SISTEMA DE MICROSERVICIOS CON COMUNICACIÓN SÍNCRONA Y ASÍNCRONA

JOHN FABER NAVIA NARVAEZ

SOFIA MORENO TRULLO

HENERSSON ESTID COBO CAICEDO

LIZ YURANY CASTIBLANCO SIERRA

YEIMY DAMARYS JOAQUI JOAQUI

Taller número en el curso Laboratorio de software II

Profesor:

BRAYAN DANIEL PERDOMO URBANO

Universidad del Cauca

Facultad de Ingeniería Electrónica y Telecomunicaciones

Departamento de Sistemas

Laboratorio de software II

Popayán, Oct 2025

JOHN FABER NAVIA NARVAEZ,

SOFIA MORENO TRULLO

HENERSSON ESTID COBO CAICEDO

LIZ YURANY CASTIBLANCO SIERRA

YEIMY DAMARYS JOAQUI JOAQUI

SISTEMA DE MICROSERVICIOS CON COMUNICACIÓN SÍNCRONA Y ASÍNCRONA

Taller presentado en el curso de Laboratorio de software II

Estudiantes del:

Programa de Ingeniería de Sistemas

Profesor:

BRAYAN DANIEL PERDOMO URBANO

Popayán

2025

Contenido

Lista de Figuras

Lista de Tablas

Lista de Siglas

SISTEMA DE MICROSERVICIO CON COMUNICACIÓN SÍNCRONA Y ASÍNCRONA

1.- Introducción

Este documento presenta la documentación del sistema de microservicios que implementa dos patrones de comunicación:

- **Comunicación Síncrona (REST)**: Para operaciones que requieren respuesta inmediata
- **Comunicación Asíncrona (RabbitMQ)**: Para procesamiento diferido y desacoplamiento

2.-Arquitectura del Sistema

2.1.- Componentes

El sistema está compuesto por los siguientes microservicios:

Servicio	Puerto	Responsabilidad	Tecnología
Submission Service	8081	Recepción de anteproyectos, persistencia en DB, publicación en RabbitMQ	Spring Boot + PostgreSQL
Notification Service	8082	Consumo de mensajes RabbitMQ, endpoint REST para notificaciones síncronas	Spring Boot + RabbitMQ
Identity Service	8083	Autenticación y generación de JWT (mock básico)	Spring Boot

API Gateway	8080	Proxy, validación JWT, enrutamiento	Spring Cloud Gateway
PostgreSQL	5432	Base de datos de anteproyectos	PostgreSQL 15
RabbitMQ	5672/15672	Message broker para comunicación asíncrona	RabbitMQ 3.13

Tabla 2.1 Componentes del sistema

2.2.-Flujos de Comunicación

Flujo Síncrono

1. Cliente → API Gateway (POST /api/submissions con JWT)
2. Gateway valida JWT con Identity Service
3. Gateway → Submission Service
4. Submission Service → PostgreSQL (persistencia)
5. Submission Service → Notification Service (REST síncrono)
6. Respuesta inmediata al cliente

Flujo Asíncrono

1. Cliente → API Gateway (POST /api/submissions)
2. Submission Service → PostgreSQL (persistencia)
3. Submission Service → RabbitMQ (publica mensaje)
4. Cliente recibe respuesta 202 Accepted
5. Notification Service consume mensaje de RabbitMQ (asíncrono)
6. Notification Service procesa y registra log

3.- Especificación de APIs(open API)

3.1.- Submission Service API

TypeScript

```
openapi: 3.0.3
info:
  title: Submission Service API
  description: API para gestión de anteproyectos de grado
  version: 1.0.0
  contact:
    name: Equipo de Desarrollo
    email: desarrollo@unicauca.edu.co

servers:
  - url: http://localhost:8081
    description: Servidor local

paths:
  /api/submissions:
    post:
      summary: Crear un nuevo anteproyecto
      description: |
        Recibe un anteproyecto, lo persiste en PostgreSQL,
        publica evento en RabbitMQ y opcionalmente llama
        a Notification Service vía REST.
      operationId: createSubmission
      tags:
        - Submissions
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/SubmissionRequest'
            example:
              titulo: "Sistema de Recomendación con ML"
              resumen: "Implementación de algoritmos de recomendación
usando aprendizaje automático"
              autoresEmails:
                - "estudiante1@unicauca.edu.co"
                - "estudiante2@unicauca.edu.co"
              directorId: 101
              estudiante1Id: 201
              estudiante2Id: 202
      responses:
        '201':
          description: Anteproyecto creado exitosamente
          content:
            application/json:
              schema:
```

```
                $ref: '#/components/schemas/SubmissionResponse'
              example:
                proyectoId: 1
                estado: "EN_PROCESO"
        '400':
          description: Datos inválidos en la petición
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/ErrorResponse'
        '500':
          description: Error interno del servidor
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/ErrorResponse'

  /ping:
    get:
      summary: Health check del servicio
      description: Verifica que el servicio está disponible
      operationId: ping
      tags:
        - Health
      responses:
        '200':
          description: Servicio disponible
          content:
            text/plain:
              schema:
                type: string
                example: "pong"

components:
  schemas:
    SubmissionRequest:
      type: object
      required:
        - titulo
        - resumen
        - autoresEmails
        - directorId
        - estudiante1Id
      properties:
        titulo:
          type: string
```

```
        minLength: 5
        maxLength: 200
        description: Título del anteproyecto
      resumen:
        type: string
        minLength: 20
        maxLength: 1000
        description: Resumen del anteproyecto
      autoresEmails:
        type: array
        items:
          type: string
          format: email
        minItems: 1
        maxItems: 2
        description: Emails de los autores
      directorId:
        type: integer
        format: int64
        description: ID del director del proyecto
      estudiante1Id:
        type: integer
        format: int64
        description: ID del primer estudiante
      estudiante2Id:
        type: integer
        format: int64
        nullable: true
        description: ID del segundo estudiante (opcional)

  SubmissionResponse:
    type: object
    properties:
      proyectoId:
        type: integer
        format: int64
        description: ID del proyecto creado
      estado:
        type: string
        enum: [EN_PROCESO, APROBADO, RECHAZADO]
        description: Estado actual del proyecto

  ErrorResponse:
    type: object
    properties:
      timestamp:
```

```
          type: string
          format: date-time
        status:
          type: integer
        error:
          type: string
        message:
          type: string
        path:
          type: string
```

