IACADEMY

ENTERPRISE AGENT ENGINEERING

MÓDULO 08

Alta Disponibilidad

Principal

iacedemy.com — 2026

MÓDULO 08

Alta Disponibilidad

Nivel: Principal / Arquitecto

Autor: Alicia Sanz

Publicación: Julio 2026

Plataforma: iacedemy.com

> “En producción enterprise, el uptime no es un objetivo. Es un contrato.”

7.1 Escalado

Horizontal: Docker Swarm

```yaml
# docker-compose.yml
services:
  agent-worker:
    image: platform/agent-worker:latest
    deploy:
      replicas: 5
      update_config:
        parallelism: 1
        delay: 30s
        failure_action: rollback
      rollback_config:
        parallelism: 1
      resources:
        limits: { memory: 4G, cpus: '4' }
        reservations: { memory: 2G, cpus: '2' }
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:8080/health"]
      interval: 15s
      timeout: 5s
      retries: 3
      start_period: 30s
```

Autoscaling: Kubernetes HPA

```yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: agent-worker-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: agent-worker
  minReplicas: 3
  maxReplicas: 20
  metrics:
    - type: Resource
      resource:
        name: cpu
        target: { type: Utilization, averageUtilization: 70 }
    - type: Pods
      pods:
        metric: { name: agent_queue_depth }
        target: { type: AverageValue, averageValue: "5" }
  behavior:
    scaleUp:
      stabilizationWindowSeconds: 60
      policies:
        - type: Pods
          value: 3
          periodSeconds: 60
    scaleDown:
      stabilizationWindowSeconds: 300
```

GPU Pool Management

```
class GPUPoolManager:
    def __init__(self, gpus: list[dict]):
        self.gpus = gpus  # [{id, vram_gb, model_loaded, utilization}]

    async def allocate(self, model: str, vram_required: int) -> dict | No
ne:
        # Buscar GPU con el modelo ya cargado
        for gpu in self.gpus:
            if gpu["model_loaded"] == model and gpu["utilization"] < 0.8:
                return gpu

        # Buscar GPU libre
        for gpu in self.gpus:
            if gpu["utilization"] < 0.3 and gpu["vram_gb"] >=
vram_required:
                await self._load_model(gpu, model)
                return gpu

        return None  # No hay GPU disponible
```

7.2 Resiliencia

Circuit Breaker con State Machine

```python
from enum import Enum
from time import time

class CBState(Enum):
    CLOSED = "closed"
    OPEN = "open"
    HALF_OPEN = "half_open"

class CircuitBreaker:
    def __init__(self, name: str, max_failures=5, reset_timeout=60,
half_open_max=3):
        self.name = name
        self.max_failures = max_failures
        self.reset_timeout = reset_timeout
        self.half_open_max = half_open_max
        self._state = CBState.CLOSED
        self._failures = 0
        self._last_failure = 0
        self._half_open_successes = 0

    async def execute(self, fn, *args, **kwargs):
        if self._state == CBState.OPEN:
            if time() - self._last_failure > self.reset_timeout:
                self._state = CBState.HALF_OPEN
                self._half_open_successes = 0
            else:
                raise CircuitOpenError(f"{self.name} is OPEN")

        try:
            result = await fn(*args, **kwargs)
            self._on_success()
            return result
        except Exception as e:
            self._on_failure()
            raise

    def _on_success(self):
        if self._state == CBState.HALF_OPEN:
            self._half_open_successes += 1
            if self._half_open_successes >= self.half_open_max:
                self._state = CBState.CLOSED
                self._failures = 0
        else:
            self._failures = 0

    def _on_failure(self):
        self._failures += 1
        self._last_failure = time()
```

```
            if self._failures >= self.max_failures:
                self._state = CBState.OPEN
```

Retry con Exponential Backoff

```
import asyncio, random

async def retry_with_backoff(fn, max_retries=3, base_delay=1.0,
max_delay=30.0):
    for attempt in range(max_retries + 1):
        try:
            return await fn()
        except Exception as e:
            if attempt == max_retries:
                raise
            delay = min(base_delay * (2 ** attempt) + random.uniform(0,
1), max_delay)
            await asyncio.sleep(delay)
```

Bulkhead Pattern

```
class Bulkhead:
    """Limita concurrencia por servicio para evitar cascading
failures."""

    def __init__(self, name: str, max_concurrent: int = 10):
        self.name = name
        self.semaphore = asyncio.Semaphore(max_concurrent)

    async def execute(self, fn, *args, timeout=30, **kwargs):
        async with self.semaphore:
            return await asyncio.wait_for(fn(*args, **kwargs), timeout=ti
meout)
```

7.3 Disaster Recovery

RTO/RPO

```
RTO (Recovery Time Objective):
  - Tier 1 (crítico): < 15 min
  - Tier 2 (importante): < 1 hora
  - Tier 3 (estándar): < 4 horas

RPO (Recovery Point Objective):
  - PostgreSQL: 0 (streaming replication)
  - Qdrant: < 1 hora (snapshot + WAL)
  - Redis: < 5 min (AOF)
  - Logs: 0 (write-ahead)
```

