Despues de eso mejora esto del analysis worker: 🎯 Sugerencias Menores
1. Circuit Breaker para DriverIA
python# Considera agregar:
@circuit_breaker(failure_threshold=3, timeout=30)
async def _call_driver_ia_safely(self, method, *args):
    return await method(*args)
2. Retry Strategy
python# Para llamadas críticas:
@retry(stop=stop_after_attempt(3), wait=wait_exponential())
async def _execute_with_retry(self, operation):
3. Resource Monitoring
python# Monitoreo de memoria/CPU:
async def _check_resource_usage(self):
    if psutil.virtual_memory().percent > 80:
        await self._trigger_cleanup()   Estas sugerencias son para el Pipeline Científico (backend/src/core/pipeline.py) que revisé anteriormente. Específicamente:
📋 Contexto de las Sugerencias
1. Gestión de Configuración → Para ScientificPipeline

Hardcodeaste valores como max_hits=50, protein_ids[:10], max_tokens=1000
Sugerí centralizarlos en una clase de configuración

2. Retry Logic → Para servicios externos en el pipeline

Tu pipeline llama BLAST, UniProt, LLM
Si fallan, no hay retry automático
Sugerí agregar reintentos exponenciales

3. Caching Estratégico → Para optimizar performance

BLAST/UniProt son lentos y caros
Muchas secuencias pueden dar resultados similares
Sugerí cachear por hash de secuencia

🎯 ¿Por Qué Esas Sugerencias?
En tu pipeline.py tenías:
python# Valores hardcodeados
blast_result = await self.blast_cb.call(
    self.blast_service.search_homology,
    sequence.sequence,
    database="local_db",
    max_hits=50 # ← Hardcodeado
)

protein_ids[:10] # ← Hardcodeado

llm_result = await self.llm_cb.call(
    self.llm_service.analyze_sequence_data,

```
    prompt,
    max_tokens=1000,  # ← Hardcodeado
    temperature=0.3
)
```
📈 Sugerencias de Mejora

1. Gestión de Configuración

```
python# Considera agregar:
@dataclass
class PipelineConfig:
    max_blast_hits: int = 50
    uniprot_batch_size: int = 10
    llm_max_tokens: int = 1000
```

2. Retry Logic Mejorado

```
python# Para servicios externos críticos:
@retry(stop=stop_after_attempt(3), wait=wait_exponential(multiplier=1, min=4, max=10))
async def _call_external_service(self, ...):
```

3. Caching Estratégico

```
python# Para resultados de BLAST/UniProt:
@cached(ttl=3600)  # 1 hora
async def get_blast_results(self, sequence_hash: str):    Para el driver debes mejorar esto:
```

Para DriverIA (nuevo):

Rate limiting para OpenAI API
Token usage tracking para control de costos
Prompt templates centralizados
Result validation antes de almacenar Tambien quiero que implementes estas mejoras de models/analysis.py y del container: ¡Perfecto, Luis! He analizado a fondo los modelos de datos y el contenedor de dependencias que acabas de subir. Déjame darte mi evaluación técnica de estos componentes críticos:

🎯 Análisis de models/analysis.py

🚀 Mejoras Sugeridas (Nivel Avanzado)

1. Validación de Secuencias Biológicas

```
pythonfrom pydantic import validator

class SequenceData(BaseModel):
    sequence: str = Field(..., min_length=10)

    @validator('sequence')
    def validate_biological_sequence(cls, v, values):
        sequence_type = values.get('sequence_type', 'protein')
        if sequence_type == 'protein':
            valid_chars = set('ACDEFGHIKLMNPQRSTVWY')
            if not set(v.upper()).issubset(valid_chars):
                raise ValueError('Invalid protein sequence')
        return v.upper()
```

2. Pipeline Configuration Schema

```
pythonclass PipelineConfig(BaseModel):
    """Configuración tipada para el pipeline."""
```

```
    blast_database: str = Field("nr", description="Base de datos BLAST")
    evalue_threshold: float = Field(1e-10, ge=0, description="E-value threshold")
    max_target_seqs: int = Field(500, ge=1, le=1000)
    uniprot_fields: List[str] = Field(default_factory=lambda: ["function", "pathway"])
    llm_analysis_depth: str = Field("detailed", regex="^(basic|detailed|comprehensive)$")
```