3.2.- Notification Service API

Open API 3.0 Specification

None

```
openapi: 3.0.3
info:
  title: Notification Service API
  description: |
    Servicio de notificaciones que soporta:
    - Recepción de notificaciones vía REST (síncrono)
    - Consumo de mensajes desde RabbitMQ (asíncrono)
  version: 1.0.0

servers:
  - url: http://localhost:8082
    description: Servidor local

paths:
  /api/notifications:
    post:
      summary: Recibir notificación síncrona
      description: |
        Endpoint REST para recibir notificaciones desde Submission Service
        en el flujo síncrono.
      operationId: receiveNotification
      tags:
        - Notifications
      requestBody:
```

```
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/NotificationRequest'
            example:
              proyectoId: 1
              titulo: "Sistema de Recomendación con ML"
              autores:
                - "estudiante1@unicauca.edu.co"
                - "estudiante2@unicauca.edu.co"
              tipo: "SUBMISSION_CREATED"
      responses:
        '200':
          description: Notificación procesada exitosamente
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/NotificationResponse'
              example:
                message: "Notificación recibida y procesada"
                timestamp: "2025-10-16T01:24:45.000Z"
        '400':
          description: Datos inválidos
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/ErrorResponse'

  /health:
    get:
      summary: Health check del servicio
      operationId: healthCheck
      tags:
        - Health
      responses:
        '200':
          description: Servicio disponible
          content:
            application/json:
              schema:
                type: object
                properties:
                  status:
                    type: string
                    example: "UP"
```

```
components:
  schemas:
    NotificationRequest:
      type: object
      required:
        - proyectoId
        - titulo
        - tipo
      properties:
        proyectoId:
          type: integer
          format: int64
        titulo:
          type: string
        autores:
          type: array
          items:
            type: string
            format: email
        tipo:
          type: string
          enum: [SUBMISSION_CREATED, SUBMISSION_APPROVED,
SUBMISSION_REJECTED]

    NotificationResponse:
      type: object
      properties:
        message:
          type: string
        timestamp:
          type: string
          format: date-time

    ErrorResponse:
      type: object
      properties:
        timestamp:
          type: string
          format: date-time
        status:
          type: integer
        error:
          type: string
        message:
          type: string
```

3.3.- Identity Service API(Mock)

None

```
openapi: 3.0.3
info:
  title: Identity Service API (Mock)
  description: Servicio de autenticación básico para generación y
validación de JWT
  version: 1.0.0

servers:
  - url: http://localhost:8083
    description: Servidor local

paths:
  /auth/register:
    post:
      summary: Registrar nuevo usuario
      operationId: registerUser
      tags:
        - Authentication
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/RegisterRequest'
            example:
              username: "estudiante1"
              email: "estudiante1@unicauca.edu.co"
              password: "password123"
      responses:
        '201':
          description: Usuario registrado exitosamente
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/AuthResponse'
```

```
/auth/login:
  post:
    summary: Login de usuario
    description: Genera un JWT válido por 24 horas
    operationId: loginUser
    tags:
      - Authentication
    requestBody:
      required: true
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/LoginRequest'
          example:
            username: "estudiante1"
            password: "password123"
    responses:
      '200':
        description: Login exitoso
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/AuthResponse'
            example:
              token: "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
              expiresIn: 86400

/auth/validate:
  get:
    summary: Validar token JWT
    description: Usado por API Gateway para validar tokens
    operationId: validateToken
    tags:
      - Authentication
    security:
      - bearerAuth: []
    responses:
      '200':
        description: Token válido
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/ValidationResponse'
      '401':
        description: Token inválido o expirado
```

```
components:
  schemas:
    RegisterRequest:
      type: object
      required:
        - username
        - email
        - password
      properties:
        username:
          type: string
        email:
          type: string
          format: email
        password:
          type: string
          format: password

    LoginRequest:
      type: object
      required:
        - username
        - password
      properties:
        username:
          type: string
        password:
          type: string
          format: password

    AuthResponse:
      type: object
      properties:
        token:
          type: string
          description: JWT token
        expiresIn:
          type: integer
          description: Tiempo de expiración en segundos

    ValidationResponse:
      type: object
      properties:
        valid:
          type: boolean
        username:
```

```
          type: string
        exp:
          type: integer
          format: int64

  securitySchemes:
    bearerAuth:
      type: http
      scheme: bearer
      bearerFormat: JWT
```

3.4.- API Gateway

Open API 3.0

```
None
openapi: 3.0.3
info:
  title: API Gateway
  description: |
    Gateway que actúa como punto de entrada único al sistema.
    Realiza validación de JWT y enrutamiento a los microservicios.
  version: 1.0.0

servers:
  - url: http://localhost:8080
    description: Gateway principal

paths:
  /api/submissions:
    post:
      summary: Crear anteproyecto (proxy)
      description: Proxy hacia Submission Service con validación JWT
      operationId: createSubmissionProxy
      tags:
        - Gateway
      security:
        - bearerAuth: []
      requestBody:
        required: true
```

```
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/SubmissionRequest'
      responses:
        '201':
          description: Anteproyecto creado
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/SubmissionResponse'
        '401':
          description: No autorizado (JWT inválido)
        '503':
          description: Servicio no disponible

  /api/notifications:
    post:
      summary: Enviar notificación (proxy)
      description: Proxy hacia Notification Service
      operationId: sendNotificationProxy
      tags:
        - Gateway
      security:
        - bearerAuth: []
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/NotificationRequest'
      responses:
        '200':
          description: Notificación enviada
        '401':
          description: No autorizado

components:
  securitySchemes:
    bearerAuth:
      type: http
      scheme: bearer
      bearerFormat: JWT

  schemas:
    SubmissionRequest:
```

```
      type: object
      # (mismo schema de Submission Service)

    SubmissionResponse:
      type: object
      # (mismo schema de Submission Service)

    NotificationRequest:
      type: object
      # (mismo schema de Notification Service)
```

