IACADEMY

AGENTIC ENGINEER & SDD

MÓDULO 07

Producción y Operaciones

Avanzado

iacedemy.com — 2026

MÓDULO 07

Producción y Operaciones

Nivel: Avanzado

Autor: Alicia Sanz

Publicación: Julio 2026

Plataforma: iacedemy.com

> “El 10% del trabajo es construir el agente. El 90% es operarlo sin que explote.”

7.1 De Staging a Produccion: Checklist

Pre-Deploy Checklist

```
## Funcionalidad
- [ ] Eval dataset completo (golden + edge + adversarial + regression)
- [ ] CI gate: accuracy >= threshold
- [ ] Regression tests: 100% pass rate
- [ ] Load test: agente soporta N queries/minuto esperadas

## Seguridad
- [ ] Prompt injection tests: >= 95% resistencia
- [ ] Data leakage: 0% en tests adversarial
- [ ] Guardrails: input validation, output sanitization
- [ ] HITL configurado para acciones HIGH/CRITICAL
- [ ] Rate limiting por usuario/sesion
- [ ] Secrets en vault, NO en env vars de texto plano

## Observabilidad
- [ ] OpenTelemetry trazas en tool calls y LLM calls
- [ ] Metricas exportadas (latencia, tokens, cost, error rate)
- [ ] Dashboard configurado
- [ ] Alertas configuradas (error rate, latencia P99, presupuesto)
- [ ] Log rotation configurado (90 dias operacional)

## Infraestructura
- [ ] LLM endpoint probado bajo carga
- [ ] Circuit breakers en todas las tools externas
- [ ] Fallback definido para cuando el LLM no responde
- [ ] Health check endpoint
- [ ] Graceful shutdown (terminar query actual antes de apagar)

## Rollout
- [ ] Feature flag creado (desactivado por defecto)
- [ ] Plan de rollback documentado
- [ ] Rollout gradual: 5% → 25% → 50% → 100%
- [ ] Runbook para incidentes
```

7.2 Estrategias de Deploy

Opcion 1: Serverless (Cloudflare Workers, AWS Lambda)

```
Pros:
  - Scale-to-zero (no pagas cuando no hay trafico)
  - Auto-scaling (picos automaticos)
  - Sin servidores que mantener

Contras:
  - Cold start (1-3s primera invocacion)
  - Timeout limitado (30s Workers, 15min Lambda)
  - Sin GPU (no puedes self-host LLM)
  - Vendor lock-in

Ideal para:
  - Agentes con LLM via API (OpenAI, Anthropic)
  - Trafico variable o impredecible
  - Presupuesto limitado (pago por uso)
```

```python
# Ejemplo: Agente en Cloudflare Worker (Python via Pyodide)
# workers/agent-support.py

async def on_fetch(request, env):
    body = await request.json()
    user_message = body["message"]

    agent = SupportAgent(
        llm_api_key=env.ANTHROPIC_API_KEY,
        kb_url=env.KB_API_URL,
    )

    response = await agent.run(user_message)

    return Response(json.dumps({
        "response": response.text,
        "tool_calls": response.tool_calls_count,
    }))
```

Opcion 2: Containers (Docker + Hetzner/AWS)

```
Pros:
  - Control total
  - Sin cold start
  - GPU disponible (self-host LLM)
  - Sin timeout artificiales

Contras:
  - Mantener servidores
  - Scaling manual (o configurar auto-scaling)
  - Coste fijo (pagas aunque no haya trafico)

Ideal para:
  - Agentes con LLM self-hosted (soberania)
  - Trafico predecible
  - Requisitos de latencia estrictos
```

```yaml
# docker-compose.yml
services:
  agent-support:
    build: ./agent
    environment:
      - VLLM_ENDPOINT=http://vllm:8000
      - QDRANT_URL=http://qdrant:6333
      - OTEL_ENDPOINT=http://otel-collector:4317
    deploy:
      replicas: 3
      resources:
        limits:
          memory: 2G
          cpus: '2'
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:8080/health"]
      interval: 30s
      timeout: 10s
      retries: 3

  vllm:
    image: vllm/vllm-openai:latest
    command: >
      --model Qwen/Qwen3.5-27B
      --max-model-len 16384
      --gpu-memory-utilization 0.9
    deploy:
      resources:
        reservations:
          devices:
            - driver: nvidia
              count: 1
              capabilities: [gpu]

  qdrant:
    image: qdrant/qdrant:latest
    volumes:
      - qdrant_data:/qdrant/storage
```

