



Manual Técnico: Microservicio de Correo con gRPC

Ileana Garza
Agosto 2025

TABLA DE CONTENIDO

1. Introducción	2
2. Arquitectura General	3
2.1 Arquitectura Detallada: Microservicio de Correo.....	4
2.2 Entorno de Ejecución (Contenedor Docker)	6
2.3 Tecnologías Principales.....	7
3. Requisitos Previos	7
Software Necesario.	7
Accesos.	7
4. Configuración del Entorno.	8
Variables de Entorno / Archivo appsettings.json.....	8
5. Guía de Instalación.....	9
Instalación Local.....	9
Ejecución y Despliegue.....	10
Ejecutar con Docker:.....	11
7. Funcionamiento Técnico.	12
Principales Componentes (API gRPC).....	12
Flujos de Trabajo.	12
Configuración de Confianza de Certificados (mTLS).....	12
8. Pruebas.....	14
9. Mantenimiento y Escalabilidad.	17
Errores Comunes.	17
Sugerencias para Escalabilidad.....	18
10. Anexos y Recursos.....	19
10.1 Glosario de Términos.....	19
10.2 Decisiones de Arquitectura y Depuración: Conexión a EWS desde Linux.....	20
10.3 Generación de Certificado de Cliente para Conexión gRPC (mTLS).....	21
10.4 Enlaces y Recursos Externos.	23
10.5 Historial de Versiones (Changelog).	24

1. INTRODUCCIÓN

Descripción del Proyecto:

Este proyecto es un microservicio backend desarrollado en C# y .NET 8, diseñado para centralizar las operaciones de correo electrónico. Proporciona una API gRPC con funcionalidades para enviar correos y leer bandejas de entrada, con el fin de ser consumido por otros microservicios dentro de la organización.

Audiencia Objetivo:

Este documento está dirigido a desarrolladores de software, arquitectos y personal de operaciones (DevOps) responsables del desarrollo, despliegue y mantenimiento del microservicio.

Estado del Proyecto:

En Desarrollo / Fase de Pruebas.

Responsable de Mantenimiento:

Ileana Garza

2. ARQUITECTURA GENERAL

Diagrama de arquitectura.

