tendrá múltiples bases de datos (holding + satélites)
- Diagnóstico rápido de problemas de conexión
- Validación de migraciones en diferentes ambientes
- **ADAPTAR:** Agregar soporte para múltiples DBs

---

3. Scripts de Git/Push

`pre-push-check.sh`

Propósito: Git pre-push hook para verificar antes de push.

Funcionalidad:
- Detecta si yarn.lock es symlink (Docker no puede copiarlo)
- Ofrece conversión automática de symlink a archivo real
- Valida rutas absolutas problemáticas
- Verifica archivos críticos para Docker build
- Valida que Dockerfile copie .yarn/ correctamente (para Yarn 4)
- Verifica que schema.prisma no tenga rutas absolutas
- Valida shebangs de scripts (bash vs sh)
- Verifica configuración de HOME en Dockerfile

Aplicable a Aurum: ✅ SÍ - RECOMENDADO
- Previene pushes con errores críticos
- Detecta problemas específicos de Docker
- Adaptable a gestores de paquetes (npm/yarn/pnpm)
- **IMPORTANTE:** Aurum debe decidir gestor de paquetes desde Fase 1

`safe-push.sh`

Propósito: Push seguro con verificaciones automáticas.

Funcionalidad:
- Verifica estado de git (cambios sin commitear)

- Ejecuta pre-push-check.sh
- Pull antes de push (evita conflictos)
- Push con token si está disponible
- Muestra último commit

Aplicable a Aurum: ✅ SÍ - ÚTIL
- Automatiza workflow seguro de git
- Previene conflictos y errores
- Útil para equipo de desarrollo

`setup-git-hooks.sh`

Propósito: Instalar git hooks preventivos automáticamente.

Funcionalidad:
- Instala pre-push hook que ejecuta pre-push-check.sh
- Hace ejecutables los scripts necesarios

Aplicable a Aurum: ✅ SÍ - RECOMENDADO
- Automatiza instalación de hooks en equipo
- Previene errores desde el inicio

---

4. Scripts de Rollback/Recovery

`emergency-rollback.sh`

Propósito: Rollback de emergencia a estado anterior estable.

Funcionalidad:
- Lista backups disponibles
- Crea backup del estado actual antes de rollback
- Restaura desde backup seleccionado (preservando .git)
- Verifica archivos críticos después del rollback

Aplicable a Aurum: ⚠️ PARCIAL - ADAPTAR
- Útil para recovery rápido
- **IMPORTANTE:** En producción con múltiples satélites, el rollback debe ser más sofisticado
- **ADAPTAR:** Usar control de versiones (git tags) + deployment system rollback
- **RECOMENDACIÓN:** Implementar blue-green deployment o canary releases

---

5. Scripts de Validación

`validate-absolute-paths.sh`

Propósito: Detecta rutas absolutas problemáticas que causan errores en Docker.

Funcionalidad:
- Busca rutas absolutas del host (/opt/, /home/ubuntu, /root/)
- Detecta symlinks en código fuente
- Verifica configuración de paths en tsconfig.json, next.config.js, package.json
- Verifica imports con rutas absolutas

- Valida Dockerfile (rutas del host vs contenedor)
- Verifica .dockerignore

Aplicable a Aurum: ✅ SÍ - CRÍTICO
- Rutas absolutas son causa común de errores en Docker
- Previene problemas de portabilidad
- **IMPORTANTE:** En monorepo, validar paths entre paquetes

`revision-fix.sh`

Propósito: Revisión de fixes aplicados para prevenir regresiones.

Funcionalidad:
- Verifica que rutas absolutas no regresen
- Detecta referencias incorrectas a gestores de paquetes
- Verifica scripts necesarios existen
- Valida .dockerignore
- Verifica dependencias críticas
- Valida NODE_PATH en scripts
- Verifica estructura del Dockerfile
- Valida configuración de Prisma

Aplicable a Aurum: ✅ SÍ - ÚTIL
- Previene regresiones de fixes aplicados
- Útil en CI/CD pipeline
- **ADAPTAR:** Agregar checks específicos de Aurum

---

6. Scripts de Utilidades

`fix-yarn-lock-symlink.sh`

Propósito: Convertir yarn.lock de symlink a archivo real.