4.- Script y Colección de Pruebas

4.1.- Flujo Síncrono

Colección Postman (JSON)

```
None
{
  "info": {
    "name": "Microservicios - Flujo Síncrono",
    "schema":
"https://schema.getpostman.com/json/collection/v2.1.0/collection.json"
  },
  "item": [
    {
      "name": "1. Health Check - Submission Service",
      "request": {
        "method": "GET",
        "header": [],
        "url": {
          "raw": "http://localhost:8081/ping",
          "protocol": "http",
          "host": ["localhost"],
          "port": "8081",
          "path": ["ping"]
```

```
        }
      },
      "response": []
    },
    {
      "name": "2. Crear Anteproyecto (Flujo Síncrono)",
      "event": [
        {
          "listen": "test",
          "script": {
            "exec": [
              "pm.test(\"Status code is 201\", function () {",
              "    pm.response.to.have.status(201);",
              "});",
              "pm.test(\"Response has proyectoId\", function () {",
              "    var jsonData = pm.response.json();",
              "    pm.expect(jsonData).to.have.property('proyectoId');",
              "    pm.environment.set('proyectoId', jsonData.proyectoId);",
              "});"
            ]
          }
        }
      ],
      "request": {
        "method": "POST",
        "header": [
          {
            "key": "Content-Type",
            "value": "application/json"
          }
        ],
        "body": {
          "mode": "raw",
          "raw": "{\n  \"titulo\": \"Sistema de Recomendación con ML\",\n
\"resumen\": \"Implementación de algoritmos de recomendación usando
aprendizaje automático para mejorar la experiencia del usuario\",\n
\"autoresEmails\": [\"est1@unicauca.edu.co\", \"est2@unicauca.edu.co\"],\n
\"directorId\": 101,\n  \"estudiante1Id\": 201,\n  \"estudiante2Id\":
202\n}"
        },
        "url": {
          "raw": "http://localhost:8081/api/submissions",
          "protocol": "http",
          "host": ["localhost"],
          "port": "8081",
          "path": ["api", "submissions"]
```

```
        }
      }
    },
    {
      "name": "3. Verificar Notificación REST (manual)",
      "request": {
        "method": "GET",
        "header": [],
        "url": {
          "raw": "http://localhost:8082/health",
          "protocol": "http",
          "host": ["localhost"],
          "port": "8082",
          "path": ["health"]
        }
      }
    }
  ]
}
```

Script PowerShell

Shell

```
# test-sincrono.ps1
# Prueba del flujo síncrono completo

Write-Host "=== PRUEBA FLUJO SÍNCRONO ===" -ForegroundColor Cyan

# 1. Health check
Write-Host "`n1. Verificando Submission Service..." -ForegroundColor Yellow
$ping = Invoke-RestMethod -Uri "http://localhost:8081/ping" -Method GET
Write-Host "   Respuesta: $ping" -ForegroundColor Green

# 2. Crear anteproyecto
Write-Host "`n2. Creando anteproyecto..." -ForegroundColor Yellow
$body = @{
    titulo = "Sistema de Recomendación con ML"
    resumen = "Implementación de algoritmos de recomendación usando
aprendizaje automático"
    autoresEmails = @("est1@unicauca.edu.co", "est2@unicauca.edu.co")
    directorId = 101
    estudiante1Id = 201
    estudiante2Id = 202
```

```
} | ConvertTo-Json

try {
    $response = Invoke-RestMethod -Method POST `
        -Uri "http://localhost:8081/api/submissions" `
        -ContentType "application/json" `
        -Body $body

    Write-Host "   ✓ Proyecto creado con ID: $($response.proyectoId)"
-ForegroundColor Green
    Write-Host "   ✓ Estado: $($response.estado)" -ForegroundColor Green
} catch {
    Write-Host "   ✗ Error: $($_.Exception.Message)" -ForegroundColor Red
}

# 3. Verificar logs en consola de Notification Service
Write-Host "`n3. Verificar logs en la consola de Notification Service"
-ForegroundColor Yellow
Write-Host "   Debe aparecer: 'Notificación enviada' o similar"
-ForegroundColor Cyan

Write-Host "`n=== FIN PRUEBA SÍNCRONA ===" -ForegroundColor Cyan
```

Script Bash

```
None
#!/bin/bash
# test-sincrono.sh

echo "=== PRUEBA FLUJO SÍNCRONO ==="

# 1. Health check
echo -e "\n1. Verificando Submission Service..."
curl -s http://localhost:8081/ping
echo ""

# 2. Crear anteproyecto
echo -e "\n2. Creando anteproyecto..."
curl -X POST http://localhost:8081/api/submissions \
  -H "Content-Type: application/json" \
  -d '{
    "titulo": "Sistema de Recomendación con ML",
```

```
    "resumen": "Implementación de algoritmos de recomendación usando
aprendizaje automático",
    "autoresEmails": ["est1@unicauca.edu.co", "est2@unicauca.edu.co"],
    "directorId": 101,
    "estudiante1Id": 201,
    "estudiante2Id": 202
  }' | jq '.'

echo -e "\n3. Verificar logs en Notification Service"
echo "=== FIN PRUEBA SÍNCRONA ==="
```

4.2. Flujo Asíncrono

Scripts PowerShell

None

```
# test-asincrono.ps1
# Prueba del flujo asíncrono con RabbitMQ

Write-Host "=== PRUEBA FLUJO ASÍNCRONO ===" -ForegroundColor Cyan

# 1. Crear anteproyecto (sin esperar notificación)
Write-Host "`n1. Creando anteproyecto (async)..." -ForegroundColor Yellow
$body = @{
    titulo = "Análisis de Datos con Big Data"
    resumen = "Sistema de análisis de grandes volúmenes de datos usando
tecnologías de Big Data y procesamiento distribuido"
    autoresEmails = @("est3@unicauca.edu.co")
    directorId = 102
    estudiante1Id = 203
} | ConvertTo-Json

try {
    $response = Invoke-RestMethod -Method POST `
        -Uri "http://localhost:8081/api/submissions" `
        -ContentType "application/json" `
        -Body $body

    Write-Host "   ✓ Proyecto aceptado con ID: $($response.proyectoId)"
-ForegroundColor Green
    Write-Host "   ✓ Estado: $($response.estado)" -ForegroundColor Green
```

```
} catch {
    Write-Host "   ✗ Error: $($_.Exception.Message)" -ForegroundColor Red
}

# 2. Verificar RabbitMQ
Write-Host "`n2. Verificar RabbitMQ..." -ForegroundColor Yellow
Write-Host "   Abrir: http://localhost:15672" -ForegroundColor Cyan
Write-Host "   Usuario: guest | Password: guest" -ForegroundColor Cyan
Write-Host "   Verificar que hay 1 mensaje en la cola 'submission.queue'"
-ForegroundColor Cyan

# 3. Esperar procesamiento
Write-Host "`n3. Esperando procesamiento asíncrono..." -ForegroundColor
Yellow
Start-Sleep -Seconds 5

Write-Host "`n4. Verificar logs de Notification Service" -ForegroundColor
Yellow
Write-Host "   Debe aparecer: 'Mensaje consumido de RabbitMQ'"
-ForegroundColor Cyan

Write-Host "`n=== FIN PRUEBA ASÍNCRONA ===" -ForegroundColor Cyan
```

