🎯 FLUJO ESTÁNDAR DE PLANIFICACIÓN DEVOPS

Script Universal para Cualquier Proyecto

Fecha: 2025-11-08
Basado en: Infrastructure Showcase + Plan Final Consensuado
Propósito: Documento maestro que define el flujo completo y repetible para transformar cualquier idea en blueprint ejecutable

---

📊 RESUMEN EJECUTIVO

Este documento define el **flujo estándar** que se aplicará a CUALQUIER proyecto que entre al sistema de planificación DevOps. No importa si es:

- Un juego en Unreal Engine 5
- Un backend con Node.js
- Una aplicación frontend con React
- Un sistema de robots
- Una arquitectura en Kubernetes

El flujo es siempre el mismo: 6 etapas, 11 agentes, proceso repetible y verificable.

🔄 EL CICLO DE 6 ETAPAS (G_0 → R → M → G_1)

ENTRADA: Requisitos del proyecto
↓
ETAPA 1: DESCUBRIMIENTO (1-2 semanas)
↓
ETAPA 2: ARQUITECTURA (2-3 semanas)
↓
ETAPA 3: REVISIÓN CRUZADA (1-2 semanas)
↓
ETAPA 4: DOCUMENTACIÓN (1 semana)
↓
ETAPA 5: APROBACIÓN (3-5 días)
↓
ETAPA 6: HANDOFF (3-5 días)
↓
SALIDA: Blueprint ejecutable

Duración total estimada: 6-9 semanas
Gates de calidad: 7 checkpoints (Gate 0 → Gate 6)
Score mínimo requerido: ≥7/10 en rubrica de 6 dimensiones

🎭 LOS 11 ACTORES DEL SISTEMA

#	División	Complejidad	Rol Principal	Especialización
1	**plan-reviewer**	Alta	Gate keeper	Validación y calidad final
2	**code-architecture-reviewer**	Alta	Arquitecto	Decisiones técnicas estructurales
3	**documentation-architect**	Alta	Ensamblador	Coherencia documental
4	**web-research-specialist**	Alta	Validador	Verificación de evidencias
5	**code-refactor-master**	Alta	Diseñador macro	Refactors futuros
6	**refactor-planner**	Alta	Estratega	Deuda técnica
7	**frontend-error-fixer**	Media	Calidad UI	Validación frontend
8	**auth-route-debugger**	Media	Seguridad	Autenticación/Autorización
9	**auth-route-tester**	Media	QA API	Contratos y criterios
10	**auto-error-resolver**	Baja	Higiene	TypeScript/estático
11	**Explore**	Media	Explorador	Gaps y riesgos ocultos

📋 MATRIZ CONSOLIDADA: AGENTES × ETAPAS × RACI

Leyenda RACI:

- **R** (Responsible): Ejecuta la tarea directamente
- **A** (Accountable): Responsable final, aprueba/rechaza
- **C** (Consulted): Se le consulta, provee input
- **I** (Informed): Se le informa, recibe output

Matriz Completa (11 divisiones × 6 etapas = 66 asignaciones)

División	Etapa 1 Descubrimiento	Etapa 2 Arquitectura	Etapa 3 Revisión	Etapa 4 Documentación	Etapa 5 Aprobación	Etapa
plan-reviewer	**A**	**A**	**R/A**	**A**	**R/A**	**A**
code-architecture-reviewer	C	**R**	C	C	C	C
documentation-architect	I	I	I	**R**	C	**R**
web-research-specialist	C	C	C	C	I	I
code-refactor-master	C	C	I	I	I	I
refactor-planner	I	C	I	I	I	I
frontend-error-fixer	I	C	C	I	I	C
auth-route-debugger	I	C	C	I	I	C
auth-route-tester	I	I	C	I	I	**R**
auto-error-resolver	I	C	C	I	C	C
Explore	**R**	C	C	I	I	I

Interpretación:

- **plan-reviewer**: Es Accountable en TODAS las etapas (Gate keeper supremo)
- **Explore**: Lidera Descubrimiento (Etapa 1)
- **code-architecture-reviewer**: Lidera Arquitectura (Etapa 2)
- **documentation-architect**: Lidera Documentación (Etapa 4) y Handoff (Etapa 6)
- **auth-route-tester**: Lidera contratos API en Handoff (Etapa 6)

