Proyecto Final

Curso de Sistemas Operativos y Laboratorio

Título del proyecto

MicroFlow: Arquitectura de microservicios con gestión de procesos y recursos aplicada a TelcoNova SupportSuite

Miembros del equipo

Ricardo Contreras Garzón
Valentina Muñoz Rincón
Juan Felipe Escobar Rendón

Resumen

Este proyecto desarrolla e implementa una arquitectura de microservicios distribuida basada en Amazon Web Services (AWS) siguiendo el patrón Event-Driven Architecture (EDA), aplicado al caso de estudio **TelcoNova** SupportSuite - un sistema integral de gestión de soporte técnico para empresas de telecomunicaciones.

La solución comprende tres microservicios independientes especializados en: (1) autenticación y autorización de usuarios, (2) gestión completa del ciclo de vida de órdenes de trabajo, y (3) seguimiento en tiempo real con sistema de notificaciones inteligentes. Estos servicios implementan comunicación híbrida mediante APIs REST síncronas y colas de mensajería asíncronas (SNS), garantizando acoplamiento débil, alta disponibilidad y tolerancia a fallos.

a infraestructura utiliza servicios nativos de AWS incluyendo Elastic Container Service (ECS) para orquestación, Elastic Container Registry (ECR) para gestión de imágenes, API Gateway como punto de entrada unificado, Relational Database Service (RDS) PostgreSQL, y CloudWatch para observabilidad básica. Se complementa con herramientas especializadas como Prometheus y

Grafana para análisis detallado de métricas de sistema operativo (utilización de CPU, memoria, I/O de disco, tráfico de red).

El proyecto incluye un pipeline completo de CI/CD implementado con GitHub Actions, demostrando la aplicación práctica de conceptos fundamentales de Sistemas Operativos: gestión distribuida de procesos (contenedores ECS), comunicación interprocesos avanzada (IPC via message queues y REST APIs), virtualización a nivel de SO (containerización Docker), scheduling inteligente (ECS task scheduling con placement strategies), y monitorización integral de recursos del sistema

Introducción

El proyecto aborda dos niveles de problemas:

Nivel de Negocio (TelcoNova):

- **Gestión ineficiente de incidencias**: Las empresas de telecomunicaciones manejan miles de órdenes de trabajo diarias sin trazabilidad adecuada.
- **Comunicación fragmentada**: Notificaciones inconsistentes entre clientes, técnicos y supervisores
- **Falta de análisis histórico**: Datos dispersos impiden identificar patrones y optimizar operaciones

Nivel Técnico (Sistemas Operativos):

- **Gestión de procesos distribuidos**: Cómo manejar múltiples servicios como procesos independientes y coordinados
- **Comunicación interprocesos eficiente**: Implementar IPC entre servicios distribuidos usando patrones síncronos y asíncronos con garantías de entrega y manejo de fallos.
- **Optimización de recursos**: Monitorear, analizar y optimizar utilización de CPU, memoria, I/O y red en entornos contenerizados dinámicos.

Importancia en el contexto tecnológico:

- **Microservicios como estándar**: Arquitectura dominante en empresas tech (Netflix, Amazon, Uber).
- **Cloud-first approach**: 90% de empresas usan cloud computing, siendo AWS líder del mercado.
- **Contenedores como unidad de despliegue**: Docker/Kubernetes son tecnologías esenciales en DevOps moderno.

- **Observabilidad como Requisito Crítico**: Sistemas distribuidos modernos requieren observabilidad proactiva. Prometheus procesa más de 10^12 muestras métricas diarias en despliegues empresariales, mientras que Grafana cuenta con más de 10 millones de instalaciones activas (CNCF Landscape, 2022).

Relevancia Académica y Pedagógica

Este proyecto integra conceptos teóricos de Sistemas Operativos con implementación práctica en entornos distribuidos:

- **Aplicación práctica de SO:** Materialización de conceptos abstractos de procesos, IPC, virtualización y scheduling en contexto tecnológico real.
- **Evolución hacia sistemas distribuidos:** Comprensión de cómo principios de SO tradicionales se extienden a arquitecturas cloud-native.
- **Performance engineering:** Análisis cuantitativo de rendimiento usando herramientas profesionales de observabilidad.
- **DevOps integration:** Integración de principios de desarrollo y operaciones aplicando conceptos fundamentales de SO.