Script Bash/curl

```
None
#!/bin/bash
# test-asincrono.sh

echo "=== PRUEBA FLUJO ASÍNCRONO ==="

# 1. Crear anteproyecto
echo -e "\n1. Creando anteproyecto (async)..."
RESPONSE=$(curl -s -X POST http://localhost:8081/api/submissions \
  -H "Content-Type: application/json" \
  -d '{
    "titulo": "Análisis de Datos con Big Data",
    "resumen": "Sistema de análisis de grandes volúmenes de datos usando
tecnologías de Big Data",
    "autoresEmails": ["est3@unicauca.edu.co"],
    "directorId": 102,
    "estudiante1Id": 203
  }')
```

```
echo $RESPONSE | jq '.'

# 2. Verificar RabbitMQ
echo -e "\n2. Verificar RabbitMQ Management UI:"
echo "   URL: http://localhost:15672"
echo "   Usuario: guest | Password: guest"

# 3. Verificar mensajes en cola
echo -e "\n3. Consultando mensajes en cola..."
curl -s -u guest:guest
http://localhost:15672/api/queues/%2F/submission.queue | jq '{name:.name,
messages:.messages}'

echo -e "\n4. Esperar 5 segundos para procesamiento..."
sleep 5

echo "=== FIN PRUEBA ASÍNCRONA ==="
```

4.3.- Pruebas de Integración

Script de prueba

```
None
# test-integracion-completa.ps1

Write-Host "╔═══════════════════════════════════════════╗" -ForegroundColor
Cyan
Write-Host "║  PRUEBAS DE INTEGRACIÓN COMPLETAS       ║" -ForegroundColor
Cyan
Write-Host "╚═══════════════════════════════════════════╝" -ForegroundColor
Cyan

# Variables de configuración
$submissionUrl = "http://localhost:8081"
$notificationUrl = "http://localhost:8082"
$testResults = @()

function Test-Service {
    param($name, $url)
```

```
    try {
        $response = Invoke-WebRequest -Uri "$url/ping" -TimeoutSec 5
-ErrorAction Stop
        Write-Host "   ✓ $name OK" -ForegroundColor Green
        return $true
    } catch {
        Write-Host "   ✗ $name FALLÓ" -ForegroundColor Red
        return $false
    }
}

# 1. Verificar servicios
Write-Host "`n[1/5] Verificando servicios..." -ForegroundColor Yellow
$submissionOk = Test-Service "Submission Service" $submissionUrl
$notificationOk = Test-Service "Notification Service" $notificationUrl

if (-not ($submissionOk -and $notificationOk)) {
    Write-Host "`n✗ Algunos servicios no están disponibles. Abortando."
-ForegroundColor Red
    exit 1
}

# 2. Prueba flujo síncrono
Write-Host "`n[2/5] Probando flujo SÍNCRONO..." -ForegroundColor Yellow
$syncBody = @{
    titulo = "Proyecto Síncrono Test"
    resumen = "Este es un proyecto de prueba para validar el flujo síncrono
con notificación REST inmediata"
    autoresEmails = @("sync@unicauca.edu.co")
    directorId = 100
    estudiante1Id = 200
} | ConvertTo-Json

try {
    $syncResponse = Invoke-RestMethod -Method POST `
        -Uri "$submissionUrl/api/submissions" `
        -ContentType "application/json" `
        -Body $syncBody

    Write-Host "   ✓ Proyecto creado: ID=$($syncResponse.proyectoId)"
-ForegroundColor Green
    $testResults += @{Test="Flujo Síncrono"; Result="PASS"}
} catch {
    Write-Host "   ✗ Error en flujo síncrono: $($_.Exception.Message)"
-ForegroundColor Red
```

```
    $testResults += @{Test="Flujo Síncrono"; Result="FAIL"}
}

# 3. Prueba flujo asíncrono
Write-Host "`n[3/5] Probando flujo ASÍNCRONO..." -ForegroundColor Yellow
$asyncBody = @{
    titulo = "Proyecto Asíncrono Test"
    resumen = "Este es un proyecto de prueba para validar el flujo
asíncrono con mensajería RabbitMQ"
    autoresEmails = @("async@unicauca.edu.co")
    directorId = 101
    estudiante1Id = 201
} | ConvertTo-Json

try {
    $asyncResponse = Invoke-RestMethod -Method POST `
        -Uri "$submissionUrl/api/submissions" `
        -ContentType "application/json" `
        -Body $asyncBody

    Write-Host "   ✓ Proyecto aceptado: ID=$($asyncResponse.proyectoId)"
-ForegroundColor Green
    Write-Host "   ⏳ Esperando procesamiento asíncrono (5s)..."
-ForegroundColor Cyan
    Start-Sleep -Seconds 5
    $testResults += @{Test="Flujo Asíncrono"; Result="PASS"}
} catch {
    Write-Host "   ✗ Error en flujo asíncrono: $($_.Exception.Message)"
-ForegroundColor Red
    $testResults += @{Test="Flujo Asíncrono"; Result="FAIL"}
}

# 4. Validación de datos inválidos
Write-Host "`n[4/5] Probando validación de datos..." -ForegroundColor
Yellow
$invalidBody = @{
    titulo = "XY"  # Muy corto
    resumen = "Resumen muy corto"
    autoresEmails = @()  # Vacío
    directorId = 100
} | ConvertTo-Json

try {
    Invoke-RestMethod -Method POST `
        -Uri "$submissionUrl/api/submissions" `
        -ContentType "application/json" `
```

```
        -Body $invalidBody `
        -ErrorAction Stop

    Write-Host "   ✗ No se validaron datos inválidos" -ForegroundColor Red
    $testResults += @{Test="Validación Datos"; Result="FAIL"}
} catch {
    if ($_.Exception.Response.StatusCode -eq 400) {
        Write-Host "   ✓ Validación correcta (400 Bad Request)"
-ForegroundColor Green
        $testResults += @{Test="Validación Datos"; Result="PASS"}
    } else {
        Write-Host "   ✗ Error inesperado:
$($_.Exception.Response.StatusCode)" -ForegroundColor Red
        $testResults += @{Test="Validación Datos"; Result="FAIL"}
    }
}

# 5. Verificar RabbitMQ
Write-Host "`n[5/5] Verificando RabbitMQ..." -ForegroundColor Yellow
try {
    $rmqResponse = Invoke-RestMethod -Uri
"http://localhost:15672/api/overview" `
        -Method GET `
        -Credential (New-Object
System.Management.Automation.PSCredential("guest", (ConvertTo-SecureString
"guest" -AsPlainText -Force)))

    Write-Host "   ✓ RabbitMQ disponible:
$($rmqResponse.rabbitmq_version)" -ForegroundColor Green
    $testResults += @{Test="RabbitMQ"; Result="PASS"}
} catch {
    Write-Host "   ✗ RabbitMQ no accesible" -ForegroundColor Red
    $testResults += @{Test="RabbitMQ"; Result="FAIL"}
}

# Resumen
Write-Host "`n╔═══════════════════════════════════════════════╗"
-ForegroundColor Cyan
Write-Host "║         RESUMEN DE PRUEBAS                    ║" -ForegroundColor
Cyan
Write-Host "╚═══════════════════════════════════════════════╝" -ForegroundColor
Cyan

$passed = ($testResults | Where-Object {$_.Result -eq "PASS"}).Count
$failed = ($testResults | Where-Object {$_.Result -eq "FAIL"}).Count
```

```
foreach ($result in $testResults) {
    $color = if ($result.Result -eq "PASS") { "Green" } else { "Red" }
    $symbol = if ($result.Result -eq "PASS") { "✓" } else { "✗" }
    Write-Host "   $symbol $($result.Test): $($result.Result)"
-ForegroundColor $color
}

Write-Host "`nTotal: $passed pasadas, $failed fallidas" -ForegroundColor
$(if ($failed -eq 0) { "Green" } else { "Yellow" })

if ($failed -eq 0) {
    Write-Host "`n TODAS LAS PRUEBAS EXITOSAS " -ForegroundColor Green
} else {
    Write-Host "`n  ALGUNAS PRUEBAS FALLARON " -ForegroundColor Yellow
}
```