Backup Strategy

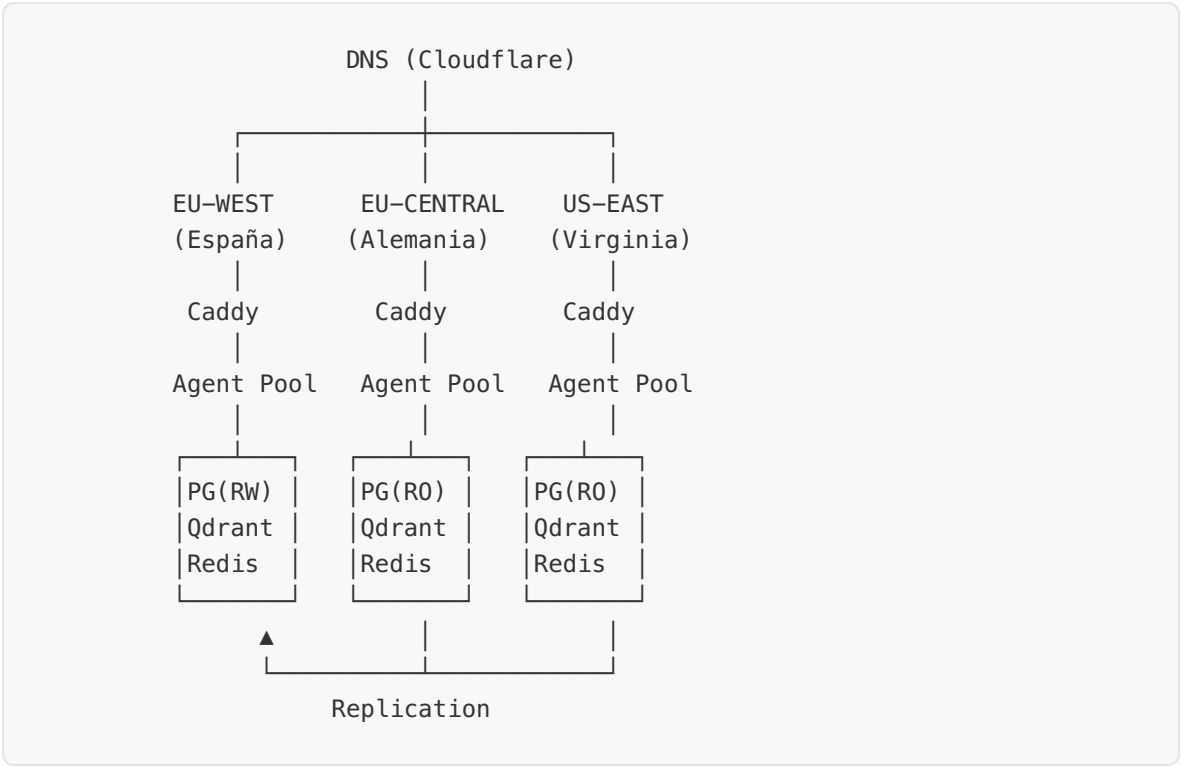
```
# PostgreSQL: streaming replication + daily base backup
# Qdrant: snapshots cada hora
# Redis: AOF + RDB cada 15 min
# Secrets (Vault): sealed backup diario
# Agent configs: Git (versionado)

# Cron de backups
0 * * * * /scripts/backup-qdrant.sh
0 3 * * * /scripts/backup-postgres-base.sh
*/15 * * * * /scripts/backup-redis.sh
0 4 * * * /scripts/backup-vault.sh
```

Restore Procedure

```
#!/bin/bash
# restore.sh — Disaster Recovery
echo "=== DR Restore ==="
echo "1. Restaurando PostgreSQL..."
pg_restore -d platform latest_backup.dump

echo "2. Restaurando Qdrant snapshots..."
curl -X POST http://qdrant:6333/collections/agents/snapshots/recover

echo "3. Restaurando Redis AOF..."
redis-cli CONFIG SET appendonly yes

echo "4. Restaurando Vault..."
vault operator unseal $UNSEAL_KEY
vault operator raft snapshot restore vault_backup.snap

echo "5. Verificando health..."
curl -f http://localhost:8080/health && echo "OK" || echo "FAILED"
```

7.4 Multi-Región

Arquitectura



Data Sovereignty

```
REGION_CONFIG = {
    "eu-west": {
        "db_primary": True,
        "allowed_models": ["qwen3.5-27b", "phi-4"],  # Self-hosted only
        "data_classification": ["confidential", "internal", "public"],
        "compliance": ["RGPD", "ENS", "NIS2"],
    },
    "eu-central": {
        "db_primary": False,
        "allowed_models": ["qwen3.5-27b"],
        "data_classification": ["internal", "public"],
        "compliance": ["RGPD"],
    },
    "us-east": {
        "db_primary": False,
        "allowed_models": ["claude-sonnet-4-6", "gpt-4o"],  # API OK, no
data sovereignty
        "data_classification": ["public"],
        "compliance": [],
    },
}
```

7.5 Ejercicios

Ejercicio 1: Autoscaling

Configura HPA para un deployment de agentes. Escala por CPU y por una métrica custom (queue depth). Prueba con carga simulada.

Ejercicio 2: Circuit Breaker

Implementa un circuit breaker completo con los 3 estados. Simula fallos y verifica que se abre, espera, y prueba con half-open.

Ejercicio 3: DR Plan

Escribe un DR plan para tu plataforma: RTO/RPO por servicio, procedimiento de backup, restore, y prueba de restore (fire drill).

Ejercicio 4: Multi-Región

Diseña una arquitectura de 2 regiones (EU + US) con data sovereignty. Define: qué datos se replican, qué modelos en cada región, cómo se enruta el tráfico.