🔍 DETALLE DE CADA ETAPA

ETAPA 1: DESCUBRIMIENTO (1-2 semanas)

Objetivo: Definir QUÉ vamos a planear

Agente Principal: Explore (R)
Gate Keeper: plan-reviewer (A)

Actores y Roles:

Actor	RACI	Responsabilidad Específica
Explore	**R**	Mapear proyecto completo, identificar riesgos
plan-reviewer	**A**	Aprobar/rechazar con Gate 1 (Score ≥6/10)
web-research-specialist	C	Validar que requisitos sean verificables
code-architecture-reviewer	C	Pre-validar viabilidad técnica

Artefactos:

ENTRADA:

- Descripción del proyecto (lo que sea: juego, backend, etc.)
- Restricciones técnicas (stack, deadline, presupuesto)
- Requisitos no funcionales (performance, escalabilidad)

SALIDA:

- ✅ **Project Charter** (1-2 páginas)
- ✅ **Scope Statement** (límites explícitos: "in scope" / "out of scope")
- ✅ **Risk Register Inicial** (top-10 riesgos con mitigaciones)
- ✅ **Glossario de Términos** (definiciones clave)

Criterios de Gate 1 (Definition of Done):

- ✅ Objetivos SMART definidos
- ✅ Scope boundaries claras
- ✅ Top-10 riesgos identificados con mitigaciones preliminares
- ✅ plan-reviewer da aprobación (Score ≥6/10)

Artefactos Clave Producidos/Consumidos por División:

Explore produce:

- Project Charter
- Risk Register
- Scope Statement

plan-reviewer consume:

- Todo lo anterior (para validar)

plan-reviewer produce:

- Reporte de revisión Gate 1
- Score preliminar (≥6/10 para pasar)

ETAPA 2: ARQUITECTURA (2-3 semanas)

Objetivo: Definir CÓMO lo haremos

Agente Principal: code-architecture-reviewer (R)
Gate Keeper: plan-reviewer (A)

Actores y Roles:

Actor	RACI	Responsabilidad Específica
code-architecture-reviewer	**R**	Definir arquitectura objetivo, crear ADRs
plan-reviewer	**A**	Aprobar/rechazar con Gate 2
code-refactor-master	C	Evaluar impacto de refactors futuros
refactor-planner	C	Identificar deuda técnica anticipada
web-research-specialist	C	Validar patrones propuestos con evidencia
frontend-error-fixer	C	Validar consideraciones UI/UX
auth-route-debugger	C	Validar flujos de autenticación
auto-error-resolver	C	Validar higiene TypeScript/estático

Artefactos:

ENTRADA:

- Project Charter (de Etapa 1)
- Decisiones arquitectónicas propuestas
- Patrones candidatos (Layered, Hexagonal, DDD, etc.)

SALIDA:

- ✅ **ADRs** (≥3 decisiones formalizadas: Context → Decision → Consequences)
- ✅ **Architecture Diagram** (componentes, bounded contexts, dependencias)
- ✅ **Trade-offs Analysis** (pros/cons de cada decisión)
- ✅ **Dependency Graph** (qué depende de qué)
- ✅ **Pattern Selection** (patrones elegidos con justificación)

Criterios de Gate 2 (Definition of Done):

- ✅ ≥3 ADRs creados (estructura completa)
- ✅ Architecture Diagram completo y coherente
- ✅ Trade-offs documentados para decisiones críticas
- ✅ plan-reviewer da aprobación (Score ≥7/10)

Artefactos Clave Producidos/Consumidos por División:

code-architecture-reviewer produce:

- ADRs (≥3)
- Architecture Diagram
- Trade-offs Analysis

plan-reviewer consume:

- Todo lo anterior (para validar)

code-refactor-master consume:

- Architecture Diagram (para evaluar refactors)

plan-reviewer produce:

- Reporte de revisión Gate 2
- Score arquitectónico (≥7/10 para pasar)

ETAPA 3: REVISIÓN CRUZADA (1-2 semanas)

Objetivo: Validar consenso de las 11 divisiones

Coordinador: plan-reviewer (R/A)
Participantes: LAS 11 DIVISIONES (C)

Actores y Roles:

Actor	RACI	Responsabilidad Específica
plan-reviewer	**R/A**	Coordinar revisión, resolver conflictos, Gate 3
TODAS las 11 divisiones	C	Comentar el plan, proponer mejoras

CRÍTICO: Cada división DEBE comentar. No se acepta "está bien" sin fundamento.