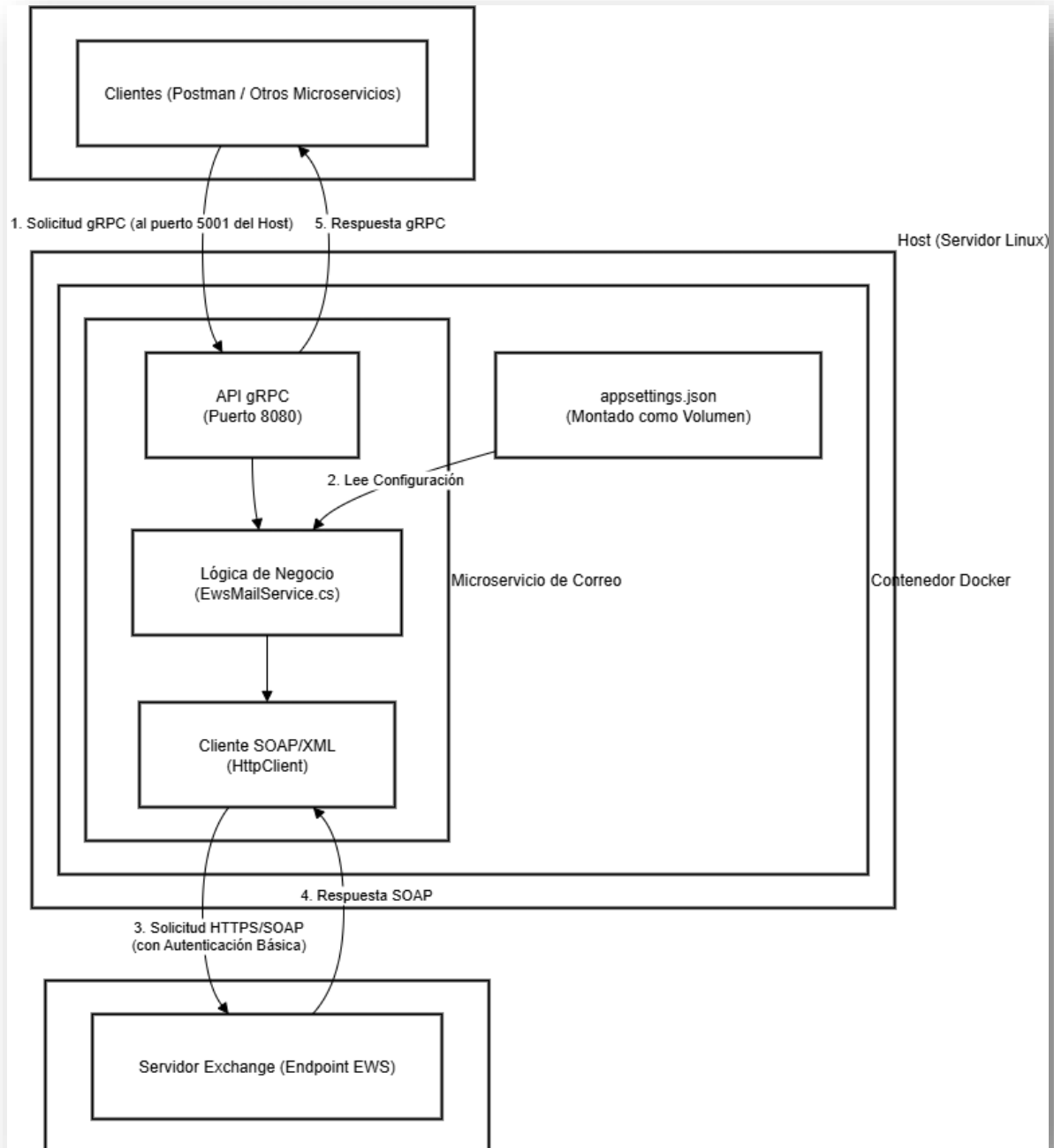


Ilustración 1. Arquitectura del microservicio.

2.1 Arquitectura Detallada: Microservicio de Correo.

La arquitectura del Microservicio de Correo es un diseño de N-capas contenido dentro de un entorno Docker, diseñado para desacoplar las operaciones de correo electrónico del resto de las aplicaciones de la empresa.

2.1.1 Clientes del Servicio (Consumers)

- **Descripción:** Son las aplicaciones que necesitan enviar o leer correos. En el ecosistema de la empresa, son otros microservicios. Para fines de prueba y desarrollo, el cliente es una herramienta como **Postman**.
- **Interacción:**
 - Se comunican con el microservicio utilizando el protocolo **gRPC**.
 - Envían solicitudes (como `SendMailRequest`) que están estrictamente definidas por el contrato en el archivo `email.proto`.
 - **Flecha en el diagrama:** Desde "Clientes" hacia "Microservicio de Correo" con la etiqueta "**Solicitud gRPC (HTTP/2)**".

2.1.2. El Microservicio de Correo (El Núcleo)

Este es el componente central, que se ejecuta dentro de un contenedor Docker. Internamente, se divide en las siguientes capas:

- **Capa de API (gRPC):**
 - **Tecnología:** Servidor gRPC de ASP.NET Core.
 - **Función:** Es la puerta de entrada. Escucha en un puerto interno (ej. 8080) las solicitudes gRPC de los clientes. Valida que las solicitudes cumplan con el contrato del `.proto` y las pasa a la capa de lógica de negocio.