Opcion 3: Hibrido

```
Agent (Cloudflare Worker) → LLM (self-hosted Hetzner)
                          → DB (Supabase EU)
                          → Vector (Qdrant self-hosted)

- El agente corre serverless (scale-to-zero, global edge)
- El LLM corre self-hosted (soberania, sin coste por token)
- Lo mejor de ambos mundos
```

7.3 Gestion de Costes

Anatomia del Coste de un Agente

```
COSTE POR QUERY = LLM tokens + Tool calls + Infra

LLM Tokens:
  - System prompt (fijo, se paga en cada query)
  - Conversation history (crece con el dialogo)
  - Tool call overhead (~100-200 tokens por call)
  - Response generation

Tool Calls:
  - APIs externas (si son de pago)
  - DB queries (negligible si self-hosted)
  - Embedding generation (para RAG)

Infra:
  - Compute (servidor/container/serverless)
  - GPU (si self-hosted LLM)
  - Storage (vector DB, logs, checkpoints)
```

Model Routing por Coste

```python
class ModelRouter:
    """
    Seleccionar modelo basado en complejidad de la tarea.
    Reduccion tipica: 60-70% en coste con <5% perdida de calidad.
    """

    MODELS = {
        "simple": {
            "model": "claude-haiku-4-5",
            "cost_per_1k_input": 0.0008,
            "cost_per_1k_output": 0.004,
        },
        "medium": {
            "model": "claude-sonnet-4-6",
            "cost_per_1k_input": 0.003,
            "cost_per_1k_output": 0.015,
        },
        "complex": {
            "model": "claude-opus-4-6",
            "cost_per_1k_input": 0.015,
            "cost_per_1k_output": 0.075,
        },
        "self_hosted": {
            "model": "qwen3.5-27b",
            "cost_per_1k_input": 0.0,  # Self-hosted
            "cost_per_1k_output": 0.0,
        },
    }

    async def select_model(self, task: str, sensitivity: str = "normal")
-> str:
        """
        Seleccionar modelo optimo.

        - Clasificacion, routing: haiku (simple)
        - Generacion de respuesta: sonnet (medium)
        - Razonamiento complejo: opus (complex)
        - Datos sensibles: self-hosted (siempre)
        """
        if sensitivity == "sensitive":
            return "self_hosted"

        # Clasificar complejidad con modelo barato
        complexity = await self._classify_complexity(task)
        return complexity

    async def _classify_complexity(self, task: str) -> str:
        response = await llm.complete(
            model="claude-haiku-4-5",
```

```
            messages=[{
                "role": "user",
                "content": f"Clasifica esta tarea como simple/medium/
complex. Solo la palabra.\nTarea: {task}"
            }],
            max_tokens=10,
        )
        return response.text.strip().lower()
```

Prompt Caching

```
# Prompt caching: pagar el system prompt UNA VEZ, reutilizar en queries
siguientes.
# Reduccion: 90% en tokens de input para system prompts grandes.

# Anthropic: cache_control en el system message
messages = [
    {
        "role": "system",
        "content": [
            {
                "type": "text",
                "text": LARGE_SYSTEM_PROMPT,  # 2000+ tokens
                "cache_control": {"type": "ephemeral"}
            }
        ]
    },
    {"role": "user", "content": user_query}
]

# Primera query: paga el system prompt completo
# Queries 2-N: paga solo el cache read (90% más barato)
```

Dashboard de Costes

```
AGENT-SUPPORT-001 — Cost Dashboard (junio 2026)
═══════════════════════════════════════════════

Volumen:        12,450 queries
Coste total:    €37.35

Desglose por componente:
  LLM tokens:     €28.40 (76%)
  Tool calls:     €5.20  (14%)  [APIs externas]
  Infra:          €3.75  (10%)  [Hetzner container]

Desglose por modelo:
  Haiku (classifier):  €2.10  (7,800 calls)
  Sonnet (responder):  €21.30 (4,200 calls)
  Self-hosted (search): €0.00  (12,450 calls)
  Embedding (RAG):     €5.00  (12,450 calls)

Metricas de eficiencia:
  Coste/query:       €0.003
  Tokens/query:      1,847 avg
  Cache hit rate:    78%
  Unnecessary tools: 4.2%

Comparacion:
  Sin model routing: ~€112/mes (3x mas)
  Sin prompt cache:  ~€58/mes  (1.5x mas)
```

7.4 Observabilidad

Que Instrumentar

```
NIVEL 1: METRICAS (numeros agregados)
  - requests_total (counter)
  - request_duration_seconds (histogram)
  - tokens_used_total (counter)
  - tool_calls_total (counter, by tool_name)
  - errors_total (counter, by error_type)
  - cost_eur_total (counter)

NIVEL 2: TRAZAS (flujo de una request)
  - Span: agent.run (parent)
    - Span: llm.call (model, tokens_in, tokens_out)
    - Span: tool.call (tool_name, duration, success)
    - Span: llm.call (second iteration)
    - Span: guardrail.check (type, passed)

NIVEL 3: LOGS (detalle para debug)
  - Input/output de cada tool call
  - LLM response completa (para debugging)
  - Guardrail violations
  - Error stacktraces
```