Artefactos:

ENTRADA:

- ADRs de Etapa 2
- Architecture Diagram
- Plan parcial ensamblado

SALIDA:

- ✅ **Review Comment Log** (tabla: División → Comentario → Resolución)
- ✅ **Conflict Resolution Records** (si hay conflictos, ADRs secundarios)
- ✅ **Sign-off Report** (100% divisiones aprobaron o tienen objeciones documentadas)

Criterios de Gate 3 (Definition of Done):

- ✅ 100% de divisiones comentaron (revisión cruzada completa)
- ✅ Conflictos resueltos (usando Protocolo ANEXOS/protocolo-conflictos.md)
- ✅ Todos los comentarios respondidos (acción tomada o rechazada con justificación)
- ✅ plan-reviewer da aprobación (Score ≥7/10, "consenso alcanzado")

Protocolo de Resolución de Conflictos:

1. plan-reviewer detecta conflicto
2. Reunir divisiones involucradas
3. Documentar posiciones (ADR con opción A vs opción B)
4. Aplicar tie-breaker:
   - Decisiones técnicas → code-architecture-reviewer
   - Decisiones de proceso → documentation-architect
5. Registrar decisión final en ADR
6. Comunicar a todas las divisiones

Artefactos Clave Producidos/Consumidos:

Todas las divisiones consumen:

- Plan parcial (ADRs + Architecture Diagram)

Todas las divisiones producen:

- Comentarios específicos (no genéricos)

plan-reviewer produce:

- Review Comment Log consolidado
- Conflict Resolution Records (si aplica)
- Sign-off Report

ETAPA 4: DOCUMENTACIÓN (1 semana)

Objetivo: Ensamblar blueprint master coherente

Agente Principal: documentation-architect (R)
Gate Keeper: plan-reviewer (A)

Actores y Roles:

Actor	RACI	Responsabilidad Específica
documentation-architect	**R**	Ensamblar PLAN_PRINCIPAL, validar progressive disclosure
plan-reviewer	**A**	Aprobar/rechazar con Gate 4
web-research-specialist	C	Validar todas las citas (rutas absolutas)
code-architecture-reviewer	C	Revisar coherencia técnica del ensamblado

Artefactos:

ENTRADA:

- ADRs y comentarios de Etapa 3
- Hallazgos de cada división (de investigación previa)
- Diagramas, trade-offs, dependency graphs

SALIDA:

- ✅ **PLAN_PRINCIPAL.md** (≤500 líneas, índice maestro)
- ✅ **ANEXOS/** (todos ≤500 líneas cada uno):
  - ANEXOS/adr-compilado.md
  - ANEXOS/architecture-diagram.md
  - ANEXOS/trade-offs-analysis.md
  - ANEXOS/[DIVISIÓN].md (uno por cada de las 11)

- ✅ **Index/Referencias** (validación de citas, rutas absolutas correctas)

Criterios de Gate 4 (Definition of Done):

- ✅ PLAN_PRINCIPAL.md < 500 líneas (progressive disclosure)
- ✅ Todos los anexos < 500 líneas cada uno
- ✅ 0 links rotos (validar con herramienta)
- ✅ 100% referencias con rutas absolutas verificables
- ✅ documentation-architect da aprobación (Score ≥7/10)

Artefactos Clave Producidos/Consumidos:

documentation-architect consume:

- ADRs compilados
- Comentarios de revisión cruzada
- Hallazgos de cada división

documentation-architect produce:

- PLAN_PRINCIPAL.md (≤500L)
- ANEXOS/ (completo)
- Index de referencias

web-research-specialist consume:

- PLAN_PRINCIPAL + ANEXOS (para validar citas)

web-research-specialist produce:

- Reporte de validación de referencias

ETAPA 5: APROBACIÓN (3-5 días)

Objetivo: ¿Está el plan listo para ejecutar?