5. Checklist de Aceptación

5.1. Criterios Funcionales

Submission Service

#	Criterio	Validación	Prioridad
1	Recibe y valida anteproyectos vía POST	Request con datos válidos retorna 201 Created	**ALTA**
2	Persiste anteproyectos en PostgreSQL	Registro visible en tabla submissions con todos los campos	**ALTA**
3	Publica mensaje en RabbitMQ	Mensaje aparece en cola submission.queue	**ALTA**
4	Llama a Notification Service (REST)	Logs muestran petición HTTP exitosa	**ALTA**

5	Valida datos de entrada	Request inválido retorna 400 Bad Request con mensaje descriptivo	**MEDIA**
6	Health check funcional	GET /ping retorna "pong"	**BAJA**

Tabla 5.1 Criterios funcionales Submission Server

Notification Service

#	Criterio	Validación	Prioridad
7	Consume mensajes de RabbitMQ	Logs muestran "Mensaje consumido de RabbitMQ: {data}"	**ALTA**
8	Recibe notificaciones REST	POST /api/notifications retorna 200 OK	**ALTA**
9	Registra notificaciones en log	Cada notificación genera log con nivel INFO	**ALTA**
10	Implementa retry logic	Si falla, reintenta 3 veces antes de enviar a DLQ	**MEDIA**
11	Health check funcional	GET /health retorna status UP	**BAJA**

Tabla 5.2 Criterios funcionales Notification Server

Identity Service (Mock)

#	Criterio	Validación	Prioridad
12	Registra usuarios	POST /auth/register crea usuario y retorna 201	**MEDIA**
13	Genera JWT válidos	POST /auth/login retorna token JWT bien formado	**ALTA**
14	Valida JWT	GET /auth/validate con token válido retorna 200	**ALTA**

15	Rechaza JWT inválidos	Tokens expirados o malformados retornan 401	**ALTA**

Tabla 5.3 Criterios funcionales Identity Server

API Gateway

#	Criterio	Validación	Prioridad
16	Valida JWT en todas las rutas	Request sin JWT retorna 401 Unauthorized	**ALTA**
17	Enruta correctamente a Submission	Proxy funcional hacia puerto 8081	**ALTA**
18	Enruta correctamente a Notification	Proxy funcional hacia puerto 8082	**ALTA**
19	Maneja errores de servicios caídos	Retorna 503 Service Unavailable si backend no responde	**MEDIA**

Tabla 5.4 Criterios funcionales API Gateway

5.2. Criterios No Funcionales

Rendimiento

#	Criterio	Métrica	Prioridad
20	Tiempo de respuesta síncrona	< 500ms para POST /api/submissions	**MEDIA**
21	Throughput RabbitMQ	Procesa mínimo 100 msg/min	**BAJA**
22	Latencia de persistencia	Insert en PostgreSQL < 100ms	**BAJA**

Tabla 5.5 Criterios NO funcionales Rendimiento

Confiabilidad

#	Criterio	Validación	Prioridad
23	Manejo de fallos DB	Si PostgreSQL cae, retorna 500 con mensaje claro	**ALTA**
24	Manejo de fallos RabbitMQ	Si RabbitMQ no disponible, no bloquea respuesta síncrona	**ALTA**
25	Reintentos automáticos	Notification Service reintenta mensajes fallidos	**MEDIA**

Tabla 5.6 Criterios NO funcionales Confiabilidad

Observabilidad

#	Criterio	Validación	Prioridad
26	Logs estructurados	Todos los logs tienen timestamp, nivel, servicio	**ALTA**
27	Trazabilidad end-to-end	Se puede seguir un request por su ID único	**MEDIA**
28	Métricas RabbitMQ	Visible en UI: mensajes publicados/consumidos	**MEDIA**

Tabla 5.7 Criterios NO funcionales Rendimiento

5.3. Criterios de Infraestructura

Docker

#	Criterio	Validación	Prioridad
29	Docker Compose levanta todos los servicios	docker compose up -d exitoso	**ALTA**
30	Servicios accesibles en puertos correctos	PostgreSQL:5432, RabbitMQ:5672/15672	**ALTA**

31	Persistencia de datos	Datos sobreviven a reinicio de contenedores	**MEDIA**

Tabla 5.8 Criterios Infraestructura Docker

6. Observabilidad y Logs

6.1. Formato de Logs Estandarizado

Todos los servicios deben seguir este formato para garantizar trazabilidad:

```
None
[TIMESTAMP] [NIVEL] [SERVICIO] [TRAZA_ID] - Mensaje detallado
```

CAMPO	REPRESENTACIÓN	EJEMPLO
TIMESTAMP	La fecha y hora exacta en que se generó el evento (con zona horaria)	2025-10-16T01:24:45.123-05:00
NIVEL	El nivel de severidad o importancia del mensaje (INFO, WARN, ERROR, DEBUG)	INFO
SERVICIO	Indica qué microservicio produjo el mensaje	`[submission-service]` o `[notification-service]`
TRAZA_ID	Identificador único que se genera para cada flujo o transacción, sirve para rastrear una misma petición a través de varios servicios	[trace:a3f7bc9d]