OpenTelemetry Setup

```
from opentelemetry import trace, metrics
from opentelemetry.sdk.trace import TracerProvider
from opentelemetry.sdk.metrics import MeterProvider
from opentelemetry.exporter.otlp.proto.grpc.trace_exporter import
OTLPSpanExporter
from opentelemetry.exporter.otlp.proto.grpc.metric_exporter import
OTLPMetricExporter

# Setup tracer
tracer_provider = TracerProvider()
tracer_provider.add_span_processor(
    BatchSpanProcessor(OTLPSpanExporter(endpoint="http://otel-
collector:4317"))
)
trace.set_tracer_provider(tracer_provider)
tracer = trace.get_tracer("agent-support-001")

# Setup metrics
meter_provider = MeterProvider(
    metric_readers=[PeriodicExportingMetricReader(
        OTLPMetricExporter(endpoint="http://otel-collector:4317")
    )]
)
metrics.set_meter_provider(meter_provider)
meter = metrics.get_meter("agent-support-001")

# Definir metricas
request_counter = meter.create_counter("agent.requests.total")
token_counter = meter.create_counter("agent.tokens.total")
duration_histogram = meter.create_histogram("agent.request.duration")
cost_counter = meter.create_counter("agent.cost.eur")
```

Alertas

```yaml
# alerting-rules.yaml (Grafana/Prometheus)

groups:
  - name: agent-support-001
    rules:
      - alert: HighErrorRate
        expr: rate(agent_errors_total[5m]) /
rate(agent_requests_total[5m]) > 0.05
        for: 5m
        labels:
          severity: warning
        annotations:
          summary: "Error rate > 5% en AGENT-SUPPORT-001"

      - alert: HighLatency
        expr: histogram_quantile(0.99, agent_request_duration_bucket) >
10
        for: 5m
        labels:
          severity: warning
        annotations:
          summary: "P99 latencia > 10s en AGENT-SUPPORT-001"

      - alert: BudgetAlert
        expr: increase(agent_cost_eur_total[24h]) > 50
        labels:
          severity: critical
        annotations:
          summary: "Coste diario > €50 en AGENT-SUPPORT-001"

      - alert: SecurityViolation
        expr:
increase(agent_guardrail_violations_total{type="data_leakage"}[1h]) > 0
        labels:
          severity: critical
        annotations:
          summary: "Data leakage detectado en AGENT-SUPPORT-001"
```

7.5 Soberania y Compliance

Decision: API vs Self-Hosted

	API Externa	Self-Hosted (vLLM)
Coste por token	$$$	Fijo (GPU)
Setup	0 min	2-8 horas
Latencia	50-200ms	20-100ms (local)
Soberania	NO (datos salen)	SI (datos en tu infra)
Modelos	Ultimos (GPT-4o, Claude, etc)	Open-weight (Qwen, Llama, Phi, etc)
RGPD	Depende del DPA	Compliant por defecto
ENS/NIS2	Riesgo	Compliant
GPU necesaria	NO	SI (24GB+ VRAM)
Scaling	Auto	Manual
Maintenance	0	Updates, monitoring

Cuándo Cada Opcion

```
API EXTERNA:
  - Datos no sensibles (contenido publico, demos)
  - Desarrollo y staging
  - Necesitas el modelo más capaz (Opus, GPT-4o)
  - Trafico bajo (<1000 queries/dia)

SELF-HOSTED:
  - Datos sensibles de clientes
  - Produccion con requisitos regulatorios
  - Trafico alto (coste fijo < coste por token)
  - Latencia critica (<100ms)

HIBRIDO:
  - Clasificacion/routing: API (Haiku, barato)
  - Processing con datos sensibles: Self-hosted
  - Mejor balance coste/compliance
```

RGPD Checklist para Agentes

```
- [ ] LLM inference con datos personales: solo en infra EU
- [ ] No enviar PII a APIs externas (OpenAI, Anthropic)
- [ ] Logs con PII: retencion max 90 dias + auto-purge
- [ ] Derecho al olvido: endpoint /gdpr/erasure funcional
- [ ] Consentimiento: usuario informado de que interactua con IA
- [ ] Data minimization: agente solo accede a datos necesarios
- [ ] DPA firmado con cada proveedor de API LLM
- [ ] Registro de actividades de tratamiento actualizado
```