Antecedentes y marco teórico

Arquitecturas de Software y Microservicios

Evolución Arquitectónica

La arquitectura de software ha evolucionado desde sistemas monolíticos hacia patrones distribuidos. Richardson (2018) identifica que las arquitecturas monolíticas presentan limitaciones fundamentales de escalabilidad, donde el acoplamiento estrecho entre componentes genera cuellos de botella operacionales y tecnológicos.

Los microservicios emergen como patrón arquitectónico que descompone aplicaciones en servicios pequeños, independientemente desplegables, comunicándose via APIs bien definidas (Newman, 2021). Esta aproximación permite:

- **Escalabilidad independiente:** Cada servicio escala según demanda específica.
- **Diversidad tecnológica:** Selección de stack tecnológico óptimo por servicio.
- **Aislamiento de fallos:** Fallas localizadas no propagan al sistema completo.
- **Desarrollo paralelo:** Equipos independientes desarrollan servicios autónomos.

Event-Driven Architecture (EDA)

Michelson (2006) define EDA como patrón donde servicios reaccionan a eventos significativos del negocio, promoviendo acoplamiento temporal débil. Esta arquitectura facilita:

- **Comunicación asíncrona:** Servicios procesan eventos independientemente.
- **Escalabilidad elástica:** Procesamiento distribuido de carga variable.
- **Resiliencia:** Tolerancia a fallos mediante event sourcing y replay capabilities.

Computación en la Nube y Amazon Web Services

Paradigma Cloud Computing

Mell & Grance (2011) del NIST definen cloud computing como modelo para acceso ubicuo, conveniente y bajo demanda a recursos computacionales configurables. Las características esenciales incluyen:

- **Auto-servicio bajo demanda:** Provisioning automático sin intervención humana.
- **Acceso amplio via red:** Disponibilidad mediante protocolos estándar.
- **Pooling de recursos:** Recursos multi-tenant con asignación dinámica.
- **Elasticidad rápida:** Escalamiento automático según demanda.
- **Servicio medido:** Monitoreo, control y reporte de utilización.

Amazon Web Services (AWS)

AWS se estableció como líder del mercado cloud proporcionando más de 200 servicios completamente gestionados. Barr & Cabrera (2019) destacan servicios fundamentales relevantes al proyecto:

Amazon Elastic Container Service (ECS): Servicio de orquestación de contenedores completamente gestionado que elimina necesidad de instalar y operar infraestructura de clustering. Proporciona scheduling inteligente, auto-scaling e integración nativa con servicios AWS.

Amazon API Gateway: Servicio completamente gestionado para crear, publicar, mantener, monitorear y asegurar APIs REST y WebSocket. Implementa throttling, caching, autenticación y documentación automática.

Amazon RDS: Servicio de base de datos relacional gestionado que automatiza tareas administrativas como provisioning de hardware, setup de database, patching y backups.

Contenedores y Virtualización

Virtualización a Nivel de Sistema Operativo

Soltesz et al. (2007) distinguen contenedores como forma de virtualización de SO que permite múltiples instancias aisladas de userspace ejecutándose en kernel único. Esta aproximación ofrece ventajas sobre virtualización tradicional:

- **Overhead reducido:** Eliminación de hypervisor reduce latencia y consumo de recursos.
- **Densidad superior:** Mayor número de instancias por host físico.
- **Startup rápido:** Inicialización en milisegundos vs minutos de VMs tradicionales.

Docker como Plataforma de Contenedores

Merkel (2014) analiza Docker como plataforma que democratiza contenedores mediante abstracciones simples. Docker Engine implementa:

- **Layered filesystem:** Optimización de almacenamiento mediante copy-on-write.
- **Process isolation:** Namespaces y cgroups para aislamiento de recursos.
- **Portabilidad:** "Build once, run anywhere" mediante imágenes inmutables.

Comunicación Interprocesos en Sistemas Distribuidos

Patrones de Comunicación

Tanenbaum & Van Steen (2016) categorizan comunicación en sistemas distribuidos:

Comunicación Síncrona: Client bloquea hasta recibir respuesta del servidor. REST APIs implementan este patrón proporcionando semantics request-response familiares, pero introduciendo acoplamiento temporal.