Funcionalidad:
- Detecta si yarn.lock es symlink
- Convierte a archivo real (cp -L)
- Verifica la conversión

Aplicable a Aurum: ⚠️ CONDICIONAL
- Solo si Aurum usa Yarn
- **DECISIÓN FASE 1:** Definir gestor de paquetes (npm/yarn/pnpm)

`update-version.sh`

Propósito: Actualizar versión del proyecto (semver).

Funcionalidad:
- Actualiza versión en package.json (major/minor/patch)
- Genera build number automático
- Actualiza version.json con metadata completa

- Actualiza CHANGELOG.md
- Crea commit y tag automáticamente

Aplicable a Aurum: ✅ SÍ - RECOMENDADO
- Versionado automático es crítico para ERP
- Trazabilidad de releases
- **ADAPTAR:** Versiones independientes para holding y satélites

---

`generate-env.js`

Propósito: Generar archivo .env con valores seguros y aleatorios.

Funcionalidad:
- Genera secretos criptográficamente seguros
- Configura DATABASE_URL automáticamente
- Crea backup si el archivo ya existe
- Genera archivo de resumen con credenciales
- Opciones para personalizar DB y app URL

Aplicable a Aurum: ✅ SÍ - CRÍTICO
- Setup automatizado de environments
- Generación segura de secrets
- **ADAPTAR:** Generar .env para múltiples entornos (holding + satélites)
- **IMPORTANTE:** Integrar con secrets management (Vault, AWS Secrets Manager)

`pg_backup.sh`

Propósito: Backup automático de PostgreSQL.

Funcionalidad:
- Crea dumps SQL comprimidos (gzip)
- Nomenclatura con timestamp
- Limpia backups antiguos automáticamente (retention policy)
- Validación de credenciales

Aplicable a Aurum: ✅ SÍ - CRÍTICO
- Backups automáticos esenciales para ERP
- **ADAPTAR:** Backup de múltiples DBs (holding + satélites)
- **IMPORTANTE:** Implementar backup remoto (S3, backup service)
- **RECOMENDACIÓN:** Point-in-time recovery (WAL archiving)

`test-hash.js`

Propósito: Test de hashing de passwords con bcrypt.

Funcionalidad:
- Prueba generación de hash
- Verifica comparación de passwords

- Genera hashes para usuarios de prueba
- Validación de seguridad

Aplicable a Aurum: ⚠️ **PARCIAL**
- Útil para testing en desarrollo
- **NO USAR** en producción para generar passwords reales
- **RECOMENDACIÓN:** Usar en seeds de desarrollo/testing únicamente

---

7. Scripts de Push/GitHub

`push-ambos-repos.sh`, `push-github.sh`, `subir-github.sh`

Propósito: Scripts para push a repositorios GitHub (específicos del proyecto Escalafín).

Aplicable a Aurum: ❌ **NO**
- Específicos de la estructura de repos de Escalafín
- Aurum tendrá su propia estrategia de repos

`verificacion-github.sh`

Propósito: Verificación de GitHub (específico de Escalafín).

Aplicable a Aurum: ❌ **NO**
- Específico del proyecto Escalafín

`verificar-links-dashboards.sh`

Propósito: Verificar links en dashboards (específico de Escalafín).

Aplicable a Aurum: ❌ **NO**
- Específico de la estructura de Escalafín
- **ADAPTAR:** Si Aurum tiene verificación de links, crear script específico

---

Resumen de Aplicabilidad

Scripts CRÍTICOS para Aurum (Prioridad 1)

1. ✅ **pre-build-check.sh** - Prevención de errores de build
2. ✅ **pre-deploy-check.sh** - Validación exhaustiva pre-deployment
3. ✅ **post-deploy-check.sh** - Validación post-deployment
4. ✅ **diagnose-db.sh** - Diagnóstico de bases de datos
5. ✅ **validate-absolute-paths.sh** - Prevención de errores de paths
6. ✅ **generate-env.js** - Setup automatizado de environments
7. ✅ **pg_backup.sh** - Backups automáticos

Scripts RECOMENDADOS para Aurum (Prioridad 2)