7.6 Feature Flags para Agentes

Rollout Gradual

```
import os

# Feature flags: controlan que version del agente esta activa
AGENT_FLAGS = {
    "SUPPORT_V2_ENABLED": os.getenv("SUPPORT_V2_ENABLED", "false") == "tr
ue",
    "SUPPORT_V2_PERCENTAGE": int(os.getenv("SUPPORT_V2_PERCENTAGE", "0"))
,
    "SUPPORT_V2_ALLOWED_USERS": os.getenv("SUPPORT_V2_ALLOWED_USERS", "")
.split(","),
}

def should_use_v2(user_id: str) -> bool:
    """Decidir si un usuario usa el agente v2."""
    if not AGENT_FLAGS["SUPPORT_V2_ENABLED"]:
        return False

    # Allowlist: usuarios especificos siempre usan v2 (beta testers)
    if user_id in AGENT_FLAGS["SUPPORT_V2_ALLOWED_USERS"]:
        return True

    # Percentage rollout: N% del trafico
    hash_val = int(hashlib.md5(user_id.encode()).hexdigest(), 16) % 100
    return hash_val < AGENT_FLAGS["SUPPORT_V2_PERCENTAGE"]


# Uso en el endpoint
async def handle_query(user_id: str, query: str):
    if should_use_v2(user_id):
        agent = SupportAgentV2()
    else:
        agent = SupportAgentV1()

    return await agent.run(query)
```

Plan de Rollout Tipico

```
Dia 1:   Flag ON, 0% traffic, solo beta testers (5 usuarios internos)
Dia 3:   5% traffic, monitorizar metricas
Dia 5:   Si metricas OK: 25% traffic
Dia 7:   Si metricas OK: 50% traffic
Dia 10:  Si metricas OK: 100% traffic
Dia 15:  Eliminar v1, limpiar flag

ROLLBACK inmediato si:
  - Error rate > 5%
  - CSAT < 3.5
  - Data leakage detectado (cualquier cantidad)
  - P99 latencia > 15s
```

7.7 Incident Response para Agentes

Runbook

```
# Runbook: AGENT-SUPPORT-001

## Alerta: High Error Rate (> 5%)

1. VERIFICAR: dashboard de errores, identificar tipo predominante
   - Si es timeout: verificar LLM endpoint y Qdrant
   - Si es tool error: verificar APIs externas
   - Si es parsing error: posible cambio en modelo, verificar outputs

2. MITIGAR: si afecta a usuarios
   - Activar fallback: redirigir a formulario de soporte manual
   - Notificar equipo soporte: "agente automatico degradado"

3. DIAGNOSTICAR: revisar trazas de las ultimas queries fallidas
   - OpenTelemetry: filtrar por status=ERROR
   - Comparar con queries exitosas del mismo periodo

4. RESOLVER: aplicar fix
   - Si es infra: restart/scale
   - Si es modelo: rollback prompt a version anterior
   - Si es tool: circuit breaker ya deberia haber actuado

5. POSTMORTEM: tras resolver
   - Anadir regression test para el caso
   - Actualizar runbook si aplica
   - Calcular impacto (queries afectadas, duracion)

## Alerta: Security Violation (data leakage)

1. INMEDIATO: desactivar agente via feature flag (0% traffic)
2. INVESTIGAR: que datos se filtraron, a quien, en que query
3. CONTENER: si datos PII, notificar DPO dentro de 24h
4. FIX: anadir caso adversarial al eval dataset
5. VERIFY: correr eval completo, confirmar fix
6. REACTIVAR: rollout gradual (5% → 25% → 100%)
```

7.8 Ejercicios Practicos

Ejercicio 1: Deploy Strategy

Tu agente procesa 500 queries/dia con datos sensibles (sector salud, RGPD). Disensar la estrategia de deploy: - Serverless, containers, o hibrido? - Que modelo y donde (API vs self-hosted)? - Estimacion de coste mensual

Ejercicio 2: Observabilidad

Configurar OpenTelemetry para un agente que: - Usa 3 tools (search, classify, escalate) - Corre en Docker - Necesita alertas en: error rate, latencia, coste diario Definir: spans, metricas, y reglas de alertas.

Ejercicio 3: Rollout Plan

Disenar el plan de rollout para actualizar tu agente de soporte de v1 a v2: - v2 cambia el modelo de Haiku a Sonnet para respuestas - v2 añade un nuevo tool (get_account_info) - Criterios de go/no-go en cada fase - Plan de rollback

Ejercicio 4: Incident Response

Simular: a las 3am, la alerta “BudgetAlert” se dispara. El coste diario paso de €15 a €180. - Que investigar primero? - Cómo mitigar mientras investigas? - 3 causas posibles y como verificar cada una.