- **Capa de Lógica de Negocio (EwsMailService.cs):**
 - **Tecnología:** Código C# personalizado.
 - **Función:** Es el cerebro del servicio. Contiene la implementación de los métodos `SendMail` y `ReadInbox`. Su responsabilidad es traducir la solicitud gRPC en una o más operaciones de EWS. Por ejemplo, para un `ReadInboxRequest`, construye la consulta SOAP/XML correspondiente.
- **Capa de Acceso a Datos (Cliente SOAP/XML):**
 - **Tecnología:** HttpClient nativo de .NET.
 - **Función:** Es el componente que se comunica con el exterior.
 1. Recibe el XML de la capa de lógica.
 2. Crea una solicitud `HTTP POST` hacia el servidor Exchange.
 3. Añade la cabecera de Autenticación Básica (`Authorization: Basic ...`).
 4. Envía la solicitud SOAP y recibe la respuesta XML del servidor.
 - **Flecha en el diagrama:** Desde "Microservicio de Correo" hacia "Servidor de Exchange" con la etiqueta "**Llamada EWS (HTTPS + SOAP/XML)**".
- **Capa de Configuración:**
 - **Tecnología:** Archivo `appsettings.json`.
 - **Función:** Almacena la configuración externa necesaria para operar, como la URL del endpoint de EWS (`EwsUrl`) y las credenciales de la cuenta de correo (`EmailAddress`, `Password`). Este archivo se monta en el contenedor como un volumen para no tener que reconstruir la imagen si las credenciales cambian.

2.1.3. Servidor de Exchange (Dependencia Externa)

- **Descripción:** Es el servidor de correo corporativo de Microsoft Exchange.
- **Interacción:**
 - Expone un endpoint de Exchange Web Services (EWS), que es una API basada en SOAP/XML.
 - Recibe las solicitudes del microservicio, valida las credenciales de Autenticación Básica, ejecuta la operación solicitada (enviar, buscar correos) y devuelve el resultado en formato SOAP/XML.
 - **Flecha en el diagrama:** Desde "Servidor de Exchange" hacia "Microservicio de Correo" con la etiqueta "Respuesta EWS (SOAP/XML)".

2.2 Entorno de Ejecución (Contenedor Docker)

- **Descripción:** Es la "caja" que empaqueta y aísla el microservicio.
- **Tecnología:** Docker.
- **Sistema Operativo Base: Linux** (ej. `mcr.microsoft.com/dotnet/aspnet:8.0`).
- **Función:**
 - Proporciona un entorno de ejecución consistente y portátil para el microservicio.
 - **Networking:** Mapea un puerto del servidor host (ej. 5001) al puerto interno del contenedor (8080) para permitir que los clientes se conecten.
 - **Volúmenes:** Monta el `appsettings.json` desde el host para permitir la configuración en tiempo de ejecución.

2.3 Tecnologías Principales.

- **Lenguaje:** C#
- **Framework:** .NET 8 (ASP.NET Core para gRPC)
- **Protocolo de Comunicación:** gRPC
- **Librería de Correo:** Se realizan llamadas SOAP/XML directas a EWS mediante HttpClient de .NET
- **Contenerización:** Docker (Linux Containers)

3. REQUISITOS PREVIOS

Software Necesario.

- .NET 8 SDK
- Docker Desktop (para ejecución y pruebas locales en contenedores)
- Un cliente gRPC como Postman para realizar pruebas manuales.
- Una cuenta de Postman (la versión de escritorio es recomendada).
- Un certificado de cliente válido, firmado por la Autoridad Certificadora (CA) del servicio. Ver Anexo para instrucciones de generación.

Accesos.

- Credenciales de una cuenta de servicio o de usuario para el servidor de Exchange con permisos para enviar y leer correos.
- Acceso al Workspace de Postman del proyecto para ejecutar la suite de pruebas de integración. (Ver sección 8 Pruebas).

4. CONFIGURACIÓN DEL ENTORNO.

Importante: Manejo de Secretos y Entornos.

El archivo `appsettings.json` contiene la estructura de configuración base para todos los entornos. Es seguro incluirlo en el control de código fuente (Git) siempre y cuando los valores sensibles como `"Password"` se dejen vacíos o con valores de ejemplo.

- **Para Desarrollo Local:** Los valores sensibles (como la contraseña) deben ser gestionados a través del Administrador de Secretos de .NET (`secrets.json`) para evitar exponerlos en el repositorio.
- **Para Pruebas y Producción (Kubernetes):** Los valores sensibles deben ser inyectados a través de Kubernetes Secrets. Estos valores se montan como variables de entorno en el contenedor y sobrescriben automáticamente cualquier valor presente en `appsettings.json`.

Variables de Entorno / Archivo `appsettings.json`.

