

Gestión de Incidentes y Recuperación - Día 3

Change Status

pending 40 min

Learning Objectives

- 1 Aprender manejo de incidentes en producción
- 2 Comprender estrategias de recuperación
- 3 Conocer importancia de post-mortems
- 4 Entender preparación para escenarios de disaster

Theory Practice Evidence Quiz

<> Practical exercise to apply the concepts learned.

Ejercicio: Crear plan de respuesta a incidentes

Definir runbook de incidentes:

```
# runbook.py
from typing import Dict, List, Callable
from datetime import datetime, timedelta
import logging

logger = logging.getLogger('incident_response')

class IncidentRunbook:
    """Runbook automatizado para respuesta a incidentes"""

    def __init__(self):
        self.incident_types = self._define_incident_types()
        self.escalation_matrix = self._define_escalation()

    def _define_incident_types(self) -> Dict:
        """Definir tipos de incidentes y respuestas"""
        return {
            'pipeline_down': {
                'severity': 'CRITICAL',
                'auto_response': True,
                'timeout': timedelta(minutes=15),
                'steps': [
                    'check_airflow_scheduler',
                    'check_database_connectivity',
                    'restart_failed_services',
                    'verify_pipeline_recovery'
                ]
            },
            'data_quality_degraded': {
                'severity': 'HIGH',
                'auto_response': False,
                'timeout': timedelta(hours=1),
                'steps': [
                    'isolate_affected_data',
                    'check_upstream_sources',
                    'implement_data_filters',
                    'notify_data_consumers'
                ]
            },
            'performance_degraded': {
                'severity': 'MEDIUM',
                'auto_response': True,
                'timeout': timedelta(hours=2),
                'steps': [
                    'check_resource_usage',
                    'scale_resources_if_needed',
                    'optimize_running_queries',
                    'monitor_recovery'
                ]
            }
        }

    def _define_escalation(self) -> Dict:
        """Matriz de escalación por tiempo y severidad"""
        return {
            'CRITICAL': {
                '5min': 'alert_lead_engineer',
                '15min': 'alert_engineering_manager',
                '30min': 'alert_vp_engineering'
            },
            'HIGH': {
                '15min': 'alert_lead_engineer',
```



```
def _verify_pipeline_status(self):
```

```
return {'pipelines_running': 5, 'pipelines_failed': 0}

def _isolate_bad_data(self):
    return {'isolated_records': 150, 'quarantined': True}

def _check_system_resources(self):
    return {'cpu_percent': 45, 'memory_percent': 60, 'disk_percent': 30}

def _scale_resources(self):
    return {'scaled_up': ['airflow-worker'], 'new_instances': 2}

def _verify_resolution(self, incident_type, context):
    # Lógica para verificar si incidente está resuelto
    return True

def _parse_time_to_seconds(self, time_str):
    # Convertir "5min", "2h" a segundos
    if 'min' in time_str:
        return int(time_str.replace('min', '')) * 60
    elif 'h' in time_str:
        return int(time_str.replace('h', '')) * 3600
    return 0
```

Simular manejo de incidente:

```
# Simular respuesta a incidente
runbook = IncidentRunbook()

# Simular incidente de pipeline caído
incident_context = {
    'triggered_by': 'alert_pipeline_down',
    'affected_components': ['etl_pipeline', 'data_warehouse'],
    'start_time': datetime.now(),
    'symptoms': ['scheduler_not_responding', 'tasks_queued']
}

# Ejecutar runbook
response = runbook.handle_incident('pipeline_down', incident_context)

print("Respuesta a incidente:")
print(f"Tipo: {response['incident_type']}")
print(f"Severidad: {response['severity']}")
print(f"Resuelto: {response['resolved']}")
print(f"Duración: {response['duration_seconds']:.1f}s")
print(f"Pasos ejecutados: {len(response['steps_executed'])}")

for step in response['steps_executed']:
    status = "✅" if step['success'] else "❌"
    print(f"  {status} {step['step']}")
```

Template de post-mortem:

```
def create_post_mortem_template(incident_data):
    """Crear template de post-mortem basado en incidente"""

    template = f"""
# Post-Mortem: {incident_data['title']}

## Executive Summary
{incident_data.get('summary', 'Incident description')}

## Timeline
- **Detection**: {incident_data.get('detection_time', 'Unknown')}
- **Start**: {incident_data.get('start_time', 'Unknown')}
- **Resolution**: {incident_data.get('end_time', 'Unknown')}
- **Duration**: {incident_data.get('duration', 'Unknown')}

## Impact
- **Users Affected**: {incident_data.get('users_affected', 0)}
- **Business Impact**: {incident_data.get('business_impact', 'Unknown')}
- **Data Loss**: {incident_data.get('data_loss', 'None')}

## Root Cause Analysis
{incident_data.get('root_cause', 'To be determined')}

## Resolution Steps
{chr(10).join(f"- {step}" for step in incident_data.get('resolution_steps', []))}

## Lessons Learned
### What went well
{chr(10).join(f"- {item}" for item in incident_data.get('went_well', []))}

### What could be improved
{chr(10).join(f"- {item}" for item in incident_data.get('improvements', []))}

## Action Items
{chr(10).join(f"- [ ] {item}" for item in incident_data.get('action_items', []))}

## Prevention Measures
{chr(10).join(f"- [ ] {item}" for item in incident_data.get('prevention', []))}
    """
```

```
---
*Post-mortem completed on {datetime.now().strftime('%Y-%m-%d')}*
"""

return template
```

Verificación: ¿Cuál es la diferencia entre un incidente que requiere respuesta inmediata vs uno que puede esperar? ¿Cómo decidir cuándo escalar un incidente a niveles superiores?

Requerimientos:

- Sistema de alertas configurado (PagerDuty, OpsGenie)
- Runbooks documentados y accesibles
- Equipo de respuesta definido con roles claros