Comunicación Asíncrona: Sender continúa procesamiento sin esperar respuesta inmediata. Message queues y event streams implementan este patrón, reduciendo acoplamiento, pero complicando manejo de errores y consistencia.

Message Queues y Event Streaming

Kleppmann (2017) analiza sistemas de mensajería como backbone de arquitecturas distribuidas. Amazon SNS implementa patrón publish-subscribe donde:

- **Publishers** emiten eventos sin conocimiento de subscribers.
- **Topics** actúan como canales de distribución.

- **Subscribers** reciben eventos relevantes asíncronamente.
- **Delivery guarantees** aseguran procesamiento confiable.

Monitorización y Observabilidad

Observabilidad en Sistemas Distribuidos

Majors et al. (2022) definen observabilidad como capacidad de inferir estado interno de sistema mediante outputs externos. En sistemas distribuidos, observabilidad comprende:

- **Metrics:** Agregaciones numéricas de datos a lo largo del tiempo.
- **Logs:** Registros discretos de eventos con timestamps.
- **Traces:** Representación de requests atravesando múltiples servicios.

Prometheus y Grafana

Prometheus implementa modelo pull-based para recolección de métricas, proporcionando (Godard, 2019):

- **Time-series database:** Almacenamiento eficiente de métricas temporales.
- **Query language (PromQL):** DSL para análisis y alerting.
- **Service discovery:** Descubrimiento automático de targets.

Grafana complementa Prometheus proporcionando visualización avanzada y dashboards interactivos.

Integración Continua y Despliegue Continuo (CI/CD)

DevOps y Automatización

Kim et al. (2016) en "The DevOps Handbook" establecen que CI/CD elimina toil operacional mediante automatización. GitHub Actions implementa CI/CD nativo en plataforma de desarrollo, proporcionando:

- **Workflow automation:** Pipelines declarativos vía YAML.
- **Matrix builds:** Ejecución paralela en múltiples ambientes.
- **Secrets management:** Manejo seguro de credenciales.
- **Ecosystem integration:** Conectores nativos con servicios cloud.

Relación con Sistemas Operativos

Gestión de Procesos Distribuidos

Los microservicios representan evolución natural de procesos de SO tradicionales. Silberschatz et al. (2018) establecen que procesos requieren:

- **Process Control Block (PCB):** Metadatos de estado y recursos.
- **Scheduling:** Asignación de CPU time.
- **Synchronization:** Coordinación entre procesos concurrentes.

En arquitecturas contenerizadas, ECS proporciona análogos distribuidos:

- **Task Definition:** Especificación declarativa análoga a PCB.
- **Service Mesh:** Coordinación de comunicación interprocesos.

Virtualización y Aislamiento de Recursos

Contenedores implementan virtualización de SO mediante kernel features:

- **Namespaces:** Aislamiento de system resources (PID, network, filesystem).
- **Control Groups (cgroups):** Limitación y accounting de recursos.
- **Copy-on-Write filesystems:** Optimización de storage y memoria.

Objetivos

Objetivo Principal

Desarrollar, implementar y desplegar una arquitectura completa de microservicios en Amazon Web Services aplicada a TelcoNova SupportSuite, que demuestre la implementación práctica y análisis cuantitativo de conceptos fundamentales de Sistemas Operativos, incluyendo gestión distribuida de procesos, comunicación interprocesos avanzada, virtualización mediante contenedores, scheduling inteligente, y monitorización integral de recursos del sistema.

Específicos:

1. Diseño e Implementación de Arquitectura de microservicios Distribuidos

- **Auth-Service**: Implementar servicio de autenticación JWT con autorización basada en roles (RBAC), incluyendo gestión completa de usuarios, perfiles y sesiones seguras

- **WorkOrder-Service**: Desarrollar servicio de gestión integral de órdenes de trabajo con workflow automatizado de estados, asignación inteligente de técnicos y trazabilidad completa.
- **Tracking-Service**: Construir servicio de seguimiento en tiempo real con gestión de evidencias multimedia, sistema de notificaciones multicanal e histórico detallado de actividades

2. Comunicación Interprocesos Distribuida

- Establecer comunicación **síncrona** mediante APIs REST.
- Implementar comunicación asíncrona event-driven usando Amazon SNS para eventos críticos de negocio (creación de órdenes, asignación de técnicos, cambios de estado, completación de trabajos)

3. Gestión Avanzada de Procesos y Recursos

- Aplicar **scheduling inteligente** usando ECS con placement strategies optimizadas.
- Configurar **auto-scaling** basado en métricas de CPU, memoria, y latencia de aplicación.