El servicio se configura a través de un archivo `appsettings.json`. La estructura necesaria es la siguiente:

JSON

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Debug",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "MailSettings": {
    "EmailAddress": "cuenta-de-servicio@triara.com",
    "Password": "la-contraseña-de-la-cuenta",
    "EwsUrl": "https://outlook.triara.com/ews/exchange.asmx"
  }
}
```

5. GUÍA DE INSTALACIÓN.

Instalación Local.

1. Clonar el repositorio:

```
Bash  
git clone https://gitlab.triara.com/triggi/emailmicroservice.git
```

2. Navegar al directorio:

```
Bash  
cd EmailMicroservice
```

3. Restaurar dependencias:

```
Bash  
dotnet restore
```

4. Compilar el proyecto:

```
Bash  
dotnet build
```

EJECUCIÓN Y DESPLIEGUE.

Comportamiento de Arranque según el Entorno El microservicio implementa una lógica de arranque híbrida en Program.cs para facilitar tanto la depuración local como el despliegue en contenedores.

- **En Desarrollo (desde Visual Studio):** El código detecta el entorno de desarrollo (`builder.Environment.IsDevelopment()`) y **no aplica** ninguna configuración forzada de puertos. En su lugar, respeta totalmente los perfiles definidos en `Properties/launchSettings.json` (ej. puerto 5194), permitiendo el uso de las herramientas de depuración nativas del IDE.
- **En Producción (desde Docker):** Al detectar que no es un entorno de desarrollo, la aplicación activa una configuración explícita de Kestrel: `options.ListenAnyIP(8080)`. Esto fuerza al servicio a escuchar en el puerto `8080` en todas las interfaces de red y utilizando el protocolo HTTP/2 sin encriptación (h2c), lo cual es el estándar requerido para la comunicación interna dentro de la red de Docker y su mapeo de puertos.

Ejecutar con Docker:

El proyecto utiliza un `Dockerfile` multi-etapa basado en la imagen `aspnet:8.0` de Linux. No requiere librerías de sistema adicionales, lo que resulta en una imagen ligera.

1. Construir la imagen:

```
Bash
docker build -t email-microservice .
```

2. Ejecutar el contenedor:

```
Bash
docker run -d -p 5001:8080 -v
/ruta/local/appsettings.json:/app/appsettings.json --name email-service
email-microservice
```

Pipeline de CI/CD:

Actualmente, el proyecto no cuenta con un pipeline de integración y despliegue continuo (CI/CD) automatizado. En caso de implementarse, la herramienta sugerida para la automatización de la compilación, pruebas y despliegue de la imagen de Docker es **Jenkins**.

7. FUNCIONAMIENTO TÉCNICO.

Principales Componentes (API gRPC).

El servicio expone dos métodos principales definidos en `Protos/email.proto`:

- `SendMail`: Recibe un `SendMailRequest` con los detalles del correo (remitente, destinatarios, asunto, cuerpo, adjuntos) y lo envía a través del servidor Exchange.
- `ReadInbox`: Recibe un `ReadInboxRequest` con parámetros de filtrado. La respuesta siempre incluye los datos principales del correo (ID, Asunto, Remitente, Cuerpo). Adicionalmente, contiene un campo booleano opcional `include_attachments`. Si este campo es `true`, el servicio realizará llamadas adicionales para obtener los detalles completos de los archivos adjuntos. Si es `false` (o no se especifica), la respuesta será más rápida al omitir los adjuntos.

Flujos de Trabajo.

El servicio se comunica con el servidor Exchange enviando solicitudes SOAP/XML directamente a través de `HttpClient`. La autenticación se realiza mediante el esquema de Autenticación Básica, enviando las credenciales codificadas en Base64 en la cabecera de cada petición. Este enfoque evita las complejidades de la autenticación integrada de Windows (NTLM/Kerberos) y garantiza la compatibilidad con el entorno de contenedores de Linux. Se requiere la URL explícita del endpoint de EWS en la configuración.

Configuración de Confianza de Certificados (mTLS).

Para permitir el uso de una Autoridad Certificadora (CA) interna y autofirmada para la comunicación segura entre servicios, la lógica de validación de certificados de cliente en `Program.cs` incluye dos configuraciones específicas que relajan las políticas de seguridad por defecto. Es crucial que el equipo entienda su propósito y sus implicaciones.

1. `VerificationFlags = AllowUnknownCertificateAuthority`

- **Propósito:** Esta línea es **necesaria** para que el sistema acepte certificados firmados por nuestra CA (`ca.crt`), ya que no es una autoridad pública reconocida por el sistema operativo.
- **Mitigación:** El riesgo potencial de esta configuración se mitiga al validar explícitamente la cadena de confianza contra nuestro propio archivo `ca.crt` dentro de la lógica de validación.

2. **RevocationMode = NoCheck**

- **Propósito:** Esta línea le indica al servicio que **no verifique** si un certificado de cliente ha sido revocado (cancelado) por la CA antes de su fecha de expiración. En un entorno de desarrollo, esto simplifica enormemente la gestión de certificados.
- **Alerta de Seguridad:** Esta configuración introduce una vulnerabilidad en un entorno de producción de alta seguridad. **Si la clave privada de un microservicio cliente es comprometida y su certificado es revocado, nuestro servicio lo seguiría aceptando como válido hasta que expire.** Para una seguridad máxima en el futuro, se debería implementar una infraestructura de Lista de Revocación de Certificados (CRL) y cambiar este modo a `Online` u `Offline`. Para el estado actual del proyecto, `NoCheck` es una configuración aceptada y práctica.

8. PRUEBAS

Cómo Ejecutar Pruebas (Workspace de Postman):

Las pruebas de integración y API del servicio se gestionan a través de un Workspace compartido en Postman, que contiene la colección de solicitudes y los entornos predefinidos. Dependiendo de la configuración del servidor, existen dos maneras de conectarse.

Acceso al Workspace:

Para acceder a la suite de pruebas, utiliza el siguiente enlace:

```
https://dao111.postman.co/workspace/Microservicio-Correo~cca1fab1-5653-4d12-b146-3e8b08ee9fc3/collection/68af860fbc88542b2e04c05e?action=share&creator=34379515
```

Pasos para la Configuración:

1. **Accede al Workspace:** Abre el enlace anterior. Si tienes la aplicación de Postman instalada, te sugerirá abrirlo directamente ahí.
2. **Crea un "Fork" de la Colección:** Para poder ejecutar y modificar las pruebas sin alterar la versión principal, busca la colección "Microservicio de Correo", haz clic en los tres puntos (...) y selecciona "Create a fork". Guárdala en tu propio workspace personal.
3. **Crea un "Fork" del Entorno:** Repite el paso anterior para el entorno (ej. "EmailMicroservice - Docker") para tener tu propia copia con las variables.
4. **Selecciona tu Entorno:** En la esquina superior derecha de Postman, asegúrate de seleccionar el entorno del que acabas de hacer "fork".
5. **Ejecuta el Servicio:** Ten el microservicio corriendo, ya sea localmente desde Visual Studio o en un contenedor de Docker.
6. **Envía Solicitudes:** Ya puedes ejecutar las pruebas desde tu copia ("fork") de la colección.

Opción 1: Conexión Insegura (para Desarrollo Rápido).

Ideal para pruebas rápidas de la lógica de negocio sin configurar la seguridad de los certificados.

- **Requisito:** El servicio debe estar ejecutándose y exponiendo un puerto HTTP inseguro (ej. `http://localhost:5000` o `http://localhost:5001` si usas Docker).
- **URL en Postman:** Introduce la dirección del servidor, por ejemplo, `localhost:5001`.
- **Configuración de TLS:** Asegúrate de que el candado junto a la URL esté abierto (TLS desactivado).
- Ya puedes invocar los métodos del servicio.

Opción 2: Conexión Segura con Certificados (mTLS)

Este método es necesario para probar el flujo de autenticación de cliente, simulando un entorno de producción.

1. Requisitos Previos:

- Haber generado tus certificados de cliente (`client.crt`, `client.key`).
- Tener el certificado de la Autoridad Certificadora (`ca.crt`) del servicio.
- (Ver [Anexo 10.3](#) para instrucciones de generación).

2. Configurar Certificados en Postman:

- Ve a `Settings` > `Certificates`.
- Haz clic en `Add Certificate`.
- Rellena los campos:
- **Host:** `localhost`
- **Port:** `5001` (o el puerto seguro que hayas configurado)
- **CRT file:** Selecciona tu archivo `client.crt`.
- **KEY file:** Selecciona tu archivo `client.key`.

3. Configurar la CA de Confianza:

- En la misma pestaña de `Certificates`, activa la opción `CA Certificates`.
- Selecciona el archivo `ca.crt` del servidor. Esto le dice a Postman que confíe en el certificado que tu servicio le presente.

4. Realizar la Solicitud:

- **URL en Postman:** Introduce la dirección segura, ej. `localhost:5001`.
- **Configuración de TLS:** Asegúrate de que el candado junto a la URL esté cerrado (TLS habilitado).
- Ya puedes invocar los métodos del servicio a través de un canal seguro y autenticado

Listado de métodos probados.

Cada uno de los métodos siguientes se adjuntó un `response` de ejemplo dentro de la colección.

- SendMail – Correo simple Éxito
- SendMail – Con Adjunto Éxito
- SendMail – Sin destinatario *Error*
 1. Retorna Error en servicio como `3 INVALID_ARGUMENT`
 2. Retorna Error interno (logs): `Error status code 'InvalidArgument' with detail 'La lista de destinatarios 'to' no puede estar vacía.' raised.`
- ReadInbox – Sin filtros