1. ✅ **pre-deploy-verification.sh** - Verificación rápida pre-push
2. ✅ **cache-diagnostics.sh** - Diagnóstico de cache

3. ✅ **pre-push-check.sh** - Git hook preventivo
4. ✅ **setup-git-hooks.sh** - Instalación automática de hooks
5. ✅ **update-version.sh** - Versionado automático
6. ✅ **revision-fix.sh** - Prevención de regresiones

Scripts ÚTILES (Prioridad 3)

1. ✅ **safe-push.sh** - Push seguro con validaciones

Scripts CONDICIONALES

1. ⚠️ **fix-yarn-lock-symlink.sh** - Solo si se usa Yarn

Scripts PARCIALES (Requieren Adaptación)

1. ⚠️ **emergency-rollback.sh** - Adaptar para multi-tenant
2. ⚠️ **test-hash.js** - Solo para dev/testing

Scripts NO APLICABLES

1. ❌ **push-ambos-repos.sh**, **push-github.sh**, **subir-github.sh** - Específicos de Escalafín
2. ❌ **verificacion-github.sh**, **verificar-links-dashboards.sh** - Específicos de Escalafín

---

Recomendaciones de Integración para Aurum Control Center

Fase 1 - Setup Inicial

Integrar inmediatamente:

1. `pre-build-check.sh` - Adaptar para estructura de Aurum
2. `pre-deploy-check.sh` - Validación exhaustiva
3. `diagnose-db.sh` - Adaptar para múltiples DBs
4. `validate-absolute-paths.sh` - Prevención de errores
5. `generate-env.js` - Adaptar para múltiples environments
6. `setup-git-hooks.sh` - Automatizar instalación

Acciones:

- Crear directorio `/scripts` en raíz del monorepo
- Copiar y adaptar scripts críticos
- Documentar en README.md
- Integrar en workflow de CI/CD

Fase 2 - Consolidación

Agregar:

7. `post-deploy-check.sh` - Validación post-deployment
8. `cache-diagnostics.sh` - Diagnóstico avanzado
9. `update-version.sh` - Versionado automático
10. `pg_backup.sh` - Backups automáticos (adaptar para múltiples DBs)

Acciones:

- Integrar post-deploy-check en CI/CD pipeline
- Configurar backups automáticos (cron jobs)
- Establecer estrategia de versionado

Fase 3 - Optimización

Agregar:
11. `safe-push.sh` - Automatización para equipo
12. `revision-fix.sh` - Prevención de regresiones
13. `emergency-rollback.sh` - Adaptar para arquitectura distribuida

Acciones:
- Implementar blue-green deployment
- Establecer políticas de rollback
- Documentar runbooks de emergencia

Adaptaciones Específicas para Aurum

1. Multi-Database Support

Scripts a adaptar: `diagnose-db.sh`, `pg_backup.sh`

Cambios necesarios:
- Soportar múltiples DATABASE_URLs (holding + satélites)
- Backup paralelo o secuencial
- Validación de conectividad a todas las DBs
- Reporte consolidado

Ejemplo:

```
# En lugar de:
DATABASE_URL="..."

# Usar:
DATABASE_URLS=(
  "HOLDING_DB_URL"
  "SATELITE_1_DB_URL"
  "SATELITE_2_DB_URL"
  # ... hasta 11 satélites
)
```

2. Monorepo Structure

Scripts a adaptar: `pre-build-check.sh`, `pre-deploy-check.sh`, `validate-absolute-paths.sh`

Cambios necesarios:
- Validar estructura de paquetes del monorepo
- Verificar interdependencias entre paquetes
- Validar workspaces (si se usa Turborepo/Nx)
- Verificar path aliases (@aurum/shared, etc.)

3. Multi-Environment

Scripts a adaptar: `generate-env.js`, `pre-deploy-check.sh`

Cambios necesarios:
- Generar .env para múltiples entornos (dev, staging, prod)

- Validar variables específicas por satélite
- Soportar .env.local, .env.production, etc.