Agente Principal: plan-reviewer (R/A)

Actores y Roles:

Actor	RACI	Responsabilidad Específica
plan-reviewer	R/A	Ejecutar checklist, calcular score, firmar
documentation-architect	C	Resolver dudas de estructura
code-architecture-reviewer	C	Resolver dudas técnicas
auto-error-resolver	C	Validar higiene final

Artefactos:

ENTRADA:

- PLAN_PRINCIPAL.md + ANEXOS de Etapa 4
- Checklist de Verificación (12 items)

SALIDA:

- ✅ **Scoring Report** (Score ≥7/10 usando rubrica de 6 dimensiones)
- ✅ **Checklist Completado** (12 items, 100% pasados)
- ✅ **Sign-off** (firma oficial de plan-reviewer)
- ✅ **Correction Log** (si score <7/10, acciones de corrección)

Checklist de 12 Items:

1. ✅ Canon Organizacional - 11 divisiones presentes, RACI 66/66
2. ✅ Skills Correctamente Activados (verification-before-completion, etc.)
3. ✅ Sin Placeholders (grep "TODO|TBD|pending" = 0)
4. ✅ RACI Completo (66 asignaciones: 11 divisiones × 6 etapas)
5. ✅ ADRs Presentes (≥3 ADRs con Context→Decision→Consequences)
6. ✅ Rutas Absolutas Verificables (100% referencias válidas)
7. ✅ Progressive Disclosure (documentos <500L)
8. ✅ Artefactos Explícitos (cada división define consume/produce)
9. ✅ Riesgos con Mitigaciones (20+ riesgos, 100% con mitigaciones)
10. ✅ Validación Cruzada (11 planes revisados, 0 conflictos)
11. ✅ Score Final ≥7/10
12. ✅ Aprobación Explícita de plan-reviewer

Rubrica de Scoring (6 dimensiones):

Dimensión	Peso	Descripción
1. Completitud	20%	Todas las secciones obligatorias presentes
2. Coherencia	20%	Sin contradicciones internas
3. Trazabilidad	15%	Decisiones justificadas con referencias
4. Viabilidad	20%	Técnicamente factible
5. Risk-Awareness	15%	Riesgos identificados y mitigados
6. Claridad	10%	Lenguaje preciso, sin ambigüedades

Total: 100% → Score de 0 a 10

Criterios de Gate 5 (Definition of Done):

- ✅ Checklist: 12 items, 100% pasados
- ✅ Score ≥7/10 (rubrica explícita)
- ✅ 0 placeholders ("TODO", "TBD", "pending")
- ✅ RACI completo (66 asignaciones)
- ✅ plan-reviewer da aprobación FINAL (firma)

Artefactos Clave Producidos/Consumidos:

plan-reviewer consume:

- PLAN_PRINCIPAL + ANEXOS completo
- Checklist

plan-reviewer produce:

- Scoring Report (con desglose por dimensión)
- Checklist completado
- Sign-off oficial o Correction Log

ETAPA 6: HANDOFF A EJECUCIÓN (3-5 días)

Objetivo: Paquete mínimo ejecutable

Agentes Principales: documentation-architect (R) + auth-route-tester (R)
Gate Keeper: plan-reviewer (A)

Actores y Roles:

Actor	RACI	Responsabilidad Específica
documentation-architect	**R**	Crear EXECUTION_PACKAGE.md
auth-route-tester	**R**	Documentar API Contracts completos
plan-reviewer	**A**	Aprobar/rechazar con Gate 6 FINAL
auto-error-resolver	C	Validar rollback strategy
frontend-error-fixer	C	Validar criterios de aceptación UI
auth-route-debugger	C	Validar flujos de autenticación

Artefactos:

ENTRADA:

- PLAN_PRINCIPAL.md + ANEXOS (aprobados en Etapa 5)
- Contratos API preliminares
- Criterios de aceptación preliminares

SALIDA:

- ✅ **EXECUTION_PACKAGE.md** (qué ejecutar, en qué orden, criterios de éxito)
- ✅ **API Contracts** (request/response schemas + ejemplos curl)
- ✅ **Acceptance Criteria** (por feature, formato GIVEN/WHEN/THEN)
- ✅ **Rollback Strategy** (cómo volver atrás si falla)
- ✅ **Success Metrics** (KPIs de ejecución, thresholds)

Criterios de Gate 6 (Definition of Done):

- ✅ EXECUTION_PACKAGE.md claro (equipo NO necesita preguntar)
- ✅ API Contracts 100% documentados (schemas + ejemplos)
- ✅ Acceptance Criteria para 100% features (GIVEN/WHEN/THEN)
- ✅ Rollback strategy documentado
- ✅ plan-reviewer da aprobación FINAL (paquete válido)

Artefactos Clave Producidos/Consumidos:

documentation-architect produce:

- EXECUTION_PACKAGE.md

auth-route-tester produce:

- API Contracts (completos con schemas)
- Acceptance Criteria (por feature)

auto-error-resolver produce:

- Rollback Strategy

plan-reviewer consume:

- Todo lo anterior (para validar)

plan-reviewer produce:

- Aprobación FINAL Gate 6
- Blueprint COMPLETO y EJECUTABLE

📦 ARTEFACTOS CONSOLIDADOS POR DIVISIÓN

1. plan-reviewer

Consume:

- Planes de todas las divisiones
- Checklists de verificación (auto-error-resolver, auth-route-tester)
- ADRs (code-architecture-reviewer)
- Matrices RACI (documentation-architect)
- Reportes de riesgos (Explore)

Produce:

- Reportes de revisión (por cada gate)
- Scores de calidad (escala 1-10)
- Aprobaciones/Rechazos formales
- Checklists de calidad
- Rubrica de scoring
- Sign-off oficial final

---

2. code-architecture-reviewer

Consume:

- Project Charter (Explore)
- Requisitos técnicos
- Patrones arquitectónicos candidatos

Produce:

- ADRs (≥3)
- Architecture Diagram
- Trade-offs Analysis
- Dependency Graph
- Pattern Selection justificado

---

3. documentation-architect

Consume:

- ADRs de todas las divisiones
- Comentarios de revisión cruzada
- Hallazgos de investigación
- Diagramas técnicos

Produce:

- PLAN_PRINCIPAL.md (≤500L)
- ANEXOS/ completo (≤500L cada uno)
- Index de referencias
- EXECUTION_PACKAGE.md

---

4. web-research-specialist

Consume:

- Documentos con citas
- Referencias bibliográficas
- Claims que necesitan validación

Produce:

- Reporte de validación de referencias
- Contradiction Registry (fuentes conflictivas)
- Matriz de evidencias

---

5. code-refactor-master

Consume:

- Architecture Diagram
- Decisiones arquitectónicas (ADRs)

Produce:

- Análisis de impacto de refactors
- Estrategia de refactors macro
- Rollback plans

---

6. refactor-planner

Consume:

- Architecture Diagram
- Plan de implementación

Produce:

- Análisis de deuda técnica anticipada
- Estrategia de refactor a nivel plan

---

7. frontend-error-fixer

Consume:

- Requisitos UI/UX
- Decisiones arquitectónicas frontend

Produce:

- Análisis de calidad UI
- Validación de estados de error
- Criterios de aceptación UI

---

8. auth-route-debugger

Consume:

- Requisitos de autenticación
- Flujos de autorización

Produce:

- Análisis de flujos de error auth
- Validación de seguridad

---

9. auth-route-tester

Consume:

- Especificaciones de API
- Requisitos de contratos

Produce:

- API Contracts (schemas completos)
- Acceptance Criteria (GIVEN/WHEN/THEN)
- Ejemplos curl

---

10. auto-error-resolver

Consume:

- Código TypeScript/JavaScript
- Configuraciones de linting

Produce:

- Checklist de higiene TypeScript
- Rollback Strategy
- Validación estática

---

11. Explore

Consume:

- Requisitos del proyecto
- Corpus de documentación
- Contexto del dominio

Produce:

- Project Charter
- Scope Statement
- Risk Register (top-10 riesgos)
- Glossario de términos
- Gaps identificados

🎯 APLICACIÓN DEL FLUJO A CUALQUIER PROYECTO

Ejemplo 1: Juego Unreal Engine 5

Entrada: "Crear juego FPS multijugador en UE5"

Etapa 1 (Explore):

- Project Charter: "FPS multijugador, 10v10, modos de juego..."
- Scope: In scope (gameplay core) / Out of scope (campaña single-player)
- Riesgos: Networking, performance, assets 3D

Etapa 2 (code-architecture-reviewer):

- ADRs: Client-Server architecture, ECS pattern, Asset streaming
- Diagram: Game Server ↔ Clients, Replication system

Etapa 3-6: Seguir flujo estándar...

Ejemplo 2: Backend Node.js + PostgreSQL

Entrada: "API REST para e-commerce"

Etapa 1 (Explore):

- Project Charter: "API REST, autenticación JWT, productos + carritos..."
- Scope: In scope (core CRUD) / Out of scope (pagos reales)
- Riesgos: Escalabilidad, seguridad, migración DB

Etapa 2 (code-architecture-reviewer):

- ADRs: Layered Architecture, Repository pattern, JWT auth
- Diagram: Routes → Controllers → Services → Repositories → DB

Etapa 3-6: Seguir flujo estándar...