- ReadInbox – Con filtros No Leídos Adjuntos
- ReadInbox – Con rango de fechas

9. MANTENIMIENTO Y ESCALABILIDAD.

Log (Informativo): `HttpMessageHandler expired...` y `Starting HttpMessageHandler cleanup cycle.`

- **Explicación:** Estos mensajes de nivel de depuración (`debug`) son parte del funcionamiento normal del `HttpClientFactory` de .NET. Indican que el sistema está gestionando su piscina de conexiones de red para optimizar el rendimiento y reciclar conexiones. No representan un error y pueden ser ignorados de forma segura. Solo aparecen cuando el nivel de log está configurado en Debug.

Errores Comunes.

- **Error:** Falla en la conexión con `ECONNREFUSED`.
Solución: Verificar que el puerto en la URL de Postman coincide con el puerto en el que el servicio está escuchando (ya sea el de Docker o el de `launchSettings.json`).
- **Error:** Falla la conexión a Exchange.
Solución: El problema más común en este entorno serán credenciales incorrectas (EmailAddress o Password) en el archivo `appsettings.json`. La autenticación a nivel de protocolo ya no debería ser una fuente de errores.
- **Error:** El servidor devuelve `401 Unauthorized` o `403 Forbidden`.
Solución: Verifica que la `EmailAddress` y `Password` en `appsettings.json` sean correctas. Confirma con TI que la Autenticación Básica sigue habilitada para la cuenta.
- **Error:** El servidor devuelve `500 Internal Server Error` o errores de XML.
Solución: Revisa el formato del XML en las solicitudes SOAP dentro de la clase `EwsMailService.cs`. Un pequeño error de sintaxis puede causar el fallo.
- **Error:** Falla en la conexión con error de TLS.
Solución: El código en `Program.cs` para aceptar certificados internos (ServicePointManager...) debe permanecer.

- **Error:** El correo se envía correctamente, pero **sin archivos adjuntos**, o la solicitud falla cuando se incluyen múltiples destinatarios.
Causa: Violación del esquema XML estricto de EWS. El servidor Exchange requiere un orden específico de los elementos y el uso correcto de espacios de nombres (namespaces).
Solución: Asegurar en `EwsMailService.cs` que la construcción del XML siga estrictamente este orden: **Subject -> Body -> Attachments -> ToRecipients -> CcRecipients**. Además, los adjuntos deben declararse bajo el espacio de nombres de tipos (`<t:Attachments>`) y no el de mensajes.
- **Error:** Excepción `Name or service not known` en los logs.
Causa: Fallo en la resolución de DNS. El servicio está intentando conectarse a una URL de EWS fija que no existe o es inaccesible desde el contenedor.
Solución: Verificar que la URL obtenida por `Autodiscover` sea correcta. Si se usa una URL fija en `appsettings.json`, confirmar que el contenedor tiene acceso a la red corporativa y puede resolver el nombre del servidor.

Sugerencias para Escalabilidad.

Al ser un servicio en contenedor, puede ser escalado horizontalmente aumentando el número de réplicas en un orquestador como Kubernetes o Docker Swarm.

10. ANEXOS Y RECURSOS.

10.1 Glosario de Términos.

- **EWS (Exchange Web Services):** La API basada en SOAP/XML que expone el servidor de Microsoft Exchange para interactuar con los datos de los buzones.
- **gRPC:** El framework de comunicación de alto rendimiento utilizado por el microservicio para exponer su API.
- **Autenticación Básica:** El esquema de autenticación simple (usuario y contraseña codificados) que utiliza el servicio para conectarse a EWS.