4. Versionado Independiente

Scripts a adaptar: `update-version.sh`

Cambios necesarios:
- Versionado independiente para holding y satélites
- Matriz de compatibilidad entre versiones
- Changelog consolidado

5. CI/CD Integration

Todos los scripts de verificación

Cambios necesarios:
- Agregar flags para output en formato JSON
- Exit codes específicos para CI
- Integración con sistemas de notificación (Slack, email)

Estructura Propuesta para Aurum

```
aurum-control-center/
├── scripts/
│   ├── build/
│   │   ├── pre-build-check.sh          # ✅ De Escalafín
│   │   ├── validate-absolute-paths.sh   # ✅ De Escalafín
│   │   └── cache-diagnostics.sh         # ✅ De Escalafín
│   ├── deploy/
│   │   ├── pre-deploy-check.sh          # ✅ De Escalafín
│   │   ├── pre-deploy-verification.sh   # ✅ De Escalafín
│   │   └── post-deploy-check.sh         # ✅ De Escalafín
│   ├── database/
│   │   ├── diagnose-db.sh               # ✅ De Escalafín (adaptar)
│   │   ├── pg_backup.sh                 # ✅ De Escalafín (adaptar)
│   │   └── multi-db-backup.sh           # 🆕 Nuevo para Aurum
│   ├── git/
│   │   ├── pre-push-check.sh            # ✅ De Escalafín
│   │   ├── safe-push.sh                 # ✅ De Escalafín
│   │   └── setup-git-hooks.sh           # ✅ De Escalafín
│   ├── utils/
│   │   ├── generate-env.js              # ✅ De Escalafín (adaptar)
│   │   ├── update-version.sh            # ✅ De Escalafín (adaptar)
│   │   ├── revision-fix.sh              # ✅ De Escalafín
│   │   └── test-hash.js                 # ✅ De Escalafín (solo dev)
│   └── emergency/
│       └── emergency-rollback.sh        # ⚠️ De Escalafín (adaptar)
├── README.md                            # Documentación de scripts
└── .github/
    └── workflows/
        ├── pre-build.yml                # CI workflow con scripts
        └── deploy.yml                   # CD workflow con scripts
```

Próximos Pasos

Inmediatos (Fase 1 - Setup)

1. ✅ Crear directorio `/scripts` en Aurum
2. ✅ Copiar scripts críticos de Escalafín
3. ✅ Adaptar `diagnose-db.sh` para múltiples DBs
4. ✅ Adaptar `generate-env.js` para múltiples environments
5. ✅ Documentar uso de scripts en README
6. ✅ Integrar `pre-build-check.sh` en CI

Corto Plazo (Fase 1 - Build)

1. ✅ Integrar `pre-deploy-check.sh` en CI/CD
2. ✅ Configurar git hooks con `setup-git-hooks.sh`
3. ✅ Establecer estrategia de versionado
4. ✅ Implementar validación de paths en CI

Mediano Plazo (Fase 2)

1. Implementar `post-deploy-check.sh` en CD
2. Configurar backups automáticos (adaptar `pg_backup.sh`)
3. Implementar diagnóstico avanzado de cache
4. Establecer runbooks de emergencia

Largo Plazo (Fase 3)

1. Implementar rollback automatizado
2. Blue-green deployment
3. Monitoring y alerting integrados con scripts
4. Self-healing capabilities

Conclusión

Los scripts del repositorio Escalafín proporcionan una **base sólida y probada** para:
- ✅ Prevención de errores comunes de build/deployment
- ✅ Diagnóstico rápido de problemas
- ✅ Automatización de tareas repetitivas
- ✅ Validación exhaustiva en múltiples capas

Para Aurum Control Center:
- **17 de 23 scripts** son aplicables (74%)
- **7 scripts críticos** deben integrarse en Fase 1
- **6 scripts recomendados** agregan valor significativo
- **Adaptaciones necesarias** para arquitectura multi-tenant y monorepo

Impacto esperado:
- ⬇️ 60-80% reducción en errores de deployment
- ⬆️ 50% mejora en tiempo de diagnóstico
- ✅ Prevención de >90% de errores comunes
- 🚀 Setup inicial más rápido y confiable

Inversión requerida:
- Fase 1: ~8-12 horas (adaptar scripts críticos)
- Fase 2: ~16-24 horas (integración completa)
- Fase 3: ~24-32 horas (optimización y runbooks)

ROI: Alto - Los scripts previenen errores costosos y aceleran development/deployment