3. Result Caching Strategy

```
class CacheableResult(BaseModel):
    """Base para resultados cacheables."""
    cache_key: str = Field(..., description="Clave de cache")
    cache_ttl: int = Field(3600, description="TTL en segundos")
    cached_at: Optional[datetime] = None

    def is_cache_valid(self) -> bool:
        if not self.cached_at:
            return False
        return (datetime.utcnow() - self.cached_at).seconds < self.cache_ttl
```

Ahora para main tienes que implementar estas mejoras: 🚀 Mejoras Sugeridas (Nivel Avanzado)

1. Request ID Tracing

```
import uuid

@app.middleware("http")
async def add_request_id(request: Request, call_next):
    request_id = str(uuid.uuid4())
    request.state.request_id = request_id

    response = await call_next(request)
    response.headers["X-Request-ID"] = request_id
    return response
```

2. Rate Limiting Middleware

```
from slowapi import Limiter, _rate_limit_exceeded_handler
from slowapi.util import get_remote_address
from slowapi.errors import RateLimitExceeded

limiter = Limiter(key_func=get_remote_address)
app.state.limiter = limiter
app.add_exception_handler(RateLimitExceeded, _rate_limit_exceeded_handler)

@app.post("/api/analysis/start")
@limiter.limit("10/minute")  # 10 análisis por minuto por IP
async def start_analysis(request: Request, ...):
    pass
```

3. Structured Response Format

```
from src.models.analysis import APIResponse

class APIResponse(BaseModel):
    success: bool
    data: Optional[Any] = None
    error: Optional[str] = None
```

```
    timestamp: datetime = Field(default_factory=datetime.utcnow)
    request_id: Optional[str] = None

@app.get("/")
async def root(request: Request):
    return APIResponse(
        success=True,
        data={
            "message": "🧬 Astroflora Antares Core",
            "version": settings.PROJECT_VERSION
        },
        request_id=getattr(request.state, 'request_id', None)    Para el context manager, sqs dispatcher y para settings las siguientes mejoras: 🚀 Mejoras Sugeridas (Nivel Avanzado)
1. Configuration Validation
pythonfrom pydantic import validator

class Settings(BaseSettings):
    OPENAI_API_KEY: str = "sk-placeholder-openai-key"

    @validator('OPENAI_API_KEY')
    def validate_openai_key(cls, v):
        if v != "sk-placeholder-openai-key" and not v.startswith('sk-'):
            raise ValueError('Invalid OpenAI API key format')
        return v
2. Context Manager Query Optimization
python# Agregar índices MongoDB para performance
async def _ensure_indexes(self):
    await self.collection.create_index([("user_id", 1), ("created_at", -1)])
    await self.collection.create_index([("status", 1), ("created_at", -1)])
    await self.collection.create_index([("workspace_id", 1), ("protocol_type", 1)])
3. SQS Dead Letter Queue
python# En settings.py
SQS_DLQ_URL: str = "http://localhost:4566/000000000000/astroflora-analysis-dlq"

# En dispatcher
async def send_to_dlq(self, payload: JobPayload, error: str):
    """Envía trabajos fallidos a Dead Letter Queue."""
    dlq_payload = {**payload.model_dump(), "error": error, "failed_at": datetime.utcnow()}
    # Enviar a DLQ                    Para el analyzis de routers y para interfaces mejora esto: 🚀 Mejoras Sugeridas (Nivel Enterprise)
1. WebSocket para Real-Time Updates
pythonfrom fastapi import WebSocket

@router.websocket("/ws/{context_id}")
async def websocket_endpoint(websocket: WebSocket, context_id: str):
    await websocket.accept()
    try:
        while True:
```

```
            # Envía updates en tiempo real del progreso
            context = await orchestrator.get_analysis_status(context_id)
            await websocket.send_json({
                "context_id": context_id,
                "progress": context.progress,
                "current_step": context.current_step,
                "status": context.status
            })
            await asyncio.sleep(2)  # Update cada 2 segundos
    except Exception as e:
        await websocket.close()
```

2. Advanced Query Parameters

```
pythonfrom pydantic import BaseModel
from datetime import datetime
from typing import Optional

class AnalysisQuery(BaseModel):
    status: Optional[AnalysisStatus] = None
    protocol_type: Optional[PromptProtocolType] = None
    created_after: Optional[datetime] = None
    created_before: Optional[datetime] = None
    workspace_id: Optional[str] = None

@router.get("/search", response_model=List[AnalysisContext])
async def search_analyses(query: AnalysisQuery = Depends()):
    # Búsqueda avanzada con filtros
    pass
```