Tabla 6.1 Formato de logs estandarizados

Ejemplo Submission Service:

```
None
2025-10-16T01:24:45.123-05:00 INFO  [submission-service] [trace:a3f7bc9d] - Anteproyecto recibido: titulo='Sistema ML', autor='est1@unicauca.edu.co'

2025-10-16T01:24:45.234-05:00 INFO  [submission-service] [trace:a3f7bc9d] - Guardado en PostgreSQL: proyectoId=1

2025-10-16T01:24:45.345-05:00 INFO  [submission-service] [trace:a3f7bc9d] - Mensaje publicado en RabbitMQ: exchange=submission.exchange, routingKey=submission.created

2025-10-16T01:24:45.456-05:00 INFO  [submission-service] [trace:a3f7bc9d] - Notificación REST enviada a http://localhost:8082/api/notifications - Status: 200
```

Ejemplo Notification Service:

```
None
2025-10-16T01:24:50.678-05:00 INFO  [notification-service] [trace:b4e8cd0e] - Mensaje consumido de RabbitMQ: proyectoId=1, titulo='Sistema ML'

2025-10-16T01:24:50.789-05:00 INFO  [notification-service] [trace:b4e8cd0e] - Notificación procesada exitosamente
```

6.2. Puntos de Observación Clave

Submission Service

Punto	Evento	Significado	Log Esperado	Nivel
1	Request recibido	El servicio recibe un POST con los datos del anteproyecto.	"Anteproyecto recibido: {datos}"	El servicio registra que recibió el anteproyecto INFO

2	Validación exitosa	Se verifica que el anteproyecto tenga todos los campos válidos	"Validación OK para proyecto: {titulo}"	Si todo está bien, se deja un log DEBUG para indicar la validación correcta.
3	Inserción en DB	Se guarda el anteproyecto en PostgreSQL.	"Guardado en PostgreSQL: proyectoId={id}"	Log INFO onfirma que el proyecto se insertó correctament e.
4	Publicación RabbitMQ	Se envía un mensaje a la cola de RabbitMQ con info del proyecto.	"Mensaje publicado: exchange={ex}, key={key}"	INFO
5	Llamada REST a Notification	Además de RabbitMQ, se hace una llamada HTTP al Notification Service.	"Notificación REST enviada - Status: {code}"	INFO
6	Error en DB	Falla al guardar el anteproyecto.	"Error al guardar: {exception}"	ERROR con excepción
7	Error en RabbitMQ	Falla al publicar mensaje en la cola.	"Error al publicar mensaje: {exception}"	WARN avisa del problema

Tabla 6.2 Puntos de observación-Submission Server

Notification Service

Pu nto	Evento	Significado	Log Esperado	Nivel

1	Consumo RabbitMQ	Notification recibe un mensaje de la cola (consume).	"Mensaje consumido: proyectoId={id}"	INFO
2	Request REST recibido	Notification también puede recibir notificaciones por API REST.	"POST /api/notifications recibido: {body}"	INFO
3	Procesamiento exitoso	Notificación generada correctamente (por ejemplo, enviada por correo o guardada).	"Notificación procesada exitosamente"	INFO
4	Retry por fallo	Hubo error, se reintenta hasta 3 veces.	"Reintento {n}/3 para mensaje: {id}"	WARN
5	Mensaje enviado a DLQ	Tras 3 fallos, el mensaje va a la *Dead Letter Queue*.	"Mensaje movido a DLQ después de 3 reintentos"	ERROR

Tabla 6.3 Puntos de observación-Notification Server

RabbitMQ

Métrica		Dónde verificar	Valor esperado
Mensajes publicados	Cuántos mensajes envió Submission.	Management UI → Exchanges → submission.exchange	+1 por cada POST
Mensajes en cola	Cuántos mensajes están esperando	Queues → submission.queue	0 Si Notification está funcionando, este valor debería ser 0.

	ser consumidos.		
Mensajes consumidos	Cuántos mensajes procesó Notification.	Queues → submission.queue → Message rates	Rate > 0/s
Conexiones activas	Servicios conectados a RabbitMQ.	Connections	Mínimo 1 (Notification Service)

Tabla 6.4 Puntos de observación-RabbitMQ

6.3. Evidencia del Flujo de Mensajes

Checklist de Verificación Visual

Flujo Síncrono

1. **Terminal Submission Service**: Ver log de request recibido
2. **Terminal Submission Service**: Ver log de INSERT en PostgreSQL
3. **Terminal Submission Service**: Ver log de llamada REST a Notification
4. **Terminal Notification Service**: Ver log de POST /api/notifications recibido
5. **Postman/curl**: Recibir respuesta 201 Created

Flujo Asíncrono

1. **Terminal Submission Service**: Ver log de publicación en RabbitMQ
2. **RabbitMQ UI** (http://localhost:15672):
   - Ver +1 mensaje en submission.exchange
   - Ver mensaje llegar a submission.queue
3. **Terminal Notification Service**: Ver log de consumo desde RabbitMQ (puede tardar algunos segundos)
4. **RabbitMQ UI**: Ver que mensaje desaparece de submission.queue (consumido)
5. **Postman/curl**: Recibir respuesta 202 Accepted inmediatamente

---

6.4. Configuración de Logs en Spring Boot

application.yml (Submission Service)

yaml

```
None
logging:
  level:
    root: INFO
    com.unicauca.submission: DEBUG
    org.springframework.amqp: DEBUG
    org.hibernate.SQL: DEBUG
  pattern:
    console: "%d{yyyy-MM-dd'T'HH:mm:ss.SSSXXX} %-5level [%thread] [submission-service] [trace:%X{traceId}] - %msg%n"
```

application.yml (Notification Service)

yaml

```
None
logging:
  level:
    root: INFO
    com.unicauca.notification: DEBUG
    org.springframework.amqp: DEBUG
  pattern:
    console: "%d{yyyy-MM-dd'T'HH:mm:ss.SSSXXX} %-5level [%thread] [notification-service] [trace:%X{traceId}] - %msg%n"
```

---

7. Guía de Ejecución

7.1. Requisitos Previos

- **Java 21** (OpenJDK o Adoptium)
- **Docker Desktop** instalado y corriendo
- **Maven 3.8+** o usar los wrappers incluidos (mvnw)
- **PowerShell 7+** o bash (para scripts de prueba)