Ejemplo 3: Frontend React + Material UI

Entrada: "Dashboard administrativo"

Etapa 1 (Explore):

- Project Charter: "Dashboard React, MUI v7, visualizaciones..."
- Scope: In scope (CRUD usuarios) / Out of scope (analytics avanzado)
- Riesgos: Performance con grandes datasets, accesibilidad

Etapa 2 (code-architecture-reviewer):

- ADRs: Feature-based folder structure, Context API, React Query
- Diagram: Components hierarchy, State management

Etapa 3-6: Seguir flujo estándar...

✅ CHECKLIST RÁPIDO PARA JOSÉ

Cuando vayas a aplicar este flujo a un nuevo proyecto:

Antes de empezar:

- [ ] ¿Tengo los requisitos básicos del proyecto?
- [ ] ¿Sé cuál es el tech stack (o al menos el dominio)?
- [ ] ¿Tengo las 11 divisiones disponibles?

Durante Etapa 1 (Descubrimiento):

- [ ] Explore generó Project Charter
- [ ] Scope Statement tiene "in/out of scope" explícito
- [ ] Risk Register tiene ≥10 riesgos con mitigaciones
- [ ] plan-reviewer aprobó Gate 1 (Score ≥6/10)

Durante Etapa 2 (Arquitectura):

- [ ] code-architecture-reviewer generó ≥3 ADRs
- [ ] Architecture Diagram está completo
- [ ] Trade-offs documentados

- [ ] plan-reviewer aprobó Gate 2 (Score ≥7/10)

Durante Etapa 3 (Revisión Cruzada):

- [ ] Las 11 divisiones comentaron (100%)
- [ ] Conflictos resueltos (si hubo)
- [ ] Sign-off de todas las divisiones
- [ ] plan-reviewer aprobó Gate 3

Durante Etapa 4 (Documentación):

- [ ] PLAN_PRINCIPAL.md <500 líneas
- [ ] ANEXOS/ completo (<500L cada uno)
- [ ] 0 links rotos
- [ ] plan-reviewer aprobó Gate 4

Durante Etapa 5 (Aprobación):

- [ ] Checklist 12/12 items ✅
- [ ] Score ≥7/10 (rubrica)
- [ ] 0 placeholders
- [ ] plan-reviewer firmó aprobación FINAL

Durante Etapa 6 (Handoff):

- [ ] EXECUTION_PACKAGE.md claro
- [ ] API Contracts completos
- [ ] Acceptance Criteria (GIVEN/WHEN/THEN)
- [ ] Rollback Strategy documentado
- [ ] plan-reviewer aprobó Gate 6 FINAL

Resultado:

- [ ] Blueprint ejecutable listo
- [ ] Equipo de ejecución puede trabajar sin preguntar

🚀 PRÓXIMOS PASOS

Para José:

1. Usa este documento como **referencia permanente**
2. Cada vez que entres un proyecto nuevo, **sigue este flujo exacto**
3. No improvises las etapas: **respeta los gates**
4. Si algo falla, **itera dentro de la etapa** antes de avanzar

Para expansión futura:

- Crear templates específicos por dominio (juegos, backend, frontend, etc.)
- Automatizar la generación de checklists
- Crear herramientas de validación (links rotos, placeholders, etc.)
- Métricas de tiempo por etapa (para optimización)

📚 REFERENCIAS

- **Plan Final Consensuado**: `/mnt/project/PLAN_FINAL_CONSENSUADO.md`
- **Ciclo de Planificación**: `/mnt/project/03_ciclo_planificacion.md`
- **Mapa de Trabajo**: `/mnt/project/01_mapa_trabajo.md`
- **ADRs**: `/mnt/project/04_adrs.md`
- **Infrastructure Showcase**: `/mnt/project/LA_BIBLIA_INFRASTRUCTURE_SHOWCASE.md`

FIN DEL DOCUMENTO

Versión: 1.0
Fecha: 2025-11-08
Autor: Claude (Sonnet 4.5) bajo guía estratégica de José
Modo: HRR (Hierarchical Recursive Reasoning) - L-module ejecutor