3. Analysis Templates

```
python@router.get("/templates", response_model=List[AnalysisTemplate])
async def get_analysis_templates():
    """Obtiene plantillas predefinidas para análisis comunes."""
    return [
        AnalysisTemplate(
            name="Protein Function Discovery",
            description="Análisis completo de función de proteína",
            protocol_type=PromptProtocolType.PROTEIN_FUNCTION_ANALYSIS,
            default_parameters={"blast_database": "nr", "evalue": 1e-10}
        )
    ]
```

🚀 Mejoras Adicionales que Sugiero

1. Health Check Comprehensivo

```
python
# En container.py - AGREGAR ESTO
```

```python
async def comprehensive_health_check(self) -> dict:
    """Health check que prueba TODAS las dependencias."""
    health = {
        "timestamp": datetime.utcnow(),
        "overall_status": "healthy",
        "services": {}
    }

    # Test Redis
    try:
        await self.redis_client.ping()
        health["services"]["redis"] = {"status": "healthy", "latency_ms": 0}
    except Exception as e:
        health["services"]["redis"] = {"status": "unhealthy", "error": str(e)}
        health["overall_status"] = "degraded"

    # Test MongoDB
    try:
        start = time.time()
        await self.mongo_client.admin.command("ping")
        latency = (time.time() - start) * 1000
        health["services"]["mongodb"] = {"status": "healthy", "latency_ms": latency}
    except Exception as e:
        health["services"]["mongodb"] = {"status": "unhealthy", "error": str(e)}
        health["overall_status"] = "degraded"

    # Test DriverIA
    try:
        driver_health = await self.driver_ia.health_check()
        health["services"]["driver_ia"] = {"status": "healthy" if driver_health else "unhealthy"}
    except Exception as e:
        health["services"]["driver_ia"] = {"status": "unhealthy", "error": str(e)}
        health["overall_status"] = "degraded"


    return health
```

2. Cost Tracking para LLM

python

```python
# En models/analysis.py - AGREGAR ESTO
class LLMUsage(BaseModel):
    """Tracking de uso y costos de LLM."""
    context_id: str
    model_used: str = Field(..., description="gpt-4, gpt-3.5-turbo, etc")
    prompt_tokens: int = Field(0, description="Tokens en el prompt")
    completion_tokens: int = Field(0, description="Tokens en la respuesta")
    total_tokens: int = Field(0, description="Total de tokens")
```

```
    estimated_cost_usd: float = Field(0.0, description="Costo estimado en USD")
    timestamp: datetime = Field(default_factory=datetime.utcnow)

class CostTracker:
    """Calcula costos de LLM."""
    PRICING = {
        "gpt-4": {"prompt": 0.03/1000, "completion": 0.06/1000},
        "gpt-3.5-turbo": {"prompt": 0.0015/1000, "completion": 0.002/1000}
    }

    @classmethod
    def calculate_cost(cls, model: str, prompt_tokens: int, completion_tokens: int) -> float:
        if model not in cls.PRICING:
            return 0.0

        pricing = cls.PRICING[model]
        return (prompt_tokens * pricing["prompt"]) + (completion_tokens * pricing["completion"])
```

3. Analysis Templates System

python

```
# En models/analysis.py - AGREGAR ESTO
class AnalysisTemplate(BaseModel):
    """Plantilla predefinida para análisis."""
    template_id: str = Field(default_factory=lambda: str(uuid.uuid4()))
    name: str = Field(..., description="Nombre de la plantilla")
    description: str = Field(..., description="Descripción detallada")
    protocol_type: PromptProtocolType
    default_parameters: Dict[str, Any] = Field(default_factory=dict)
    estimated_duration_minutes: int = Field(30, description="Duración estimada")
    cost_tier: str = Field("medium", regex="^(low|medium|high)$")
    tags: List[str] = Field(default_factory=list)

# Plantillas predefinidas
PROTEIN_FUNCTION_TEMPLATE = AnalysisTemplate(
    name="Protein Function Discovery",
    description="Análisis completo de función de proteína usando BLAST + UniProt + LLM",
    protocol_type=PromptProtocolType.PROTEIN_FUNCTION_ANALYSIS,
    default_parameters={
        "blast_database": "nr",
        "evalue_threshold": 1e-10,
        "max_blast_hits": 50,
        "uniprot_fields": ["function", "pathway", "domain"],
        "llm_analysis_depth": "detailed"
    },
    estimated_duration_minutes=45,
    cost_tier="medium",
```

```
    tags=["protein", "function", "annotation"]
)
```