- **Postman** (opcional, para colección de pruebas)

7.2. Paso a Paso

Clonar repositorios (si aplica)

bash

```
Shell
git clone <repo-submission-service>
git clone <repo-notification-service>
```

Levantar infraestructura

powershell

```
None
# Navegar al directorio con docker-compose.yml
cd <ruta-proyecto>

# Levantar PostgreSQL y RabbitMQ
docker compose up -d

# Verificar que están corriendo
docker ps
```

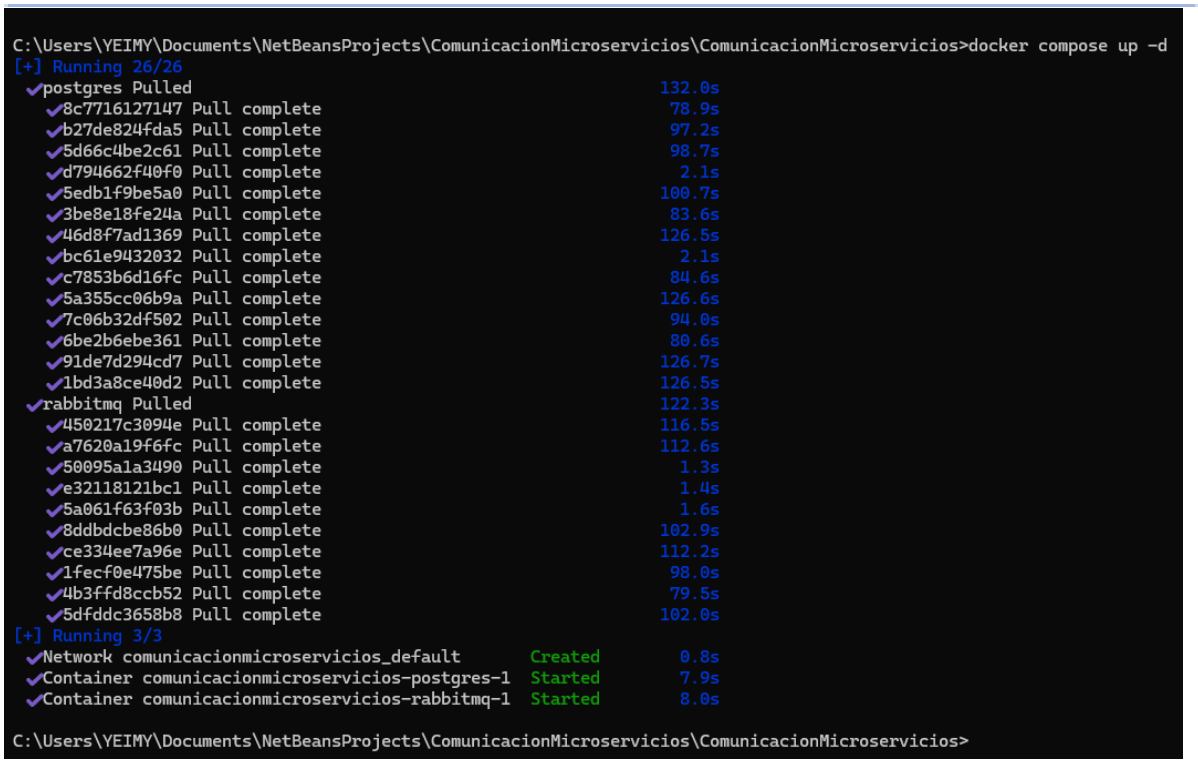


Figura 7.1 LevantarPostgreQL y RabbitMQ

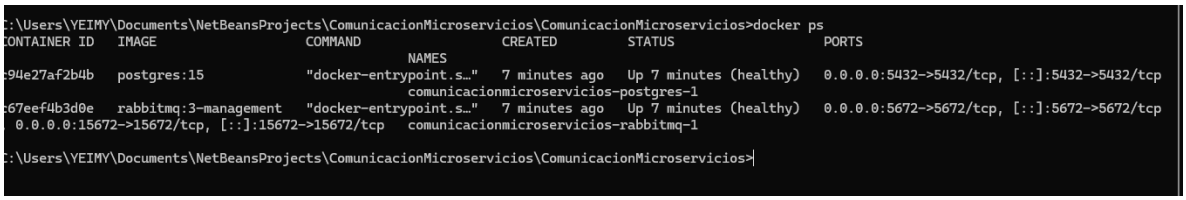


Figura 7.2 Verificar si está corriendo

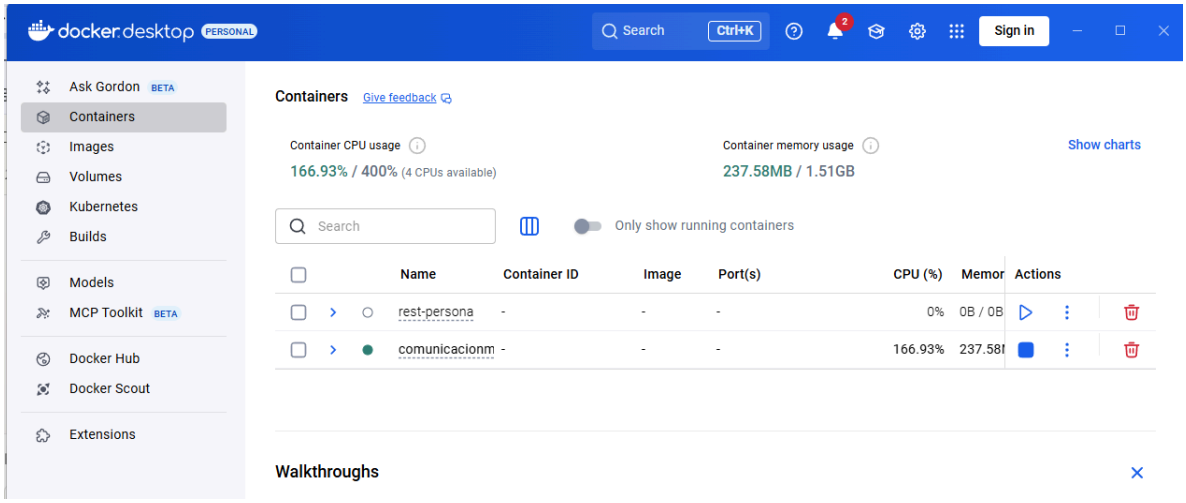


Figura 7.3 Comunicación en Docker

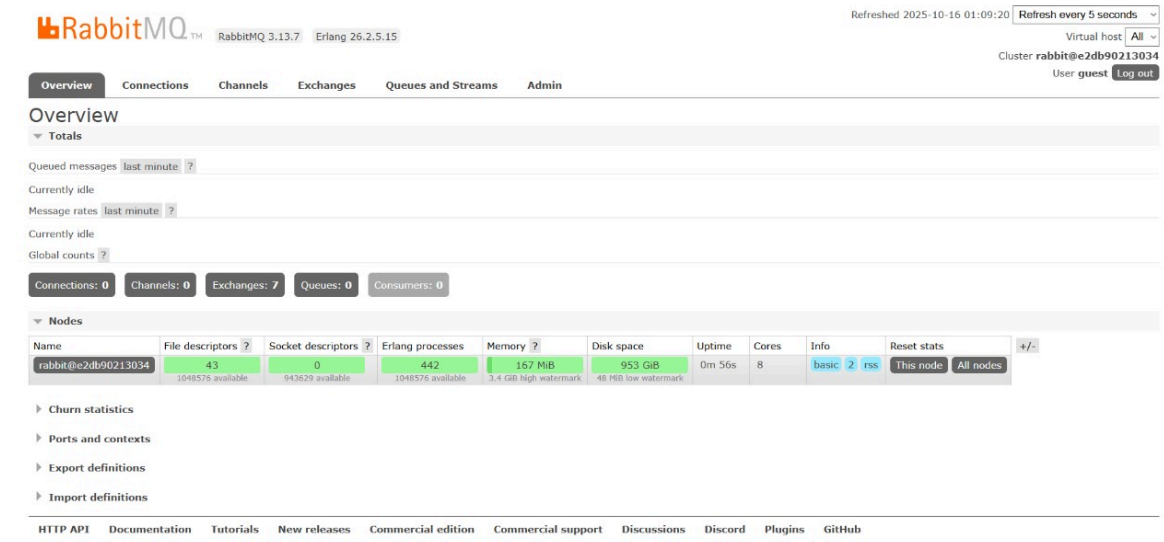


Figura 7.4 Comunicación RabbitMQ Management Dasheboard

Compilar servicios

powershell

```
None
# Submission Service
cd submission-service
mvn clean package -DskipTests

# Notification Service
cd ../notification-service
mvn clean package -DskipTests
```

Ejecutar servicios

Terminal 1 - Submission Service:

powershell

```
None
cd submission-service
mvn spring-boot:run
```

Terminal 2 - Notification Service:

powershell

```
None
cd notification-service
mvn spring-boot:run
```

Esperar a ver:

```
None
Tomcat started on port(s): 8081 (http)
Tomcat started on port(s): 8082 (http)
```

Verificar servicios

powershell

```
None
# Health checks
curl http://localhost:8081/ping   # Debe responder "pong"