- **SOAP:** El protocolo de mensajería basado en XML que utiliza EWS.
- **Contenedor Docker:** El paquete de software que aísla la aplicación y sus dependencias para asegurar que se ejecute de manera consistente en cualquier entorno.

10.2 Decisiones de Arquitectura y Depuración: Conexión a EWS desde Linux.

Resumen del Problema: Durante el desarrollo, se encontró un bloqueo persistente al intentar conectar con el servidor Exchange desde un contenedor de Docker en Linux, resultando en un error `401 Unauthorized`. Sin embargo, la conexión desde un entorno de desarrollo en Windows funcionaba correctamente.

Alternativas Exploradas:

- **Librería EWS con Autenticación Integrada:** Fue el primer enfoque. Falló debido a una incompatibilidad fundamental entre la autenticación **NTLM/Kerberos** del servidor y el **entorno no unido a dominio de Linux**, a pesar de instalar las librerías de soporte (`libgssapi-krb5-2`, `gss-ntlmssp`).
- **Librería EWS Forzando Autenticación Básica:** La librería demostró seguir intentando una negociación de autenticación integrada incluso cuando se le indicaba usar solo Autenticación Básica, haciéndola inviable.
- **Contenedores de Windows:** Se consideró como una solución viable, ya que el entorno Windows sí maneja la autenticación de forma nativa. Se descartó para cumplir el requisito de desplegar en servidores Linux.

Solución Final Adoptada: Se eliminó por completo la dependencia de la librería `Microsoft.Exchange.WebServices.NETStandard`. La solución final consiste en usar el `HttpClient` nativo de .NET para construir y enviar solicitudes **SOAP/XML** directamente al endpoint de EWS, utilizando **Autenticación Básica** de forma explícita.

- **Justificación:** Este enfoque fue elegido porque:
- **Resuelve el problema de raíz:** Evita por completo la frágil negociación de NTLM/Kerberos en Linux.
- **Simplifica el entorno:** El `Dockerfile` es más ligero y no depende de paquetes de sistema operativo.
- **Es compatible:** Se alinea con la configuración del servidor, que tiene la Autenticación Básica habilitada.
- **Control de Serialización:** Las librerías estándar de EWS presentaban problemas al serializar adjuntos binarios y listas complejas de destinatarios en entornos .NET 8. El uso de `HttpClient` permite construir manualmente el cuerpo `SOAP/XML`, otorgando control total sobre la estructura, el orden de los elementos y los espacios de nombres, lo cual fue crítico para cumplir con las validaciones de esquema de Exchange 2013.

10.3 Generación de Certificado de Cliente para Conexión gRPC (mTLS)

Para interactuar con este microservicio, es obligatorio que el cliente se autentique usando un certificado TLS (mTLS). Cada cliente debe generar su propio par de claves y solicitar que sea firmado por la Autoridad Certificadora (CA) del Microservicio de Correo.

Requisitos Previos:

- Tener OpenSSL instalado.

Nota de Seguridad: En un entorno de producción, el cliente solo generaría una Solicitud de Firma (`.csr`) y la enviaría al equipo del servicio para ser firmada, sin tener acceso a la clave de la CA (`ca.key`). Para entornos de desarrollo, este método es más rápido.

Paso 1: Crear la Autoridad Certificadora (CA) La CA es la entidad de máxima confianza. Su certificado (`ca.crt`) se distribuirá a todos los servicios para que puedan validar las firmas.

1. Generar la clave privada de la CA:

```
openssl genrsa -out ca.key 4096
```

2. Generar el certificado raíz de la CA (Autofirmado):

```
openssl req -x509 -new -nodes -key ca.key -sha256 -days 3650 -out ca.crt -subj  
"/CN=MicroservicioCorreo-CA"
```

Paso 2: Generar el Certificado del Servidor (Para el Microservicio) Este certificado identifica al microservicio ante sus clientes.

1. Generar clave privada del servidor:

```
openssl genrsa -out server.key 2048
```

2. Crear solicitud de firma (CSR): *Nota: En CN (Common Name), usa localhost para pruebas locales o el nombre DNS del servicio en producción.*

```
openssl req -new -key server.key -out server.csr -subj "/CN=localhost"
```

3. Firmar el certificado con la CA:

```
openssl x509 -req -in server.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out server.crt -days 365 -sha256
```

Paso 3: Generar Certificados de Cliente (Para Consumidores) Cada microservicio o herramienta (como Postman) que quiera consumir el servicio de correo debe tener su propio certificado firmado por la CA.

1. Generar clave privada del cliente:

```
openssl genrsa -out client.key 2048
```

2. Crear solicitud de firma (CSR):

```
openssl req -new -key client.key -out client.csr -subj "/CN=Cliente-Postman"
```

3. Firmar el certificado con la CA:

```
openssl x509 -req -in client.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out client.crt -days 365 -sha256
```

Archivos Resultantes y Ubicación:

- **En el Servidor (Microservicio de Correo):** Debe tener `server.crt`, `server.key` y `ca.crt` (para validar clientes).
- **En el Cliente (ej. Postman):** Debe tener `client.crt`, `client.key` y `ca.crt` (para validar al servidor).

10.4 Enlaces y Recursos Externos.

Docker. (2025, 10 de junio). *Dockerize a .NET application*. Docker Documentation. Recuperado el 27 de agosto de 2025, de <https://docs.docker.com/language/dotnet/>

Microsoft. (2025, 22 de julio). *Empezar a usar servicios web en Exchange*. Microsoft Learn. Recuperado el 27 de agosto de 2025, de <https://learn.microsoft.com/es-es/exchange/client-developer/exchange-web-services/start-using-web-services-in-exchange>

Exclaimer. (s.f.). *Disabling Transport Layer Security (TLS) 1.0 or Secure Socket Layer (SSL) 3.0 causes Outlook Web App (OWA) signature deployment via Exchange Web Services (EWS) to fail*. Exclaimer Support. Recuperado el 27 de agosto de 2025, de <https://legacy.support.exclaimer.com/hc/en-gb/articles/360024742632-Disabling-Transport-Layer-Security-TLS-1-0-or-Secure-Socket-Layer-SSL-3-0-causes-Outlook-Web-App-OWA-signature-deployment-via-Exchange-Web-Services-EWS-to-fail>

Microsoft. (2025, 22 de julio). *Authentication and EWS in Exchange*. Microsoft Learn. Recuperado el 27 de agosto de 2025, de <https://learn.microsoft.com/en-us/exchange/client-developer/exchange-web-services/authentication-and-ews-in-exchange>

Microsoft. (2025, 12 de agosto). *Autenticación Kerberos con HttpClient en Linux y Windows*. Microsoft Learn. Recuperado el 27 de agosto de 2025, de <https://learn.microsoft.com/es-es/dotnet/core/extensions/http-client-kerberos>

Microsoft. (2025, 22 de julio). *Cómo: Trabajar con elementos del buzón de Exchange mediante EWS en Exchange*. Microsoft Learn. Recuperado el 27 de agosto de 2025, de <https://learn.microsoft.com/es-es/exchange/client-developer/exchange-web-services/how-to-work-with-exchange-mailbox-items-by-using-ews-in-exchange>

10.5 Historial de Versiones (Changelog).

Versión	Fecha	Descripción de Cambios
1.0.0	2025-08-27	Versión inicial del servicio. Funcionalidades SendMail y ReadInbox implementadas con HttpClient y Autenticación Básica.