4. Persistencia y Gestión de Datos

- Integrar **Amazon RDS PostgreSQL.**

5. Seguridad y Control de Acceso

- Implementar **autenticación JWT.**
- Gestionar **secrets y configuración** usando Parameter Store y Secrets Manager con KMS encryption.

6. Monitorización Integral de Recursos

- Desplegar stack Prometheus + Grafana para métricas custom de aplicación y métricas detalladas de sistema operativo
- Integrar AWS CloudWatch para métricas nativas de infraestructura, alarmas automatizadas y log aggregation
- Implementar distributed tracing para análisis de latencia end-to-end e identificación de bottlenecks

7. Pipeline CI/CD Automatizado

- Implementar **GitHub Actions** con multi-stage pipeline (build, test, security scan).

Metodología

Stack Tecnológico Simplificado

Plataforma Cloud y Servicios AWS

- **Amazon ECS:** Orquestación de los 3 contenedores
- **Amazon ECR**: Registry privado para imágenes Docker
- **Amazon API Gateway**: Punto de entrada unificado con rate limiting, request/response transformation y documentación automática.
- **Amazon RDS PostgreSQL**: Base de datos principal con Multi-AZ
- **Amazon SNS:** Event messaging entre servicios
- **Amazon S3**: Storage para evidencias y documentos (Opcional)
- **AWS Secrets Manager + Parameter Store**: Configuración y secrets
- **Amazon CloudWatch**: Logging y métricas básicas
- **Prometheus + Grafana**: Métricas custom y dashboards

Desarrollo

- **Java 21 + Spring Boot**: Framework para los 3 microservicios
- **Docker**: Containerización
- **GitHub Actions**: CI/CD pipeline

Componentes de cada servicio:

1. Auth-Service (Autenticación y Autorización)

Responsabilidades:

- **Gestión de usuarios**: Registro, login, perfil de usuario
- **Autenticación JWT**: Token generation, validation, refresh
- **Autorización RBAC**: Roles (Cliente, Técnico, Supervisor, Admin)

APIs REST:

- POST /auth/register - Registro de nuevo usuario
- `POST /auth/login` - Autenticación
- `POST /auth/logout` - Cerrar sesión
- `POST /auth/refresh` - Renovar token
- `GET /auth/profile` - Obtener perfil
- POST /auth/password/change - Cambio de contraseña
- GET /auth/validate - Validación de token (uso interno)
-

Eventos Publicados:

- `user.registered` - Usuario registrado
- `user.login` - Usuario autenticado
- `user.authenticated - Login exitoso`
- `user.password.changed` - Cambio de contraseña
- `session.invalidated - Sesión invalidada`

2. WorkOrder-Service (Gestión de Órdenes)

Responsabilidades:

- **CRUD de órdenes**: Crear, leer, actualizar órdenes de trabajo
- **Gestión de técnicos**: Perfiles, especialidades
- **Workflow de estados**: CREATED → ASSIGNED → IN_PROGRESS → COMPLETED/CANCELLED

APIs REST:

- `POST /orders` - Crear orden
- `GET /orders` - Listar órdenes (con filtros)
- `GET /orders/{id}` - Obtener orden específica
- `PUT /orders/{id}` - Actualizar orden
- PUT /orders/{id}/status - Cambiar estado de orden
- `POST /orders/{id}/assign` - Asignar técnico
- `GET /technicians` - Listar técnicos disponibles

Eventos Publicados:

- `order.created` - Nueva orden creada
- `order.assigned` - Orden asignada a técnico
- `order.status_changed` - Cambio de estado de orden
- `technician.assigned` - Técnico asignado a orden
- `order.completed` - Orden completada exitosamente
- `order.cancelled` - Orden cancelada

Eventos Consumidos:

- `tracking.work_started` - Trabajo iniciado
- `tracking.work_completed` - Trabajo completado
- `tracking.evidence.uploaded` - Evidencia subida