curl http://localhost:8082/health  # Debe responder {"status":"UP"}
```

Ejecutar pruebas

powershell

```
None
# Flujo síncrono
.\scripts\test-sincrono.ps1


# Flujo asíncrono
.\scripts\test-asincrono.ps1
```

```
# Suite completa
.\scripts\test-integracion-completa.ps1
```

7.3. Verificación de Infraestructura

PostgreSQL

powershell

```
None
# Conectar a la base de datos
docker exec -it <postgres-container-id> psql -U postgres -d submissiondb


# Ver anteproyectos creados
SELECT * FROM submissions;
```

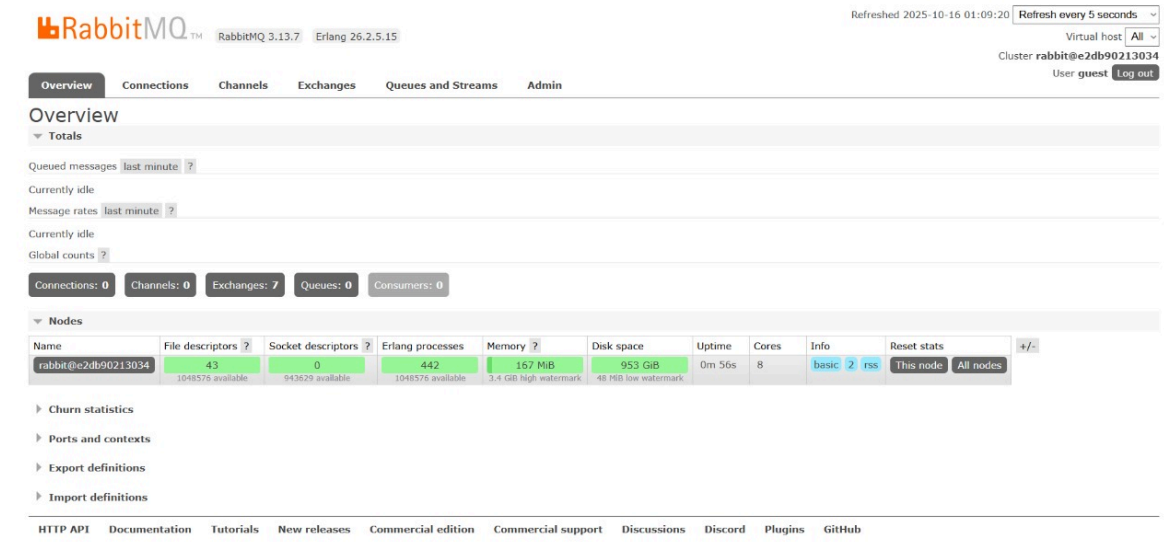
RabbitMQ

Abrir navegador en: **http://localhost:15672**

- Usuario: guest
- Password: guest

Verificar:

- **Exchanges**: Debe existir submission.exchange
- **Queues**: Debe existir submission.queue
- **Bindings**: Exchange vinculado a queue con routing key submission.created



Bibliografía

[1] Spring Boot Documentation, "Building REST Services with Spring", Spring.io, 2025. [En línea]. Disponible: https://spring.io/guides/tutorials/rest/

[2] RabbitMQ, "RabbitMQ Tutorial - Work Queues", RabbitMQ.com, 2025. [En línea]. Disponible: https://www.rabbitmq.com/tutorials/tutorial-two-java.html

[3] OpenAPI Initiative, "OpenAPI Specification v3.0.3", OpenAPIs.org, 2025. [En línea]. Disponible: https://spec.openapis.org/oas/v3.0.3

[4] C. Richardson, "Pattern: API Gateway / Backends for Frontends", Microservices.io, 2025. [En línea]. Disponible: https://microservices.io/patterns/apigateway.html