3. Tracking-Service (Seguimiento y Notificaciones)

Responsabilidades:

- **Seguimiento en tiempo real**: Updates de progreso
- **Gestión de evidencias**: Upload de fotos y documentos

- **Notificaciones multicanal**: Email, SMS, push notifications
- **Histórico de actividades**: Timeline completo de cada orden

APIs REST:

- `POST /tracking/{orderId}/update` - Actualizar progreso
- `GET /tracking/{orderId}` - Obtener histórico
- `POST /tracking/{orderId}/evidence` - Subir evidencia
- `GET /tracking/{orderId}/evidence` - Listar evidencias
- DELETE /evidence/{evidenceId} - Eliminar evidencia
- `POST /notifications/send` - Enviar notificación
- `GET /notifications/{userId}` - Notificaciones de usuario

Eventos Publicados:

- `tracking.work_started` - Trabajo iniciado
- `tracking.work_completed` - Trabajo completado
- `tracking.evidence_uploaded` - Evidencia subida
- `notification.sent` - Notificación enviada
- `tracking.progress.updated` - Actualización de progreso

Eventos Consumidos:

- `order.created` - Nueva orden (enviar notificaciones)
- `order.assigned` - Orden asignada (notificar técnico)
- `order.status_changed` - Cambio de estado (notificar partes interesadas)

Actividades necesarias para cumplir los objetivos:

Fase 1: Infraestructura Base

- Setup AWS

Fase 2: Ajustes de los microservicios

- Dockerización
- Subida de las imágenes a ECR (Github actions)
- Integración con algunos de los servicios de AWS.
- Api gateway

Fase 3: Integración y Despliegue

- ECS deployment
- Inter-service communication, event messaging

Fase 4: Monitoreo

- Prometheus + Grafana deployment

Fase 5: Refactorización y ajustes

Cronograma

	Semana 1	Semana 2	Semana 3	Semana 4	Semana 5	Semana 6	Semana 7	Semana 8	Semana 9
Propuesta del proyecto									
Infraestructura Base AWS									
Ajustes de los microservicios									
Integración y Despliegue									
Monitoreo									
Refactorización y ajustes									
Revisión componentes finales AWS									

Referencias

- Newman, S. (2021). Building Microservices: Designing Fine-Grained Systems (2nd Edition). O'Reilly Media.
- Kleppmann, M. (2017). Designing Data-Intensive Applications. O'Reilly Media.
- Fowler, M. (2014). Microservices Architecture Pattern. Retrieved from: https://martinfowler.com/articles/microservices.html
- Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). Operating System Concepts (10th Edition). John Wiley & Sons.
- Tanenbaum, A. S., & Bos, H. (2014). Modern Operating Systems
- Cloud Native Computing Foundation. (2022). CNCF Annual Survey 2022. Retrieved from https://www.cncf.io/reports/cncf-annual-survey-2022/
- Richardson, C. (2018). Microservices Patterns: With Examples in Java. Manning Publications.
- Michelson, B. M. (2006). Event-Driven Architecture Overview. Patricia Seybold Group.
- Mell, P., & Grance, T. (2011). The NIST Definition of Cloud Computing. National Institute of Standards and Technology Special Publication 800-145.
- Barr, J., & Cabrera, L. (2019). AWS Well-Architected Framework. Amazon Web Services. https://aws.amazon.com/architecture/well-architected/
- Soltesz, S., Pötzl, H., Fiuczynski, M. E., Bavier, A., & Peterson, L. (2007). Container-based operating system virtualization: A scalable, high-performance alternative to hypervisors. ACM SIGOPS Operating Systems Review, 41(3), 275-287.
- Merkel, D. (2014). Docker: Lightweight Linux containers for consistent development and deployment. Linux Journal, 2014(239), Article 2.
- Tanenbaum, A. S., & Van Steen, M. (2016). Distributed Systems: Principles and Paradigms (3rd Edition). Pearson.

- Majors, C., Fong-Jones, L., & Miranda, G. (2022). Observability Engineering: Achieving Production Excellence. O'Reilly Media.

- Godard, B. (2019). Prometheus: Up & Running: Infrastructure and Application Performance Monitoring.

- Kim, G., Humble, J., Debois, P., & Willis, J. (2016). The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations. IT Revolution Press. O'Reilly Media.